Migrating stateful containers using an autonomous collector

Kjetil Madsen Knutsen

Thesis submitted for the degree of
Master in Applied Computer and Information Technology - ACIT
(Cloud-based Services and Operations)
30 credits

Department of Computer Science
Faculty of Technology, Art and Design

Oslo Metropolitan University — OsloMet

Spring 2021

Migrating stateful containers using an autonomous collector

Kjetil Madsen Knutsen

Migrating stateful containers using an autonomous collector

http://www.oslomet.no/

Printed: Oslo Metropolitan University — OsloMet

Abstract

With the growth of cloud computing and Infrastructure as a Service, and the need to keep costs low many companies tries to utilize all their available resources as much as possible. Today, there is not currently any solution that allows you to move services based the available resources.

This thesis compares a machine learning approach to create a autonomous collector against using a round robin approach. It creates a solution on how to live migrate containers using these types of selecting what host to visit next, and to select where the containers should be moved to.

Using our solution, we have found a way to visit the servers with the most change on the most, and visit those with the least change less. During our experiments, we have gotten data that supports our algorithm, and solution. In a run of 180 iterations, 71 of the visits where on the host with the most change.

Key words: containers, autonomous, cloud computing, container migration, criu, podman, machine learning

Acknowledgments

I wish to show my gratitude to my supervisors, Associate Professor Hårek Haugerud, and Professor Anis Yazidi, without their assistance, patience, motivation, and involvement, this thesis would not be were it is today. I would like to thank them very much for their feedback, and guidance during this project. It has been a joy to be working with you.

I would also like to thank my co-student, and friend Anne Igeltjørn for her support throughout the thesis, especially acting as a "sounding board" in the later stages.

Thanks to Oslo Metropolitan University for offering this Masters degree program, and giving us this chance.

Lastly, I would like to thanks my parents, and the rest of my close family for not only their support this semester during the thesis, but also for all their support during my bachelor and master studies.

Knutsen, Kjetil Madsen

May 14, 2021

Oslo, Norway

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the growth of Infrastructure as a Service (‘Global Infrastructure as a Service (IaaS) Market 2021-2025: Market is Poised to Grow by $136.21 Billion, at a CAGR of 27% - ResearchAndMarkets.com’, 2021), many companies have started to move to the cloud. Often this leads to having many servers to monitor the different resource usages.

As Infrastructure as a Service can tend to be expensive, especially if you need higher end specifications or need many of them, you usually want to keep a record of what usage you have and turn off services on servers that you do not need or move services to a common server to have less servers up and about.

Today, for this reason many people use containers to keep their services on, as this allows you to keep many services side by side, without them interfering with each other. Of course this is as long as you have the available resources.

What we propose is a framework to respond to some of those problems. Mainly the migration of containers based on metrics from monitoring your services.

We have however chosen to do this in a way that allows you to not need any containers that collect metrics 24/7, but instead send a scout that will come collect, depending on former resource usage of said server. In this we have decided to see if using machine learning to control where the scout goes will improve the efficiency of the scout compared to running it in a round robin approach.

1.1 Problem Statement

P1: To what extent can machine learning be utilized to create a autonomous collector, that can be used for the migration of live containers.

To what extent can machine learning be utilized refers to if a machine learning algorithm can be utilized to improve a collectors visits compared to a round robin.

to create a autonomous collector refers to designing, and implementing a autonomous collector based on research, and earlier work.
that can be used for the migration of live containers refers to migrating containers from host to host in a automated way.

based on data from a autonomous collector refers to using data that are being collected using a collector, to move containers to the most suitable location.

1.2 Chapter Outline

1. Introduction:

This chapter will introduce us to my motivation, to the problem statement, and will show the papers outline.

2. Background:

This chapter goes through what a container is, what checkpointing and restoring is, and will go through different papers in relation to the paper.

3. Approach:

This chapter will go through how we are going to forward to solve our problem statement.

4. Results:

This chapter will go through, the results from the different tests both preliminary tests and the main tests.

5. Discussion:

This chapter contains our afterthoughts on how the project went, what we have seen that is working, and what could be improved upon.

6. Conclusion:

This chapter will go through how the project has gone, what we mean we have succeeded in, and what the solution in the end contains.

7. Future works:

This chapter will go through what we did not have the time to do, and what can be expanded upon on a later date.

Chapter 2

Background

2.1 Containerization: An introduction

In this section I am going to go through what a container is, and why we use them.

2.1.1 What is a container

A container is a way of packaging different services and its dependencies in a standard package. It allows you to run the services reliably from one environment to another, as it comes pre-configured ready to work.

This is done so that any container can be used on as many platforms that you can without changing anything. Which means you can setup an container that you are able to use on Linux distributions, but also on Windows servers.

It also allows you to deploy your software faster than previously, as it is less configuration that you have to do yourself, and that many containers come ready out of the box. It allows you also to easily remove and redeploy a service when necessary.

As said earlier, a container packages everything you need to run a container. That includes code, run time, system tools, system libraries, setting and configurations. These containers usually come shipped as a image file, which many container managers use, including Docker and Podman.

Most container managers uses the container standard OCI, Open Container Initiative. This means that most containers will work the same on different managers. So, you are able to use the same images on both Docker, Podman, and other container managers.

One upside of using containers is that in addition to what we have talked about, is that containers are completely isolated from each other, unless you want them to know of each other. This means that it is only the host operating system, and the container manager that would know of all the containers currently running.

This is something which does not only have a positive impact when it comes to security, but also it is something which allows us to have setups which were not available for us before. Such as running conflicting software side by side on the same host.

Now you would instead just run them on one container each. It also allows you to run multiple instances of the same software, spread again on multiple containers, where you route the traffic to different ports. So now, you can for example easily host multiple web servers on the same host.

Another thing that you can use containers for is to run larger systems, which you then split into multiple parts, containers instead. An example here is the radio software, “Azuracast” (AzuraCast, n.d.), which launches

six containers for its program.

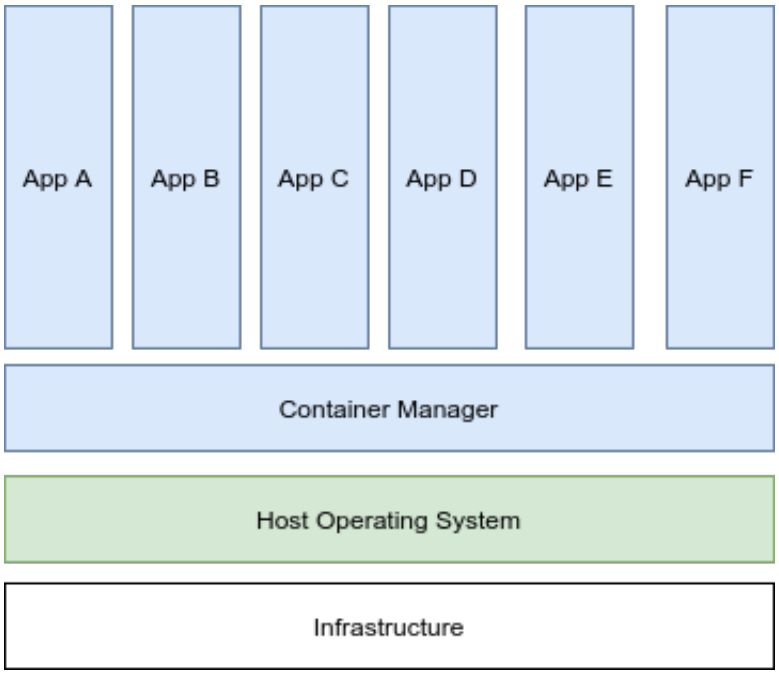

Figure 2.1: How an host running containerized services looks.

In the figure above, we are looking at how containerized applications works. On the bottom of the picture, we can see the first layer, which is the infrastructure. By infrastructure we here mean, the physical hardware of the machine.

Either if you have your own physical server, or if you rent server space. On the next layer, we have the host operating system. This is the main operating system that is installed on the physical hardware, either be it any Linux distributions, windows, macOS or other operating systems

On the next layer we have the container manager. This is the software which controls our containers, and here are multiple to choose from. Some of the known ones are, Docker and Podman. In this project, we have chosen to use Podman specifically.

Lastly we have the application, which come on the top layer. Here is any applications that we want to run in a containerized form. So any container, be it a web server or operating system itself, will come here.

2.1.2 Why use containers?

So, while we talked a little about why you can use containers, and some specific examples, lets try to talk a little more in-depth. So, what are the main upsides to run containers? One thing is that containerization is more lightweight than for example virtualization (Baeldung, 2020).

In virtualization you virtualize at the hardware level, and then run full operating systems on top. This is something that makes virtualization more heavy, and use more time. For containers, they are more lightweight because you share the OS kernel with the host operating system and can use much less memory compared to booting an entire operating system.

Another upside is the ability to run separate and consistent environments that are isolated from other applications. Containers include all the dependencies they need inside of their own container.

When it comes to isolation, it is not only that a containers are isolated from each other on a file level, but also when it comes CPU, memory, storage and network resources. All of these are virtualized on a os-level.

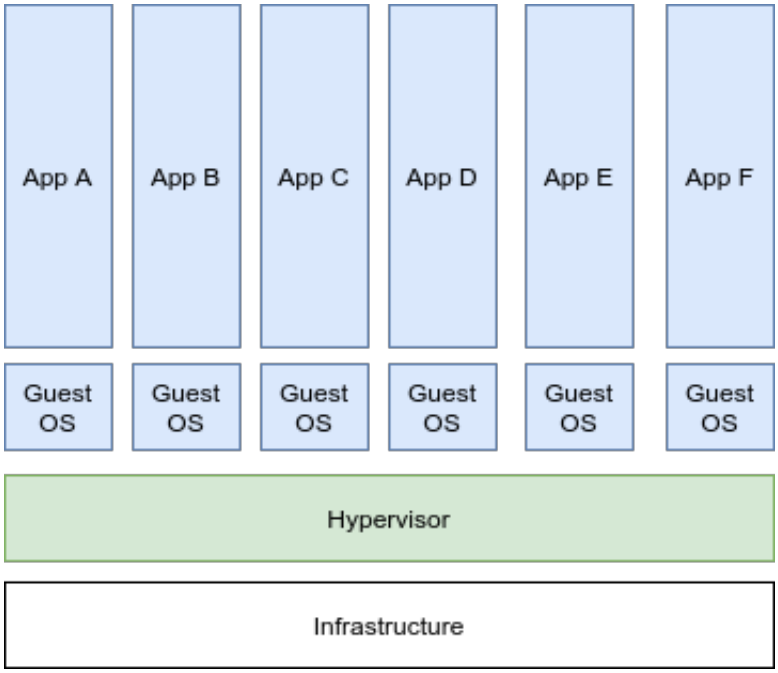

Figure 2.2: How a machine running virtualized application looks.

In the figure above, we can see what a virtualized machine would look like. Here we see that still, like earlier we have our infrastructure at the bottom. Next we have the hypervisor instead of a main operating system. On top of the hypervisor we have guest operating systems, which then run the applications that you want.

This brings us to another point, the ability to run a container anywhere, this especially combined with CRIU is powerful. Since a container is having all its dependencies already, and is packed in the way it is, means that you can run the same containers, from everything from Linux, to Windows as long as you have a container manager installed which supports it.

2.1.3 Container types, and standards

When it comes to containers, there are many different software that you can use to run containers off. These we normally call “Container man-

agers". Some of these are Docker , Podman, LXC, and OpenVZ.

In this paper I am mainly going to talk about Docker and Podman which supports OCI, Open Container Initiative. Of all those container managers that I mentioned, all of the are also supporting check pointing and restoring using CRIU, although to different degrees.

In this paper the main focus is going to be on the containers which use the OCI standard. The Open Container Initiative is an attempt to create an industry standard around container formats and run times. OCI was started in 2015 by Docker, and other leaders in the industry. Currently it has two specifications, the run-time spec, and the Image specification.

When the project was first started, we started by using Docker together with CRIU. To able to use Docker's implementation of check pointing, we had to use an outdated version, namely version 17.03 together with CRIU 2.6. This was done running on a virtual machine running Ubuntu 16.04.

Also, as check pointing in Docker is seen as an experimental version, this feature does not always work, and has to be enabled. It were not possible to use later versions of Ubuntu.

While we got the basic functionality up and working with Docker in Ubuntu, and had CRIU's main example working, not every container that we tried was functioning correctly. Some had problems check pointing, while some had problems restoring.

To be able to mitigate this, but also not lose any progress it was decided to move over to Podman, which not only uses almost the same syntax as Docker, but it also supports the same OCI standards. That meant we

could continue to use the same container images as previously, and also the same scripts, with some small modifications.

After having changed to Podman running in Fedora 33, instead of Docker running in Ubuntu 16.04, we found out that Podman's implementation functions better, and was better implemented as the feature was better developed. Here we also did not have any problems running our containers, as we found out earlier through our limited research.

2.2 Checkpointing & Restoring

Checkpointing and restoring is the act freezing a container, and saving all the process of a specified container into a checkpoint. This checkpoint can then be used later to restore said container either on the same host or another.

After it has been restored it will continue off, where it had left of when it had been checkpointed. This means that if you use this to checkpoint and restore an counter, the counter will first be checkpointed where it was when first running, and then restored on its latest state.

So that means that it will continue to count from where it was last. When you do the checkpointing, you save all the processes and information either to a folder or to a compressed file.

With this we are to use it for different use cases, like live migration of containers, seamless kernel upgrades, process duplication where you would checkpoint a container so you can launch a second instance of it, you are able to take snapshots of apps, and more. (Al-Dhuraibi et al., 2017)

Depending on what you want to do, one or the other option might be better for your use case. As stated with live migration, check pointing and restoring does not only enable you to restore it on the same host, that it was check pointed on, but it is also possible to transfer it to another host and restore it there.

Here is where the ability in Podman, to checkpoint to a compressed file comes in handy. This use of check pointing, transferring and restoration on another host is what this paper is going to focus the most on.

2.2.1 CRIU: Checkpoint/Restore In Userspace

Checkpoint/Restore In Userspace, or CRIU as it is also called, is an open-source software tool based on Linux with the ability to checkpoint and restore containers.

It's functions is currently integrated into multiple container managers, including Docker, and Podman. CRIU supports many Linux distributions, thereby including many of the popular ones such as Ubuntu, Fedora, Arch, OpenSUSE and more ('Packages - CRIU', n.d.).

CRIU started as a project of the company Virtuozzo and grew there with the help of its community. It started as a way to do live migration of OpenVZ containers but has in later times grown.('CRIU', n.d.)

The initial version was presented in July of 2011, and had its first stable release on the 23rd of July 2012. As of this writing, the latest stable release, version 3.14 was released on the 29th of April 2020. The software is distributed using the GNU GPL v.2 license. Its code can be found on

GitHub. (Checkpoint-Restore, n.d.)

2.3 Learning Automata

For the autonomous part of this project, we are going to use a Learning Automata to choose which host that it visits next. In this section, we are going to go through what a learning automata is.

2.3.1 What is an Learning Automata

A Learning Automata is a type of machine learning algorithm, that has been around since the 1970's. This algorithm uses their past experience, to choose what their next action is going to be. An learning automatas actions, are chosen by a set of probabilities.

In our case, they way that we use our learning automata is to feed it with the absolute difference in memory available. From this it, it calculates the probabilites for the different hosts, and chooses one of them to be the next host visited.

2.4 Learning Automata based Polling

In this section we are going to go through, how our selected learning algorithm works and functions.

2.4.1 Learning with barriers

As stated by (Yazidi et al., 2020) instead of having the limits of the probability space to be zero and unity, we are going to work with a constraint that says that no probability can have a value lower than a specified level

of p_{min}, or have a higher value than the specified level of p_{max}.

According to (Yazidi et al., 2020), the action choosing probabilities, which usually move proportionally between zero and unity for the L_{RI} scheme, are now going to move towards $pmin$ and $pmax$ instead. This minor change is renders the scheme to be ergodic, making the analysis also to be correspondingly distinct from that of the L_{RI} and simillar schemes.

(Yazidi et al., 2020) said that to be able to do this, we are forcing a minimal value p_{min}, where $0 < p_{min} < 1$ for each selection probability x_i, where $1 \leq i \leq r$ and r is the number of actions. From this the maximum value any selection probability p_i, where $1 \leq i \leq r$, can achieve is $p_{max} = 1 - (r-1)p_{min}$. This is going to happen, when other $r-1$ actions take their minimum value p_{min}, while the action with the highest probability takes the value p_{max}. Consequently, p_i, for $1 \leq i \leq r$, will take values in the interval $[p_{min}, p_{max}]$.

According to (Yazidi et al., 2020) when continuing with the formula, let $\alpha(t)$ be the index of the chosen action at time instant t. Then, the value of $p_i(t)$ is updated as per the following simple rule (the rules for other values of $p_j(t), j \neq i$, are analogous):

$$
\begin{aligned}
p_i(t+1) &\leftarrow p_i(t) + \theta r_i(p_{max} - p_i(t)) \\
&\text{when } \alpha(t) = i \text{ and} \\
p_i(t+1) &\leftarrow p_i(t) + \theta r_i(p_{min} - p_i(t)) \\
&\text{when } \alpha(t) = j, j \neq i \text{ and}
\end{aligned}
$$

where θ is a user-defined parameter $0 < \theta < 1$, typically close to zero. Further, r_i is a reward function indicator defined by the amount of change.

2.4.2 Summary of how the learning automata works

To simplify, the algorithm works in the way that takes a positive value, checks if it is larger than the max value that they currently have, and then puts it through a formula that calculates the probabilities. What is interesting about our formula, is that we are setting a minimum and a maximum probability, that is lower than the range.

This creates barriers that stops a host from reaching 100 % probability. This means that there is always a chance for every host to be visited, and that the algorithm is never going to be stuck on one, forever.

2.5 Related Works

There are many articles which touch upon the topic of containerization and the usage on CRIU: Checkpoint/Restore in Userspace. The following are some of them.

The article “Efficient service handoff across edge servers via docker container migration” (Ma et al., 2017) is an article that present an service handoff system, which they claim are able to seamlessly migrate services to the closest edge server.

This is done via the use of Docker container migration. They have proposed a migration method which leverages layered storage to reduce file system synchronization overhead. They have implemented and tested their prototype using world product applications.

Another article is “Sledge: Towards Efficient Live Migration of Docker Containers” (Xu et al., 2020). The article talks about that modern cloud platforms that are on the large side, need live migration techniques with stateful workload on Docker containers that gives them the possibility to support load balancing, host maintenance and Quality of Service.

In their paper they present a live migration system that they claim are highly efficient, called Sledge. Their solution has shown that compared to state-of-the-art solutions, their solution reduces 57 percent of total migration time, 55 percent of image migration time, and 70 percent downtime.

The article “Migrating LinuX Containers Using CRIU” (Pickartz et al., 2016)is an article which talks about how process migration is one of the most important techniques in modern computing centers, as it enables the implementation of load balancing strategies and eases the system administration.

They talk about that migration is usually accomplished by the use of hypervisor-based virtualization, and that container-based approaches is an attractive alternative. In the article, they present a prototype implementation of a libvirt driver, enabling migration of Linux containers using CRIU.

The article “Live Migration of Docker Containers through Logging and Replay” (Yu & Huan, 2015)talks about that lightweight virtualization techniques has made virtual machines more portable, work more efficient, and easier to management. It presents an approach to logging and replay for live migration of containers, specifically Docker.

With their approach they have been able to reduce not only the downtime, but also the total migration time. This have they been able to do through

the use of base images, an NFS server, and log files that gets replayed on the image.

The last article “Checkpoint and Restore of Micro-service in Docker Containers” (Chen, 2015) talks about the rapid adoption of micro-services, and talks about that it is imperative to build a system to support and ensure high performance and high availability for micro services.

The article addresses about the ability to checkpoint, restore and using live migration tools. The article goes through a proposed approach, namely using CRIU to checkpoint Docker containers.

Afterwards it also had an evaluation on the performance of the check pointing, and restoring, where it presented how much time it took to checkpoint and restoring it, depending on how large the container was.

The article ”Adaptive Monitoring: A systematic mapping” (Zavala et al., 2019) is a survey that is written by Edith Zavala, Xavier Franch, and Jordi Marco. The survey is a mapping survey about the current state of art of adaptive monitoring.

It does this by specifically, by identiifying the main concepts in the topic, determine the demographic characteristics of studies published in this topic, identify how adaptive monitoring is conducted and evaluated by different approaches. In the survey they have evaluated 110 studies, organized in 81 approaches.

The article ”Self-adaptive cloud monitoring with online anomaly detection” (Wang et al., 2018), is an article that talks about the monitoring of cloud computing systems.

They address the issue of having enourmous and complex sturctures of cloud computing which gives a significant performance overhead, which also increase the difficulty of analyzing useful information. To address this problem, they are proposing a self-adaptive monitoring approach for cloud computing systems.

2.6 Papers mentioned in this chapter

Year	Title	Author	Venue	Topic
2017	Efficient service handoff across edge servers via docker container migration	Lele Ma, Shanhe Yi, Qun Li	SEC '17: Proceedings of the Second ACM/IEEE Symposium on Edge Computing	Container Migration
2020	Sledge: Towards Efficient Live Migration of Docker Containers	Bo Xu, Song Wu, Jiang Xiao, Hai Jin, Yingxi Zhang, Guoqiang Shi, Tingyu Lin, Jia Rao, Li Yi, Jizhong Jiang	2020 IEEE 13th International Conference on Cloud Computing	Live Migration
2016	Migrating LinuX Containers Using CRIU	Simon Pickartz, Niklas Eiling, Stefan Lankes, Lukas Razik, Antonello Monti	High Performance Computing	Migration
2015	Live Migration of Docker Containers through Logging and Replay	Chenying Yu, Fei Huan	Proceedings of the 3rd International Conference on Mechatronics and Industrial Informatics	Live Migration
2015	Checkpoint and Restore of Micro-service in Docker Containers	Yang Chen	Proceedings of the 3rd International Conference on Mechatronics and Industrial Informatics	Migration
2019	Adaptive Monitoring: A systematic mapping	Edith Zavala, Xavier Franch, Jordi Marco	Information and Software Technology	Adaptive Monitoring
2018	Self-adaptive cloud monitoring with online anomaly detection	Tao Wang, Jiwei Xy, Wenbo Zhang, Zeyu Gu, Hua Zhong	Future Generation Computer Systems	Adaptive Monitoring

Table 2.1: Overview of articles that are mentioned in this chapter.

2.7 Summary

In this chapter, we have gone through what containers, and containerization is compared to virtualization, we have gone through what it means to checkpoint and restore containers using CRIU, and what it requires.

We have also gone through, what a learning automata is, including explaining how the algorithm that we are planning to use works. In addition to that, we have also gone through papers that touch upon the theme of containerization and checkpointing/restoring.

Chapter 3

Approach

In this chapter we will go through our project's objective, the experiments that we are planning to have, and what scripts that our solution contains.

3.1 Objective

As stated earlier in this paper, the aim of this project is to design, implement, and test a framework for migration of stateful containers, based on data from a autonomous collector.

To be able to solve our problem statement, we require infrastructure that run our solution. For this the project is mainly planning to use OsloMet's own cloud service ALTO, which is currently the largest OpenStack deployment in Norwegian higher education, with 1024 cores, and 4TB of ram according to the university's own site ('Autonomous Systems and Networks (ASN)', n.d.).

In addition to ALTO, we are also planning to use public cloud services, such as Amazon Web Services, and Google Cloud Engine.

3.2 Experiments

As for the experiments that we are planning doing, we are currently planning to perform three main experiments. The first one is going to be a basic checkpoint and restore operation from one host to another. This will give us data on the performance of both the transfer, and its speed, but also how check-pointing and restoring a container continuously went.

As for the second experiment, we will testing our collector script without our learning automata. This collector script does mainly x things. Firstly, it sends our collector which consists of two Podman containers between the hosts, which are specified in the hosts.txt file.

On each hosts, it will restore itself, use Telegraf to collect metrics, and then checkpoint itself. It will then go to the next host, and repeat itself.

Afterwards it will send itself back to the host that the script initially was started from. From here it will first restore itself, then it will making queries to the database, and based on those queries it will give out the ip address of the host that has the most available memory. This will be repeated x times to have a good dataset.

The last experiment is going to be very similar to the second one. This one will still test the collector script, but this time with a learning automata. This means that after running the initial rounds with a round robin approach, it will then go over to using the learning automata to decide what host it will visit next.

To be able to use the learning automata, with the collector, there are a couple of things that are added in addition. The first thing is a calc script

which calculates the difference in available memory between to runs on the same host.

This data is then feed to the learning automata algorithm, which then outputs a index numbers, which corresponds to which host is going to be visited next. In addition to this, we will also need to create a timestamp, which is created before each host is visited. This gives us the ability to know exactly what data to pick out, so that we can get the correct data about available memory.

To be able to do these experiments, we are using Podman together with CRIU: Checkpoint/Restore in Userspace. Our experiments will also be running on Fedora 33, with Podman v3.1.0 and CRIU v1.15. In the addition to these our scripts contains the use of rsync, scp, sed, and awk among others. These are programs witch normally are preinstalled.

3.3 Scripts

Our finished solution is going to contain these main files:

1. collector.sh
2. trigger.sh
3. LA_with_barriers.py
4. hosts.txt
5. Telegraf.conf

In addition to these main files, we will also be creating files to save different information. These will be in its own folder, bin. As of now, this folder is going to container a timestamp file, a counter file, a calc file, a ip file, and three results files.

1. Collector.sh This file is the main bash script which will run our solution. This file is what decides what to run on each host. It is also going to do the necessary queries needed, including finding what host has the most available memory, but also the queries need for calculating the difference in memory.

The script will also do the calculations for finding the difference in memory. Lastly this script is going to do the migration of the main application.

2. Trigger.sh The trigger.sh script is going to be a simple script which is run every x minutes by a cron job. This script will then check for example the available amount of ram on the current host, where the main application is. If the memory is below a certain threshold, it is going to start the collector.sh script.

3. LA_with_barries.py Our LA_with_barries.py is our learning automata algorithm. This is what calculates what host to go next to, after the collector script has run its initial runs. This algorithm takes in the difference in memory, and gives out what index number to visit next. From this we can know the next ip to visit.

4. Hosts.txt Our hosts.txt file, is the file that decides what hosts can be visited and not. This file is a simple list, with the ip addresses of our hosts. For the initial rounds, this file is used as a round robin, but later it is only used for checking which ip, the index from LA_with_barriers refers to.

5. Telegraf.conf The Telegraf.conf file is our Telegraf config. This is the config that Telegraf is going to use when starting up on each host. This also decides what is being recorded by Telegraf. As of now, that is memory.

All of the scripts, are available in the Appendices. The project is also available at github. (Knutsen, 2021)

Chapter 4

Results

In this chapter we will have a look at the different migrations tests that have been done in this paper. First we will start of with the Preliminary Migration Test, which will give us an idea for how much time a migration takes, and how reliable it is.

4.1 Preliminary Migration Tests

For the preliminary migration test, we started with setting up two virtual machines that were going to be our test machines. In this test we will call them “A” and “B”. On both virtual machines, we are running Fedora running on version 33.

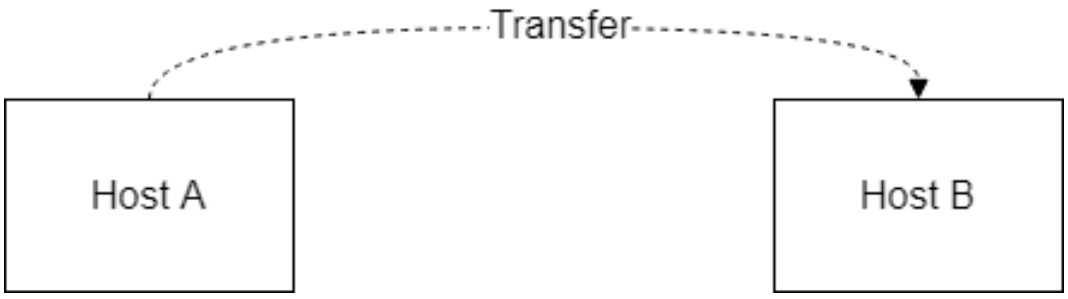

Figure 4.1: Shows transfer from host A to B

To be able to run the script, A has ssh access to B through ssh-keys. Both of the machines have Podman and CRIU installed on them. As for Podman, the version that is being used is version 3.0.1, and as for CRIU we are running version 3.14, which is the latest stable version as of this writing.

As for the script itself, it mainly consist of three parts. The first part is the check pointing of the selected container, and the export of the checkpoint tar.gz file. This is done through Podmans container checkpoint command.

As for the second part, we are going to transfer the tar.gz file that we created in the last step. This is done through the use of rsync. Rsync was chosen as it is possible to log its output to a file. The file is transferred then from A to B.

For the last step, we are going to restore and start the container. Here we are restoring the container on B, using Podman container restore. Here we are restoring the container using the file that we transferred in the last step.

For all of the steps, we have added the time command in front, this so we could see how long it took for each part when running the script. All the parts are also setup to log its output to a log file. This is done using the tee command.

When testing, we have decided we wanted to run the script many times to be able to see if the tests were stable. This was done through an additional script. This script is mainly to set a loop that runs the amount of times that you want. Here we chose 25.

In addition to this, this script also resets the environment ready for the next test. This includes starting the container again on A, so it could be check-

pointed, and removing the container on B after the test was done.

As for the container that was chosen to be used in this preliminary test, we choose to use the Ubuntu container with the 18.04 tag. This container, is about 50MB large, and has been made no modifications on.

Test	Time Transfer (s)	Time Checkpoint (s)	Time Restore (s)	Time Total (s)	MB/s
1	0.651	2.064	1.724	4.439	17.35
2	0.296	1.920	1.709	3.925	17.32
3	0.342	1.953	1.642	3.937	17.21
4	0.349	1.927	1.609	3.885	17.40
5	0.338	1.821	1.639	3.798	17.32
6	0.338	1.982	1.687	4.007	17.36
7	0.289	1.966	1.673	3.928	17.32
8	0.300	2.007	1.684	3.991	17.38
9	0.285	1.913	1.638	3.836	17.39
10	0.353	1.987	1.691	4.031	17.36
11	0.348	1.892	1.663	3.903	17.36
12	0.293	1.942	1.768	4.003	17.35
13	0.316	1.853	1.659	3.828	17.20
14	0.342	1.820	1.657	3.819	17.28
15	0.349	1.925	1.655	3.929	17.40
16	0.336	1.930	1.665	3.931	17.34
17	0.274	1.844	1.673	3.791	17.30
18	0.300	1.925	1.593	3.818	17.29
19	0.301	1.946	1.604	3.851	17.37
20	0.285	1.867	1.614	3.766	17.22
21	0.320	1.771	1.738	3.829	17.23
22	0.303	1.945	1.637	3.885	17.19
23	0.285	2.065	1.628	3.978	17.33
24	0.329	1.966	1.624	3.919	17.35
25	0.337	1.946	1.721	4.004	17.31
Median	0.320	1.930	1.659	3.919	17.33
Average	0.330	1.927	1.664	3.921	17.32
Mode	0.285	1.925	1.673	3.885	17.32

Table 4.1: Overview of the 25 test

4.1.1 Experiment Results

After the preliminary tests, we have gotten some results. In this part, we are going to try to go through those numbers. All of the results can be seen in the table 4.1 on the page above.

As previously said, we ran the test 25 times, and in those tests we have recorded five things. The time it took to transfer the tar.gz file from host A to host B. The time it took to checkpoint the container on host A, the time it took to restore the the container, and start it, The total time the script took each time, adding all the previous together, and the MB/s rate that rsync took when transferring.

At the total time it took, we have times ranging from shortest time of 3.766 seconds, and the longest of 4.439 seconds. This means that largest difference between the runs was at 0.673 seconds. This compared to some of the tests that were done when first started, that had time difference up to 6 seconds, shows that the test were had less irregular times than earlier.

This can be because of a few different reasons. The first that mainly pops up, is that originally scp was used instead of rsync, but this was changed out because scp's progress bar turns off, when logging to a file. Another reason could be that in the early tests, the tests were done manually without a loop script.

This meant since the tests did not automatically start after each other, there could be a change on network traffic or other hardware differences. It is hard to be able to know for sure, as in the start we only recorded the total time, and not the part times.

For the total time, the average time ended up on 3.921, with the mode being 3.885 and median being 3.919.

For the checkpoint time, we can see that off the 25 tests, the shortest time was 1.771, and the longest time was 2.065. This means that the difference was the maximum at 0.294. This means that the checkpoint time held itself stable, and did not have any large changes throughout the tests. The average of the checkpoint times, ended up on 1.927, with the mode being 1.925 and median being 1.930.

For the restore time, we can see that the shortest time was 1.593, and that the longest time was 1.768. This means that the difference between them were at 0.175, which has an even smaller difference than the checkpoint times. The average was at 1.664, the mode at 1.673 and the median at 1.659.

For the transfer times, we would think that it would somewhat match up to to the MB/s. From the table, this seems to not always to be the case. For example, the shortest transfer has a MB/s rate of 17.30, but the longest transfer had a rate of 17.35 MB/s. From this we can see that that the longest transfer had the faster rate, even if it was only by a little.

The fastest transfer speed that was had, is at 17.40 MB/s, with the slowest at the 17.19 MB/s. This shows that the speed itself does not change much throughout the tests. The transfer time, also show this. Here the shortest time was at 0.274, and the longest was at 0.651.

From the table, we can also see that the longest time here, is also a outlier, with the next up being at 0.353 seconds. It can also seem to be just natural fluctuations in the transfer speed, as the MB/s speed did not change much,

and stayed from 17.1 to 17.4 throughout all the tests.

For the transfer time, the average was at 0.330, with a mode of 0.285 and a median at 0.320. For the MB/s speed, the average was at 17.32, with a mode of the same, and a median of 17.33

4.2 Collector using round-robin

For the second experiment I have decided to test our collector. The collector is what is sent between different hosts to collect metrics about said hosts.

The metrics are collected through the use of InfluxDB version 2.0.4 and Telegraf version 1.8. The first thing that the collector does is to check if any export files already exists, and if it does not it will start an instance of InfluxDB, and checkpoint it to a export.tar.gz file.

Then we come to the main part of the script which is a for loop which executes a number of commands on each host, which are specified in a hosts file. To be exact the actions which are done on each host are:

1. Creating the folder /home/fedora/MVAC if it does not exist.
2. Transferring export.tar.gz file to host
3. Transferring Telegraf config file to host.
4. Replacing IP in Telegraf config with the hosts, using sed.
5. Restoring influxdb container on host, using the export.tar.gz which was transferred earlier.
6. Start said container, if it did not want to start properly.
7. Start a instance of Telegraf container using the Telegraf.conf file that was transferred earlier.

8. Sleeps for x seconds, to give it time to collect metrics. 9. Removes Telegraf from host.
10. Checkpointing influxdb container to export.tar.gz file.
11. Transferring the newly created export.tar.gz file back.
12. Removes influxdb container.
13. Removes all volumes which are on the host.

After all of these actions are done on each host, we start by restoring the influxdb container on the main host. To do this, we first check if there is any influxdb container already running, and if so removes it.

It will then also remove all of the volumes. It will then start restoring the container using the export.tar.gz file which was transferred back earlier.

Lastly it will start the migrate.sh script. This script does two main things. The first one is to query from the InfluxDB container, to see which host is the most suitable. It will then pull out that's host's IP address.

Afterwards it will start to transferring the scripts, and the main container to said IP. It will also perform a cleanup, of the host it was on.

For the first run of the test, it was decided to run the test from a virtual machine on OsloMet's ALTO. In hosts file, we added three hosts addresses. The first one added is the same as the one the script is running on, a virtual machine on ALTO.

This is to good base to start with. The second host which was added a virtual machine running on Amazon Web Services, located in London, running a t2.micro vm.

For the third and last host, that one is also running on Amazon Web Services, but this one is located in Ohio instead, and as the previous one is also running a t2.micro vm.

As for what the three hosts have in common are that all of them are running Fedora Cloud 33, with Linux kernel 5.8.15-301.fc33.x86_64. All of them are also running Podman 3.1.0 and Criu 3.15.

In all the runs, the hosts are visited according to a round-robin schedule. As for the second run of the test, we ran the same test again, but now on 9 hosts, where three of them are located on ALTO. The other six are located on Amazon Web Services. Three in Ohio and three in London.

Experiment Results

As for the results, we are going to a look at the numbers that we got from the two runs that we did.

For the first run, we noticed something interesting when it comes to the transferring speed, but which also makes sense. Of the three hosts in the first run, host 1 had the fastest transferring speed at 57.7MB/s when transferring the export file.

This was much faster than what host 2 had at 7MB/s and host 3 had at 2.6MB/s. These differences has probably a lot to do with that the three hosts, were put in three different countries, Norway, England and USA.

This makes sense as, the distance between the transfers are longer for the last host compared to the two others, as you transfer the export file from the main host were the script is run to the selected host. In this case the

main host is located in Norway.

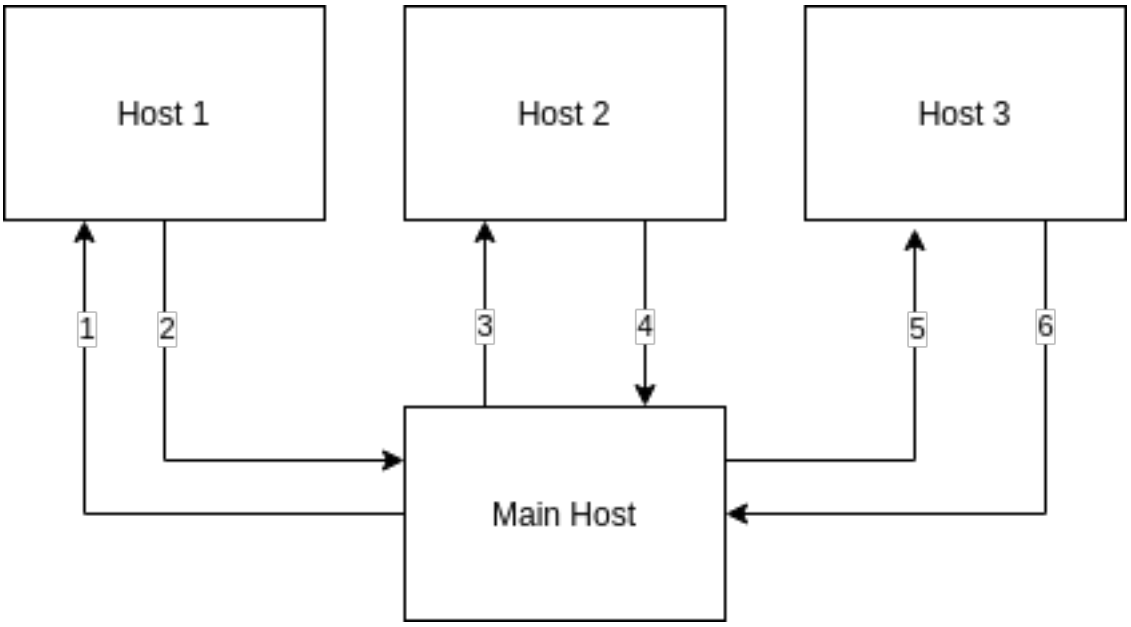

Figure 4.2: Figure of what order the host comes in.

The above figure shows how the transfer of the export file happens. However what made intrigued me even more, was the total time for each host. While host 1 and 2 where quite even at 55.71 seconds and 54.75s, host 3 ended up at 1m20.82s. This is very interesting as host 2 and 3 should run on equally similar hardware as both run on Amazons t2.micro instances.

The difference in transfer here, is also not big enough to have a that big impact.

For the second run, we decided to up the numbers of hosts from each location. Also note that this run was done at a different time of day, so the top transfer speed is for some reason higher here.

The second run shows about the same as the first one, where we can see

that the hosts that are located in Oslo have the highest speeds when it comes to transfers, and that Ohio has the slowest transfer speeds. This both in MB per second, but in total time for the transfers.

We also found some interesting things. For example, when first starting to run the script on AWS, we had some problems that the restoring of the InfluxDB would not work.

All were setup the same as the other hosts, the only exception were that the image that we first used had a kernel version that was newer than the other hosts. When we used an image which has the same kernel version, that problem was gone.

			Transfer: Export.tar.gz to host		**Transfer: Telegraf.conf to host**	
Run	**Location**	**Host**	**Speed**	**Time**	**Speed**	**Time**
1	Oslo	Host 1	57.7MB/s	00:00	40.2MB/s	00:00
1	London	Host 2	7MB/s	00:00	1.7MB/s	00:00
1	Ohio	Host 3	2.6MB/s	00:02	367.4KB/s	00:00
2	Oslo	Host 1	79.7MB/s	00:00	33.7MB/s	00:00
2	Oslo	Host 2	81.9MB/s	00:00	35.7MB7s	00:00
2	Oslo	Host 3	85.6MB/s	00:00	40.2MB/s	00:00
2	London	Host 4	7.7MB/s	00:00	2.1MB/s	00:00
2	London	Host 5	16.1MB/s	00:00	1.7MB/s	00:00
2	London	Host 6	8.4MB/s	00:00	1.5MB/s	00:00
2	Ohio	Host 7	3.7MB/s	00:01	484.1KB/s	00:00
2	Ohio	Host 8	2.9MB/s	00:02	404.9KB/s	00:00
2	Ohio	Host 9	3.3MB/s	00:01	352.2KB/s	00:00

Table 4.2: Overview of second experiment results

Table 4.2 above shows all the results for the two runs, in the second

experiment. Below is also the total time for each of the hosts that was run during the experiment.

Run	1	1	1	2	2	2	2	2	2	2	2	2
Host	1	2	3	1	2	3	4	5	6	7	8	9
Time	55.71	54.75	80.82	44.58	48.83	43.62	41.72	41.65	42.05	61.56	62.17	62.17

Table 4.3: Total times of hosts during second experiment

4.3 Collector with Learning Automata

In our third experiment, we are going to take a little different route. This time we are going to compare our collector with the learning automata versus what a run without it would look like.

When doing the run without the learning automata, we are going to do a simulation of a round robin. The reason as for why we are not actually running it, is that we want our recorded metrics to be as close to each other as possible. Since a run with 180 iterations takes approximately 4 to 5 hours, doing them each for itself is not possible.
When doing this test, we are going to be using three hosts, which we will be calling host 1 through 3. All of them are going to be hosted on Oslo-Met's ALTO Cloud. The main reason for this, is that we are currently not looking for transfer speeds or times in general, but what the results are for the hosts that have been visited. Of course, you would be able to perform the experiment on other cloud providers as well.

In this experiment our collector will run a set of iterations, this time 180 as specific earlier, and between each time it will choose one of the hosts to go to next. When running this experiment, we are going to generate variable ram usage on 2 and 3, while host one will not have any ram generated.

This is so if the learning algorithm will choose to visit the hosts with more change on, than not. This ram will generated using stress-ng. This is done with a stress.sh script, an example is shown below.

```
#!/bin/bash
while :
do
    stress-ng --vm-bytes $(awk '/MemAvailable/{printf "%d\n", $2 * 0.5;}' \
    < /proc/meminfo)k --vm-keep -m 1 -q -t 10
    sleep 15
done
```

As we said we are going to run the round robin as a simulation. With that I mean that we are going to record the available ram on the hosts, as if the collector using round robin was visiting. Below is an example script of how we have done that.

```
#!/bin/bash
while :
do
        sleep 10
        date=$(date +%s)
        mem=$(free -b | awk '{print $6}' | sed -n 2p)
        echo $date $mem >> ~/MVAC/logfile
        sleep 230\\
done\\
```

In the script above, what we have done is to run this in a loop as long as the collector script runs, as it both stops it and starts it. The first thing is that it sleeps for 10 seconds, this is the amount of time it takes before it records

any metrics, when you have first arrived on the host. Then we have the recording of unix time, and what the available ram is. This is then written to a file.

Lastly, we have that it sleeps for 230 seconds, this is the total amount of time that it takes before we believe that a round robin will appear again on this host minus 10, as we have calculated that a normal time usage is 80 seconds. To get the total amount of sleep time, we can then just time 80 with the number of hosts, which is in our case 3.

To be able to see if there is any difference in results, depending on how the Learning Algorithm is configured, we are going to do three different runs. In the first run, θ is going to be set at 0.1, at the second run it is going to be at 0.15, and on the last run it is going to be at 0.2

4.3.1 Experiment Results

As said earlier, we are going to do 3 runs, each with different values as θ.

For the first round, we are running with θ as 0.1. In Figure 4.3, we are able to see from the graph below, what probability each host have throughout the iterations.

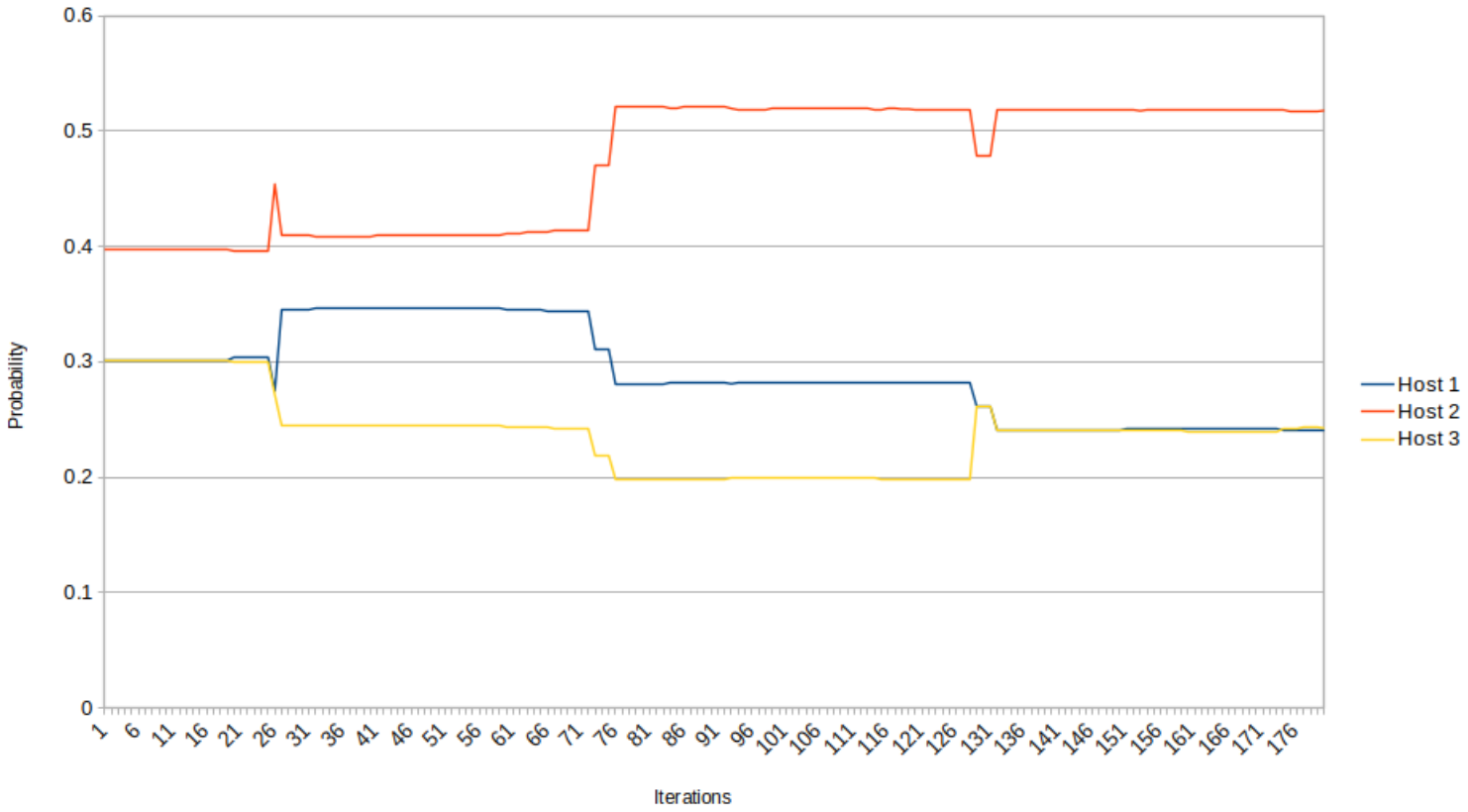

Figure 4.3: Probabilities in run 1

From the graph we are able to see that after the first iteration, that host 1 and 3, are at approximately the same probability, while host 2 has gone up to just below 0.4, ending at 0.398, while host 1 and 3 is at 0.301.

If we go further out into the graph, we can see that around iteration 26, that also host 1 and 3 is separating. We see that host 1 is gaining probability, while host 3 is losing. In the end though, we see that host 3 and 1 has found each other again, and is both at around 0.24, while host 2 is at 0.51.

This is something that makes sense, as the algorithm is suppose to visit those with much change, more. Here host 1 has been at a standstill with available ram, and host 2 and 3 is having ram usage generated throughout the run.

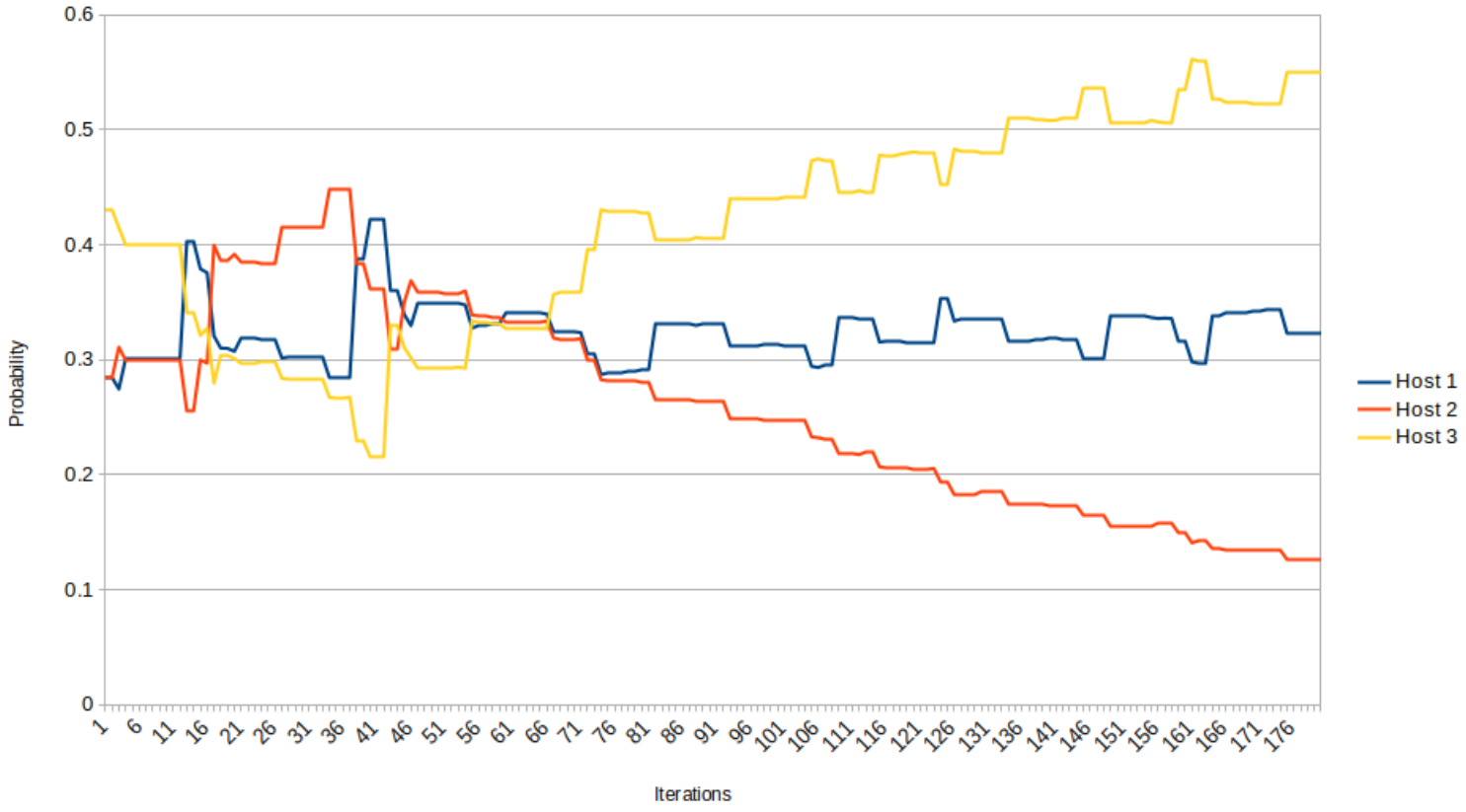

Figure 4.4: Probabilities in run 2

If we now look at the second run, in the graph at figure 4.4, we can see that got a bit of a different story. In this run, we have set the θ at 0.15. In this run, we can see in total that there is much more variation, and that the probabilities tend to not stand much at the same values.

We can see here that host 3 is now the one who takes the headstart, while host 1 and 2 stays behind. However, this changes a lot until around iteration number 67. From there on out, we can see that host 3 is skyrocketing up, while host 1 is trying to stay stable, and host 2 is going down.

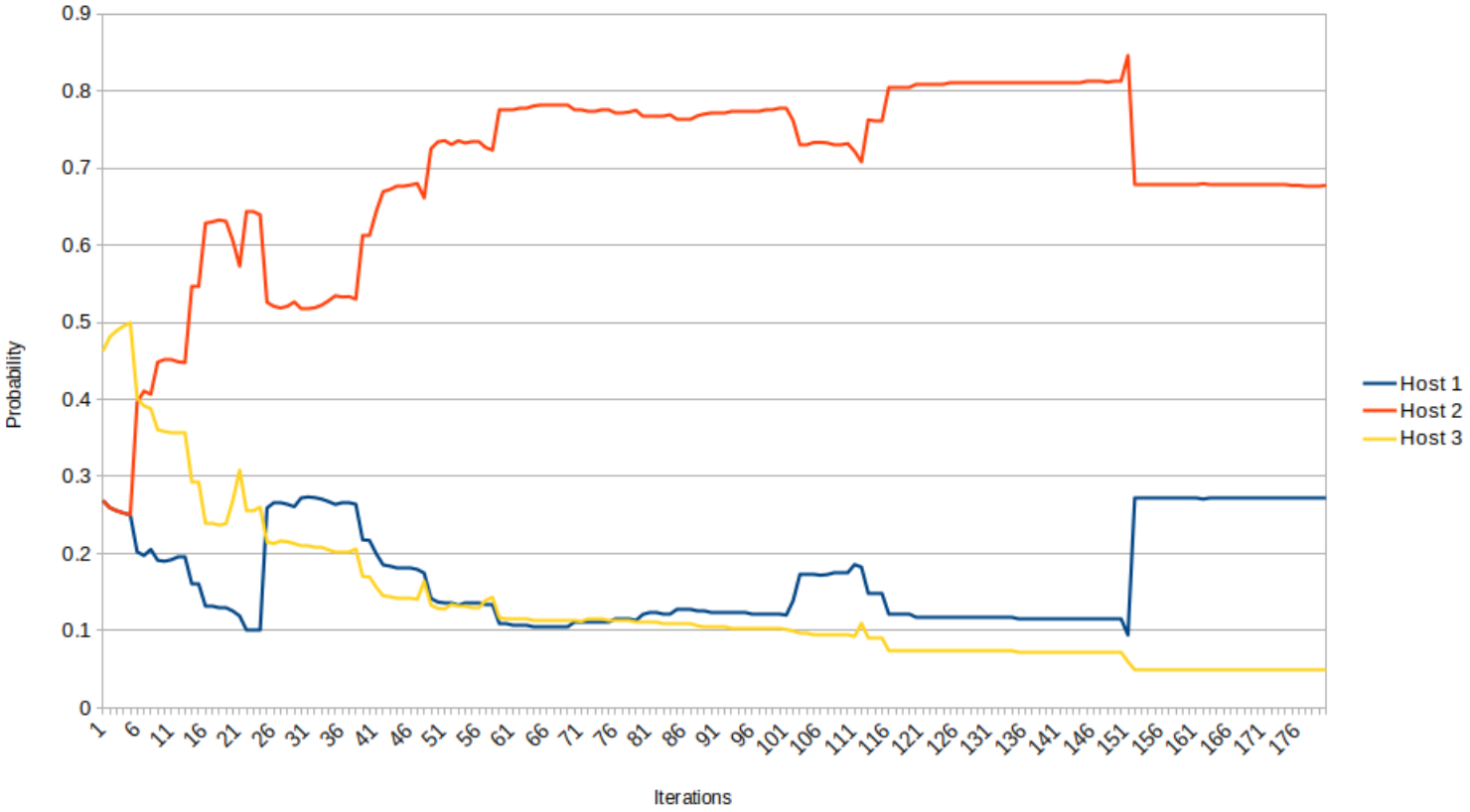

Figure 4.5: Probabilities in run 3

In the graph above at figure 4.5, we see what the run with θ as 0.2 looks like. What we can see here is that in the start host 3 is the one getting a high probability first. We can see that it keeps that not to long, being overtaken by host 2 around iteration 11.

From iteration 11 and outwards we can see that host 2 goes up in the beginning, and then flattens for a while, before coming to a peak and going back down a bit. For host 1 and 3 we can see that they are going downwards in the start before keeping fairly stable, but in the end host 1 goes upwards towards 0.3, and host 1 at 0.1.

What is interesting here is that host 1 has through all the runs, come in the middle place, or at the same value as the last one, while one of the ones

with memory generation has been having the highest probability each time.

Below in Figure 4.6 we can see the number of visits for each host, during each run. As we can see with run 1, that host 2 is the most visited with 89 visits, while host 1 comes at 59 visits and host 3 comes at 32 visits. This seems to mirrors the probabilities, very well.

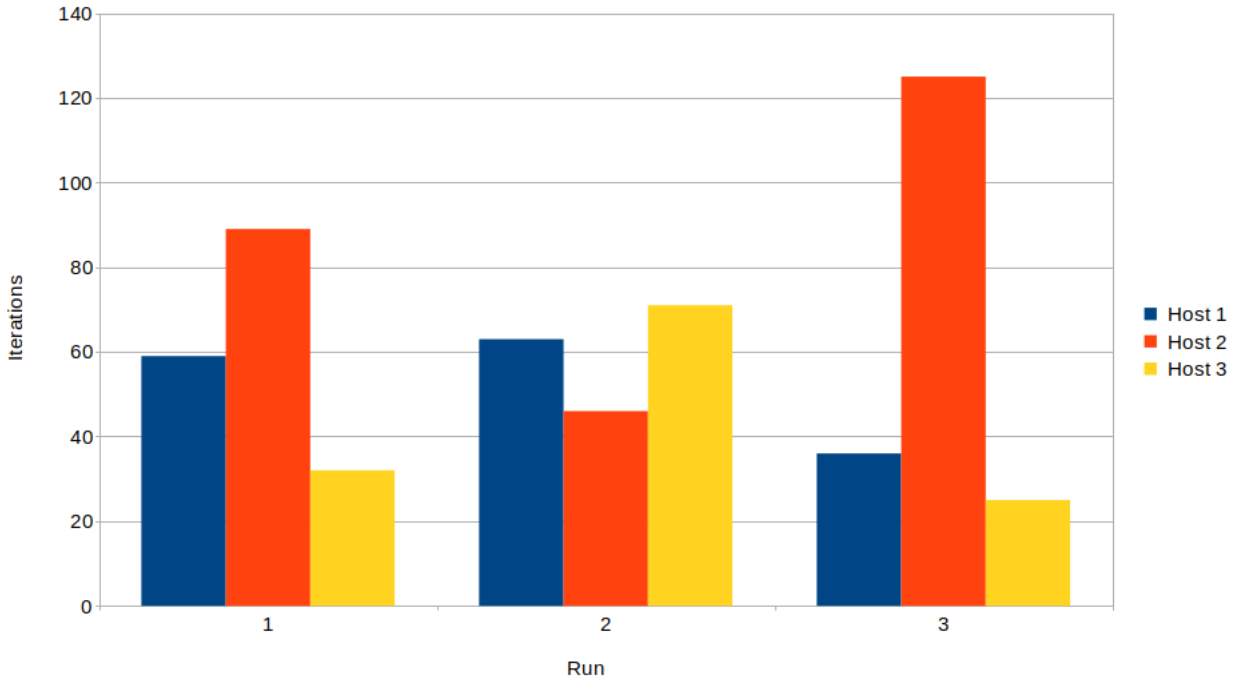

Figure 4.6: Overview of visits through the different runs

From what we can see, we can see that host 1 is visited a total of 59 times, while host 2 is visited 89 times, and lastly host 3 is visited 32 times to a total of 180 visits. Here we can see that the one host that were visited the most is host 2, as we know host 2 is one of the hosts which has the stress script generating memory usage.

What is interesting here, is that the second most visited is not the one with the highest memory generated on, however it is the one were we did not generate any usage. It will be interesting if this changes when θ is set to

0.15 instead.

When looking at the run with θ set at 0.15 we can see that it choose the one with the most memory change, but on the one with 0.2 θ, we can see that it choose host 2 instead. This might be because there was not enough memory difference between host 2 and 3.

If we look at the available ram throughout the runs, which we can see in Figure 4.7 below, we see interesting, which is that the average memory available when using the learning automata is always higher than the ones with the round robin. This confirms to us that, the learning algorithm is choosing the ones where the most change happens, and that makes them visit the ones with higher available ram more.

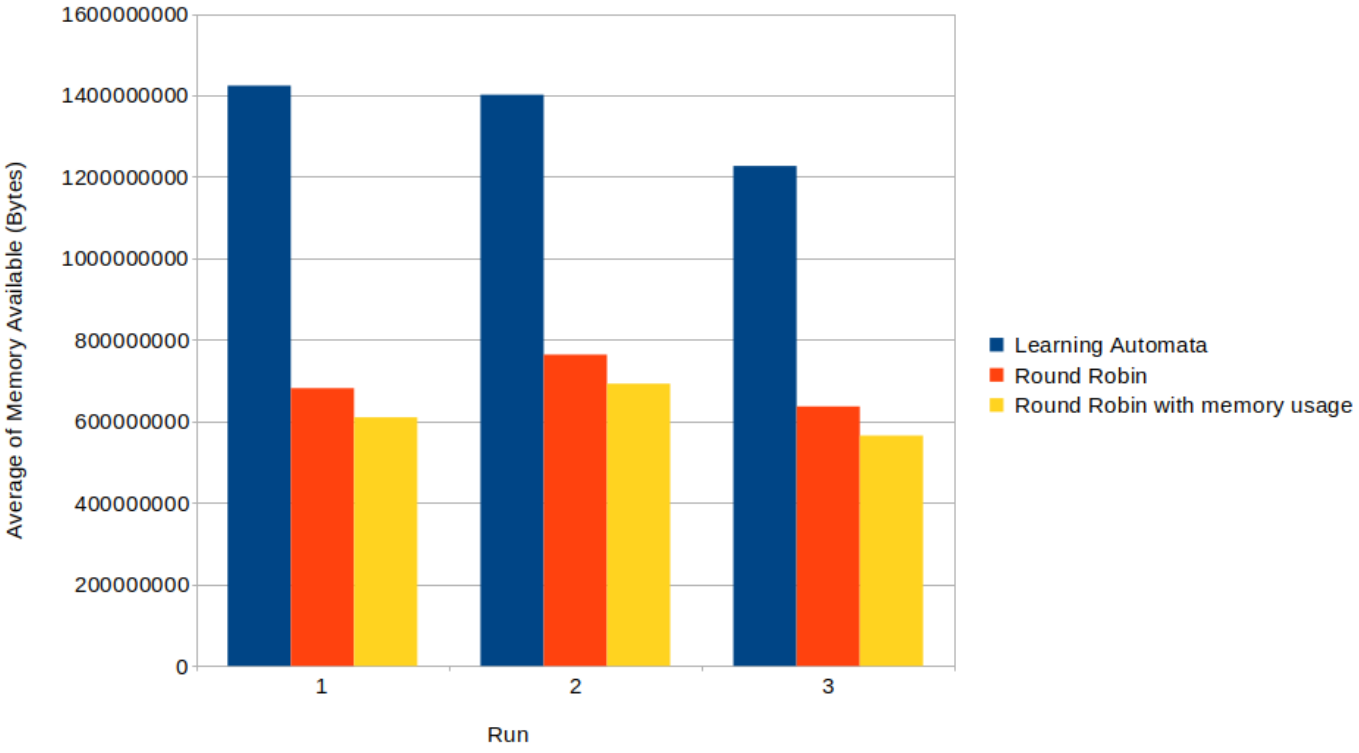

Figure 4.7: Overview of the averages for all three runs. Including both LA, Round Robin and Round Robin with memory usage.

Below in figure 4.8, is a screenshot of the last iteration in run 3. From it we can first see a couple of things. First is that the last iteration was run at host 2, which was at 172.16.0.70. We can also see the transfer speeds for the different transfers done, both to the host, and back to the host were the script is running. In addition to this we also see how long the process took on the host, which in this case was 50 seconds.

Lastly we see the restoring of the database on the main host, where we can see the database being setup, so we can access it via curl. In the end, we see the probabilities for the next iteration.

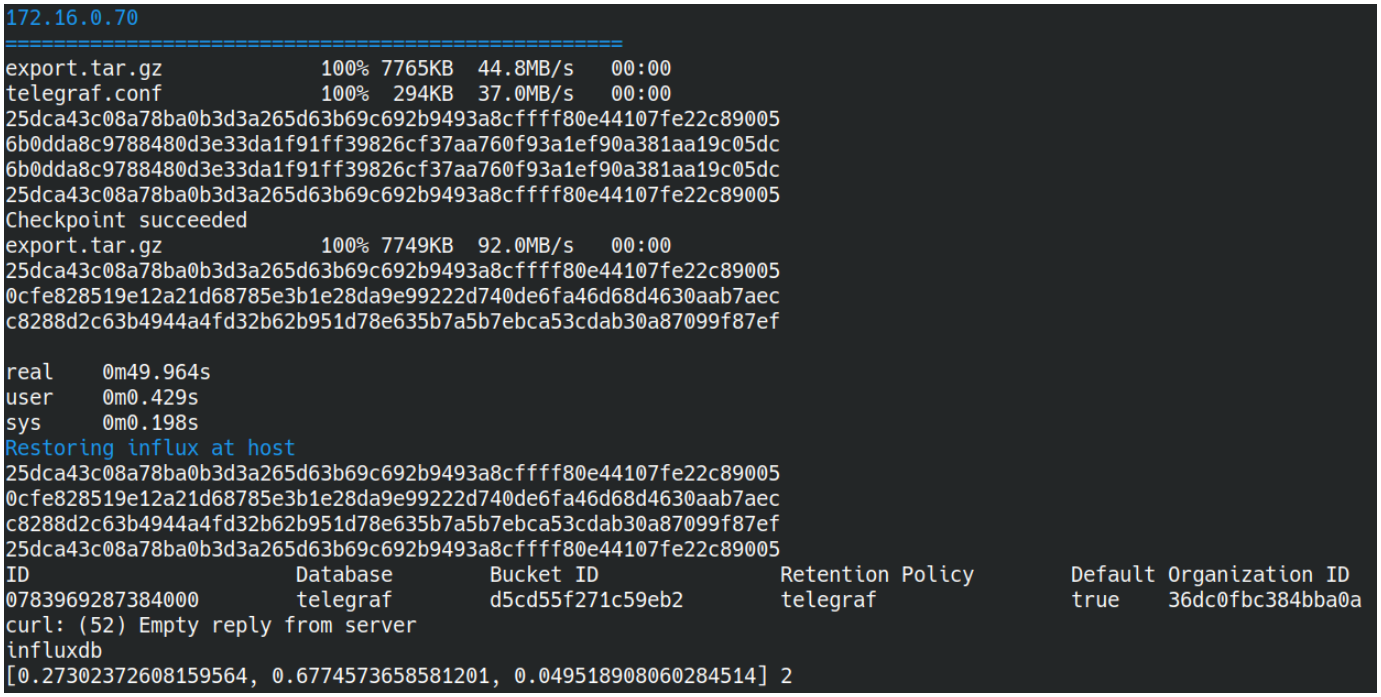

Figure 4.8: Screenshot from the running of experiment 3, run 3

Here we can see the different probabilities together with the number for which host it wants to go to. So here it chose to go to host 2 next, which has the probability of 0.6774573658581201.

[0.27302372608159564, 0.6774573658581201, 0.049518908060284514]
2.

4.4 Summary of Results chapter

To summarize this chapter, we have gone through three different experiments. In the first experiment we tested what it was like checkpoint and transfer repeatedly, between two hosts. For the second experiment we tested our collector script without the learning algorithm, and compared the transfer speeds between different locations.

Lastly, in experiment three, we tested our collector with the learning algorithm. In this test we ran it 3 times, with 180 iterations. Here we have compared the different probabilities against each other, and looked at the total average of the available ram, and total visits for each host.

Chapter 5

Discussion

In this chapter, an overview of the project will be presented. It will go through the different chapters as well, as a comparison with existing projects, and take up issues that we have encountered throughout the project.
When working with the project, we have gathered a good amount of data when doing the experiments. We have gathered this data through our experiments using our collector. This collector is entirely self-written, as because of that, we have to look to see if our results are reliable.

When looking at the results from the first two experiments, we can see that those experiments have repeatedly gathered data about transfer speeds from one host to another. The first experiment gathered specifically data from host A to B repeatedly, while experiment two did it in a round robin approach.
In the experiments, the main software that has been in use to do the transfers has been rsync, and scp. Using these widely used software, we can see that our experiment 1 and 2 results were relatively consistent and accurate.

What can cause a difference if someone tried to do the experiment again,

is network speeds. As it would be necessary to have virtual machines with similar networks and specifications, which might not be the easiest for everyone to acquire.

In the third experiment, we measure what hosts are visiting when using the learning algorithm compared to using round-robin. The results are here accurate when it comes to the learning algorithm, but less so on the simulated round-robin. The reason for this is that in the round-robin approach, we use the time it takes to visit one host when there is no memory usage. This is less accurate, as some of the hosts that were in use are having memory usage generated on them.

To be able to generate memory usage on two of the hosts, we have been using stress-ng. This makes it hard to calculate how much memory usage each have, and how long time the collector would use, as it would vary. This means that, while it is accurate initially, the simulated round-robin will get more inaccurate towards the end.

This is because of both varying memory usage and if anyone the iterations need to restart underway. This is probably still the most accurate way to do it. For example, if it had been done through the collector script instead, it would have gotten numbers from many hours previously.

In experiment three, we have put the learning automata up against a round robin approach. The reason for why we have used a learning automata instead of just running round robin, is so we can check if we can use machine learning to be more efficient about what hosts are visited and not. This is especially important if using many hosts.

From the experiment we have gathered some interesting data. In the first

run, we have seen that of the three hosts used, host2 was the one that was chosen the most by the learning automata. We can both see this in the probabilites chart, but we can also see it in the overview of the visits, where host 2 was visited 89 times out of 180.

This tells us that it has chosen to visit host 2 more than a round robin approach would have. A round robin would have visited 60 times.

When looking at the second run, we can see something similar, where host 3 is now the one being chosen the most. With now, it being visited 71 times of 180.

From the averages that we have gathered, we can also see that averages are higher in the learning automata than the round-robin all over. This is something which is interesting, especially when the numbers are so different.

From the data from experiment 3, it could be interesting to test the learning automata, with more hosts, and more iterations. This could be done we could see if it would bring any change.

Especially, after we have now tested different values of θ. The reason this was not done now is that it is time-consuming, as just running 180 iterations takes 4 to 5 hours.

During the experiments, some interesting finds were done. One of them was about Amazon's servers during experiment 2. Not only did the transfer take longer to Ohio than to London, which was expected, but the checkpoint and restoring of the containers also took longer, which was not ex-

pected.

As the different AWS virtual machines were running on the same tier instance, this is interesting. Running on comparable hardware should have given them similar performance.

During the experiments, and also when first creating the collector we have had some problems. During experiment three, we had the problem that the execution of the collector would crash after 1 to 2 hours. This was weird, as the script runs in a loop. What was found out was that the InfluxDB container was crashing more when doing larger runs. To mitigate this, the best solution was to check if the container was up running and, if not, restart it. These were, however, only quick fixes.

Early on, there were also some issues when starting to create and test containers for checkpointing and restoring. This was that Docker that we initially used would not correctly migrate containers using CRIU. Also, the only supported versions of it, where it worked at all was older ones. To fix this, we changed to use Podman instead, which has better support for migration.

Because of our project's nature, there are not many papers out there that can confirm or deny or claims, at least when it comes to our collection script. However, some papers touch upon similar topics and learning algorithms.

Some of them are ”Migrating LinuX Containers Using CRIU” (Pickartz et al., 2016), and ”Sledge: Towards Efficient Live Migration of Docker Containers” (Xu et al., 2020), which we have talked about earlier in the thesis.

Because of how our project was made and how little is required to execute

it, it should be possible to replicate our prototype and data. To replicate it, what is needed to replicate is a minimum of 2 to 3 virtual machines.

This can easily be acquired via Cloud Services such as Amazon Web Services and Google Cloud Engine. It is also possible to use equipment already available, such as running it in Virtualbox or a homelab.

When working on a project like this, something that always pops up is updating software that has been in use. For our project, we were lucky enough that instead of breaking something, it instead fixed something. This happened when we had a Podman update, where it fixed the issue we had with short exec commands not executing correctly.

When working with the project, balacing the available time is something which is not always easy. Especially when it comes what part of a project to work on. This became somewhat true, as the technical part took more time that first estimated.
This has been a topic, especially now when it comes to a time where we have less human contact, and almost only have contact over video due to Covid-19. For example, during this period, I have meet one of my supervisors once physically, and the rest over video.

This has made it a little different than usual. This has had both positive and negative impacts, the positive have been that you at times have more time available than normal, due to not many other things are happening, and the negatives has been that while there is some extra time, it is not always that easy to use it properly without getting tired out.

When it comes to our solution’s approach, we have chosen a not so typical way to do it. Since one of the main tasks is collecting information and met-

rics, there are many ways to do it. The most common type is probably to install an agent that gathers and sends to a central location when it comes to servers.

This could be done with the tools that we have used, with each server having a Telegraf instance each and a central location for the Influx database. However, we choose to go a different route, more akin to how a web crawler works, as we visit each server ourselves and bring the metrics back with us.

Our problem statement which is, ”P1: To what extent can machine learning be utilized to create a autonomous collector, that can be used for the migration of live containers.” is a problem statement that specifies to a certain degree what our plan is, and such if it would be changed drastically, that would probably change the project itself.

This is especially true when it comes to the two last parts of the problem statement, as that is an integral part. Removing or editing these parts, would drastically change what would be worked towards.

When working with projects such as these, you can always ask yourself afterwards, if you would do anything different if you did the project again? There are always things that you could try to do differently, but most of them I would do the same. Though, there are some things that I would do different, these things mainly comes down to research, especially when it comes to the tools that I have been using.

I would probably try to research these things earlier, and try to get a more in depth knowledge about them. This is especially true, if I had more time on my hands, as there is only so much that you are able to do through one semester.

Some of the interesting things that have emerged throughout the project, is the thought about what other things you could use it for, especially when it comes to different type of metrics, but also what kind of containers you can move instead of InfluxDB and Telegraf.

When it comes to these thoughts, luckily the project should be manageable to repeat. All of our tools and scripts are available online through GitHub, and all of the software that has been used are free to use. People should be able to follow us, chapter by chapter, especially through the results chapter.

Chapter 6

Conclusion

The initial idea of this project started with wanting to use the checkpoint/restoring features of CRIU to migrate stateful containers from one host to another, to be able to migrate a container to another host when needed, easily. This expanded to include using an autonomous collector to collect metrics, which then chooses where the container should be moved depending on the metrics collected.

There has been performed three different experiments. Moving a container from one host to another repeatedly, using a collector script to collect metrics from one host, and then go to another host and repeat in a round robin approach. In the last experiment, we have tested to see if a learning algorithm is the better approach for collecting metrics compared to a approach using only round-robin.

From the data from experiment 3, we can see that using a learning algorithm greatly improves what host is visited based on the available ram.

Through the averages, we can see that the average is always higher on the learning automata compared to the round robin approach, as such it seems likes it visits those with higher available ram and those that has

more change in ram usage.

While the learning algorithm seems to work, as it repeatedly visits those with the most change on the most, there seems to be a need to do more adjusting and more experiments, especially with more hosts and more iterations.
Our problem statement has been ”P1: To what extent can machine learning be utilized to create an autonomous collector, that can be used for the migration of live containers.” From this statement, we can go through what has been achieved during this thesis.

We have performed experiments that check to what extent machine learning can be utilized. We have created a collector which has used our learning algorithm to decide where to go next. We have created a collector which is capable of live migrating containers based on data that has been collected using InfluxDB and Telegraf.

To conclude, we have achieved what our problem statement states. As we have created a collector that uses a learning algorithm to be autonomous, we have performed experiments on said learning algorithm to see what approach is better to utilize and can also migrate live containers when needed.

Chapter 7

Future Works

Due to the time limit when working with this project, there are multiple things that we could not cover this time, and are things that can be done at a later date.

The first thing comes to the use of Telegraf plugins. In this project, we have only tested a couple and used one of the standard plugins. A thing that is possible to work on in the future is the testing and use of other plugins, as Telegraf has many plugins that can record metrics in different ways.

Since there are many plugins available, you could create a function that lets you change what metric you want to be recorded without changing any of the script directly. This is something that could be done, and could be for example used using command arguments.

One expansion of this is using a plugin that can record the metrics of the main running containers, instead of recording the host. This can be done through the use of the plugins ”Docker” and ”Docker Logs” which records metrics and logs.

One thing that we wanted to implement, but did not have enough time for,

is the ability to instead of only moving the collector between already existing hosts, is to make the script able to create new virtual machines as it seems fit.

For example if none of the hosts come back with satisfactory data, that it could then create a new virtual machine instead of just staying on the old host. This can be done through different API's, and would need to be setup up differently depending on where the virtual machines is hosted. This is something that should be possible on most cloud providers, and also OpenStack.

One thing that should be possible to do is expand the system by adding more options to what is already there. This can for example be moving more than one container, or have the system do other things than just moving containers.

One such things can be turning on/off containers based the metrics collected. Another is starting up duplicates, if that is needed. Also, because of the way the system is made, you can also have it run any script and commands you want. So if you want it to, you can have it run a set of tasks after the metrics have been collected.

In addition to these it should be looked upon if there is any better databases to store the metrics on, as InfluxDB tends to be somewhat unstable, and crash a lot. Especially on longer runs.
It should also possible to test the prototype on more operating systems than it currently is. Today it is only tested on Fedora 33

Lastly, one part of future work is performing more tests on the collector script with the learning automata, because today it has only tested with

180 iterations. If possible, increasing the number of iterations and number of hosts would give you more reliable data. This could easily done, the main part why it has not been done, is that it is time consuming, and has had a tendency to crash underway. If you could fix the crashing, running it over more iterations would easily be done.

Bibliography

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N. & Merle, P. (2017). Autonomic vertical elasticity of docker containers with elasticdocker. *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 472–479. https://doi.org/10.1109/CLOUD.2017.67

Autonomous systems and networks (asn). (n.d.). Retrieved April 12, 2021, from https://www.oslomet.no/en/research/research-groups/autonomous-systems-networks

AzuraCast. (n.d.). Azuracast is a free and open-source self hosted web radio management suite. Retrieved April 12, 2021, from https://www.azuracast.com/

Baeldung. (2020). Virtualization vs containerization. Retrieved April 12, 2021, from https://www.baeldung.com/cs/virtualization-vs-containerization

Checkpoint-Restore. (n.d.). Checkpoint-restore/criu. Retrieved April 12, 2021, from https://github.com/checkpoint-restore/criu

Chen, Y. (2015). Checkpoint and restore of micro-service in docker containers. *Proceedings of the 3rd International Conference on Mechatronics and Industrial Informatics*. https://doi.org/10.2991/icmii-15.2015.160

Criu. (n.d.). https://criu.org/Main_Page

Global infrastructure as a service (iaas) market 2021-2025: Market is poised to grow by $136.21 billion, at a cagr of 27% - researchand-markets.com. (2021). https://www.businesswire.com/news/home/

20210402005093/en/Global-Infrastructure-as-a-Service-IaaS-Market-2021-2025-Market-is-Poised-to-Grow-by-136.21-Billion-at-a-CAGR-of-27---ResearchAndMarkets.com

Knutsen, K. M. (2021). Kjetilknutsen/migration-via-autonomous-collector. https://github.com/KjetilKnutsen/Migration-via-autonomous-collector

Ma, L., Yi, S. & Li, Q. (2017). Efficient service handoff across edge servers via docker container migration. *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. https://doi.org/10.1145/3132211.3134460

Packages - criu. (n.d.). Retrieved April 12, 2021, from https://criu.org/Packages

Pickartz, S., Eiling, N., Lankes, S., Razik, L. & Monti, A. (2016). Migrating linux containers using criu. *Lecture Notes in Computer Science*, 674–684. https://doi.org/10.1007/978-3-319-46079-6_47

Wang, T., Xu, J., Zhang, W., Gu, Z. & Zhong, H. (2018). Self-adaptive cloud monitoring with online anomaly detection. *Future Gener. Comput. Syst.*, *80*(100), 89–101. https://doi.org/10.1016/j.future.2017.09.067

Xu, B., Wu, S., Xiao, J., Jin, H., Zhang, Y., Shi, G., Lin, T., Rao, J., Yi, L. & Jiang, J. (2020). Sledge: Towards efficient live migration of docker containers. *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, 321–328. https://doi.org/10.1109/CLOUD49709.2020.00052

Yazidi, A., Hassan, I., Hammer, H. & Oommen, B. (2020). Achieving fair load balancing by invoking a learning automata-based two-time-scale separation paradigm. *IEEE Transactions on Neural Networks and Learning Systems*, *PP*, 1–14. https://doi.org/10.1109/TNNLS.2020.3010888

Yu, C. & Huan, F. (2015). Live migration of docker containers through logging and replay. *Proceedings of the 3rd International Conference on Mechatronics and Industrial Informatics*. https://doi.org/10.2991/icmii-15.2015.106

Zavala, E., Franch, X. & Marco, J. (2019). Adaptive monitoring: A systematic mapping. *Information and Software Technology*, *105*, 161–189. https://doi.org/https://doi.org/10.1016/j.infsof.2018.08.013

Appendices

A.1 Code & Scripts

Listing 1: Collector script

```
1  #!/bin/bash
2  function collect {
3      time (
4      if [ "$line" = "$host_ip" ]
5      then
6          tput setaf 4
7          echo ${line} LOCALHOST
8          echo ================================================
9          tput setaf 7
10         sed -i "94s/.*/ hostname = \"${line}\"/" /home/fedora/
               MVAC/telegraf.conf
11         sudo podman rm -f influxdb
12         sudo podman volume prune -f
13         sudo podman container restore --tcp-established -i /home
               /fedora/MVAC/export.tar.gz
14         sleep 5
15         sudo podman start influxdb
16         sleep 5
17         influx=$(sudo podman ps | grep -o influxdb | sed -e '1d'
               )
18         if [ "influxdb" = "$influx" ]
19         then
20             sudo podman run -v /home/fedora/MVAC/telegraf.conf:/
                   etc/telegraf/telegraf.conf:Z -d --name telegraf
                   --net=container:influxdb docker.io/library/
                   telegraf:1.18
21             sleep 20
22             sudo podman rm -f telegraf
23             sudo podman container checkpoint influxdb --tcp-
                   established -e /home/fedora/MVAC/export.tar.gz
24         else
25             sudo podman start influxdb
```

```
26              sudo podman run -v /home/fedora/MVAC/telegraf.conf:/
                    etc/telegraf/telegraf.conf:Z -d --name telegraf
                    --net=container:influxdb docker.io/library/
                    telegraf:1.18
27              sleep 20
28              sudo podman rm -f telegraf
29              sudo podman container checkpoint influxdb --tcp-
                    established -e /home/fedora/MVAC/export.tar.gz
30          fi
31          sudo podman rm -f influxdb
32          sudo podman volume prune -f
33      else
34          tput setaf 4
35          echo "${line}"
36          echo ================================================
37          tput setaf 7
38          ssh -i ./id_rsa fedora@${line} "mkdir -p /home/fedora/
                MVAC"
39          scp -i ./id_rsa /home/fedora/MVAC/export.tar.gz fedora@$
                {line}:/home/fedora/MVAC/export.tar.gz
40          scp -i ./id_rsa /home/fedora/MVAC/telegraf.conf fedora@$
                {line}:/home/fedora/MVAC/telegraf.conf
41          ssh -i ./id_rsa fedora@${line} "sed -i '94s/.*/ hostname
                 = \"${line}\"/' /home/fedora/MVAC/telegraf.conf"
42          ssh -i ./id_rsa fedora@${line} "sudo podman container
                restore --tcp-established -i /home/fedora/MVAC/export
                .tar.gz"
43          ssh -i ./id_rsa fedora@${line} "sleep 5"
44          ssh -i ./id_rsa fedora@${line} "sudo podman start
                influxdb"
45          ssh -i ./id_rsa fedora@${line} "sleep 5"
46          influx=$(ssh -i ./id_rsa fedora@${line} "sudo podman ps
                | grep -o influxdb | sed -e '1d'")
47          if [ "influxdb" = "$influx" ]
48          then
49              set +e
```

```
50              ssh -i ./id_rsa fedora@${line} "sudo podman run -v /
                    home/fedora/MVAC/telegraf.conf:/etc/telegraf/
                    telegraf.conf:Z -d --name telegraf --net=
                    container:influxdb docker.io/library/telegraf
                    :1.18"
51              ssh -i ./id_rsa fedora@${line} "sleep 20"
52              ssh -i ./id_rsa fedora@${line} "sudo podman rm -f
                    telegraf"
53              ssh -i ./id_rsa fedora@${line} "sudo podman
                    container checkpoint influxdb --tcp-established -
                    e /home/fedora/MVAC/export.tar.gz"
54              if [ $? -eq 0 ]; then
55                  echo "Checkpoint succeeded"
56              else
57                  ssh -i ./id_rsa fedora@${line} "sudo podman
                        start influxdb"
58                  ssh -i ./id_rsa fedora@${line} "sleep 10"
59                  echo "Trying again"
60                  ssh -i ./id_rsa fedora@${line} "sudo podman run
                        -v /home/fedora/MVAC/telegraf.conf:/etc/
                        telegraf/telegraf.conf:Z -d --name telegraf
                        --net=container:influxdb docker.io/library/
                        telegraf:1.18"
61                  ssh -i ./id_rsa fedora@${line} "sleep 20"
62                  ssh -i ./id_rsa fedora@${line} "sudo podman rm -
                        f telegraf"
63                  ssh -i ./id_rsa fedora@${line} "sudo podman
                        container checkpoint influxdb --tcp-
                        established -e /home/fedora/MVAC/export.tar.
                        gz"
64              fi
65              set -e
66          else
67              ssh -i ./id_rsa fedora@${line} "sudo podman start
                    influxdb"
68              ssh -i ./id_rsa fedora@${line} "sleep 10"
```

```
69              ssh -i ./id_rsa fedora@${line} "sudo podman run -v /
                    home/fedora/MVAC/telegraf.conf:/etc/telegraf/
                    telegraf.conf:Z -d --name telegraf --net=
                    container:influxdb docker.io/library/telegraf
                    :1.18"
70              ssh -i ./id_rsa fedora@${line} "sleep 20"
71              ssh -i ./id_rsa fedora@${line} "sudo podman rm -f
                    telegraf"
72              ssh -i ./id_rsa fedora@${line} "sudo podman
                    container checkpoint influxdb --tcp-established -
                    e /home/fedora/MVAC/export.tar.gz"
73          fi
74          scp -i ./id_rsa fedora@${line}:/home/fedora/MVAC/export.
                tar.gz /home/fedora/MVAC/export.tar.gz
75          ssh -i ./id_rsa fedora@${line} "sudo podman rm -f
                influxdb"
76          ssh -i ./id_rsa fedora@${line} "sudo podman volume prune
                -f"
77      fi
78  )
79  }
80
81  function migrate {
82      set +e
83      timestamp=$(cat ./bin/timestamp)
84      bucket_id=$(sudo podman exec -it influxdb influx bucket list
            | grep bucket | awk '{print $1}')
85      if [ -z "$bucket_id" ]
86      then
87          echo "InfluxDB down, restarting"
88          sudo podman start influxdb
89          sleep 5
90          bucket_id=$(sudo podman exec -it influxdb influx bucket
                list | grep bucket | awk '{print $1}')
91          echo $bucket_id
92      fi
```

```
93      sudo podman exec -it influxdb influx v1 dbrp create --db
            telegraf --rp telegraf --bucket-id ${bucket_id} --default
94      sudo podman exec -it influxdb curl --get http://localhost
            :8086/query?db=telegraf --header "Authorization: Token
            mfndKMWpG8B3tVjjJYZm " --header "Accept: application/csv"
             --data-urlencode "q=SELECT time,host,max(available) FROM
             telegraf.telegraf.mem WHERE time > '${timestamp}'"
95
96      if [ $? -eq 0 ]; then
97          query=$(sudo podman exec -it influxdb curl --get http://
                localhost:8086/query?db=telegraf --header "
                Authorization: Token mfndKMWpG8B3tVjjJYZm " --header
                "Accept: application/csv" --data-urlencode "q=SELECT
                time,host,max(available) FROM telegraf.telegraf.mem
                WHERE time > '${timestamp}'" | awk -F',' '{print $4}'
                )
98      else
99          sudo podman start influxdb
100         sleep 20
101         query=$(sudo podman exec -it influxdb curl --get http://
                localhost:8086/query?db=telegraf --header "
                Authorization: Token mfndKMWpG8B3tVjjJYZm " --header
                "Accept: application/csv" --data-urlencode "q=SELECT
                time,host,max(available) FROM telegraf.telegraf.mem
                WHERE time > '${timestamp}'" | awk -F',' '{print $4}'
                )
102         if [ $? -ne 0 ]; then
103             echo "RESTARTING"
104             bucket_id=$(sudo podman exec -it influxdb influx
                    bucket list | grep bucket | awk '{print $1}')
105             sudo podman exec -it influxdb influx v1 dbrp create
                    --db telegraf --rp telegraf --bucket-id ${
                    bucket_id} --default
106             query=$(sudo podman exec -it influxdb curl --get
                    http://localhost:8086/query?db=telegraf --header
                    "Authorization: Token mfndKMWpG8B3tVjjJYZm " --
```

```
                    header "Accept: application/csv" --data-urlencode
                     "q=SELECT time,host,max(available) FROM telegraf
                    .telegraf.mem WHERE time > '${timestamp}'" | awk
                    -F',' '{print $4}')
107             fi
108         fi
109         ip=$(echo $query | awk '{print $2}')
110     }
111
112     function move {
113         rsync -ahl /home/fedora/MVAC fedora@${ip}:/home/fedora/MVAC
114         sudo podman container checkpoint main-app -e /home/fedora/
                MVAC/main-export.tar.gz
115         rsync -ah --progress /home/fedora/MVAC/main-export.tar.gz
                fedora@${ip}:/home/fedora/MVAC/main-export.tar.gz
116         ssh fedora@${ip} "sudo podman restore -i /home/fedora/MVAC/
                main-export.tar.gz"
117         ssh -i ./id_rsa fedora@${ip} "(sudo -u root crontab -l 2>/
                dev/null; echo "* * * * * ~/MVAC/trigger.shh") | sudo
                crontab -u root -"
118
119         rm -r /home/fedora/MVAC
120         (sudo crontab -u root -l | grep -v 'trigger.hh')  | sudo
                crontab -u root -
121     }
122
123     function query {
124         timestamp_2=$(cat ./bin/timestamp_2)
125         cp ./bin/results-current ./bin/results-previous
126         sudo podman exec -it influxdb curl --get http://localhost
                :8086/query?db=telegraf --header "Authorization: Token
                mfndKMWpG8B3tVjjJYZm " --header "Accept: application/csv"
                 --data-urlencode "q=SELECT time,host,max(available) FROM
                 telegraf.telegraf.mem WHERE time > '${timestamp_2}'
                GROUP by host" > ./bin/results-current
127         sed -i "1d" ./bin/results-current
```

```
128     if [ $c -eq 1 ]
129     then
130         cp ./bin/results-current ./bin/results-list
131     fi
132 }
133
134 function calc {
135     > ./bin/calc
136     while read line
137     do
138         ip1=$(echo $line | awk -F ',' '{print $4}')
139         pre=$(cat ./bin/results-list | grep $ip1)
140         first=$(echo $pre | awk -F ',' '{print int($5)}')
141         second=$(echo $line | awk -F ',' '{print int($5)}')
142
143         calc=$(expr $first - $second)
144         echo "$first - $second = $calc"
145         if [[ ${calc:0:1} == "-" ]]
146         then
147             newcalc=$(echo $calc | sed 's/^-\(.*\)/\1/')
148             echo $newcalc >> ./bin/calc
149         else
150             echo $calc >> ./bin/calc
151         fi
152
153         sed -i -e "s/$pre/$line/" ./bin/results-list
154
155         if [ $? -eq 1 ]
156         then
157             echo $line >> ./bin/results-list
158         fi
159
160     done < ./bin/results-current
161     var=$(sort -nrk1 ./bin/calc | head -1)
162     echo $var > ./bin/calc
163
```

```
164     calc_file=$(cat ./bin/calc)
165 }
166
167 function restore {
168     tput setaf 4
169     echo "Restoring influx at host"
170     tput setaf 7
171     influxdb=$(sudo podman ps -a | grep -o influxdb | sed -e '1d
            ')
172
173     if [ "influxdb" = "$influxdb" ]
174     then
175         sudo podman rm -f influxdb
176         sudo podman volume prune -f
177         sleep 5
178         sudo podman container restore --tcp-established -i /home
                /fedora/MVAC/export.tar.gz
179     else
180         sudo podman container restore --tcp-established -i /home
                /fedora/MVAC/export.tar.gz
181     fi
182 }
183 set -e
184
185 file=/home/fedora/MVAC/export.tar.gz
186 if [ ! -f $file ]
187 then
188     sudo podman run -d --name influxdb -p 8086:8086 \
189         -e DOCKER_INFLUXDB_INIT_MODE=setup \
190         -e DOCKER_INFLUXDB_INIT_USERNAME=telegraf \
191         -e DOCKER_INFLUXDB_INIT_PASSWORD=telegraf \
192         -e DOCKER_INFLUXDB_INIT_ORG=my-org \
193         -e DOCKER_INFLUXDB_INIT_BUCKET=bucket \
194         -e DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=mfndKMWpG8B3tVjjJYZm
                \
195         docker.io/library/influxdb:2.0.4
```

```
196             sudo podman container checkpoint influxdb --tcp-
                    established -e /home/fedora/MVAC/export.tar.gz
197             sudo chown fedora:fedora /home/fedora/MVAC/export.
                    tar.gz
198             echo 0 > ./bin/counter
199             if [ -f ./bin/calc* ]; then
200                 rm ./bin/calc*
201             fi
202             if [ -f ./bin/pickle.p ]; then
203                 rm ./bin/pickle.*
204             fi
205             > ./bin/results-list
206             > ./bin/ip
207 fi
208
209 n=$(cat ./bin/counter)
210 m=$(( n + 1 ))
211 echo $m > ./bin/counter
212
213 #host_ip=$(curl -s ifconfig.me)   # use this if using public ip
        addresses
214 host_ip=$(hostname -I | awk '{print $1}')   # use this if using
        local ip addresses
215
216 FILES="./hosts.txt"
217
218 c=$(cat ./bin/counter)
219 if [ $c -lt 3 ]
220 then
221     ssh -i ./id_rsa fedora@172.16.0.70 ~/MVAC/stress.sh &
222     ssh -i ./id_rsa fedora@172.16.0.71 ~/MVAC/stress.sh &
223     date --rfc-3339=seconds | sed 's/ /T/' | sed 's/.\{6\}$//' |
            sed 's/$/Z/g' > ./bin/timestamp
224     date --rfc-3339=seconds | sed 's/ /T/' | sed 's/.\{6\}$//' |
            sed 's/$/Z/g' > ./bin/timestamp_2
225     LINES=$(cat $FILES)
```

```
226     for line in $LINES
227     do
228         collect
229     done
230     restore
231     migrate
232     query
233
234     if [ $c -gt 1 ]
235     then
236         calc
237         py=$(python3 LA_with_barriers.py)
238         echo $py
239         nr=$(echo $py | awk 'END {print $NF}')
240         cat ./hosts.txt | sed -n "${nr}p" > ./bin/ip
241     fi
242     echo $ip
243 fi
244
245 if [ $c -gt 2 ]
246 then
247     r=$(cat ./hosts.txt | wc -l)
248     r=$(expr $r \* 2)
249     r=180
250     date --rfc-3339=seconds | sed 's/ /T/' | sed 's/.\{6\}$//' |
          sed 's/$/Z/g' > ./bin/timestamp
251     ssh -i ./id_rsa fedora@172.16.0.69 ~/MVAC/log.sh &
252     ssh -i ./id_rsa fedora@172.16.0.70 ~/MVAC/log.sh &
253     ssh -i ./id_rsa fedora@172.16.0.71 ~/MVAC/log.sh &
254     for i in $( seq 1 $r )
255     do
256         date --rfc-3339=seconds | sed 's/ /T/' | sed 's/.\{6\}$
              //' | sed 's/$/Z/g' > ./bin/timestamp_2
257         line=$(cat ./bin/ip)
258         rc=$(cat ./bin/results-current)
259         collect
```

```
260         time (
261         restore
262         migrate
263         query
264         if [ -z "$rc" ]
265         then
266             echo "Restarting restore, migrate and query"
267             restore
268             migrate
269             query
270         fi
271         calc
272         py=$(python3 LA_with_barriers.py)
273         echo $py
274         nr=$(echo $py | awk 'END {print $NF}')
275         cat hosts.txt | sed -n "${nr}p" > ./bin/ip
276         echo $ip > ./bin/ip-2
277     )
278 done
279 ip=$(cat ./bin/ip-2)
280 echo $ip
281 ssh -i ./id_rsa fedora@172.16.0.69 "pkill -f log.sh"
282 ssh -i ./id_rsa fedora@172.16.0.70 "pkill -f log.sh"
283 ssh -i ./id_rsa fedora@172.16.0.71 "pkill -f log.sh"
284 ssh -i ./id_rsa fedora@172.16.0.70 "pkill -f stress.sh"
285 ssh -i ./id_rsa fedora@172.16.0.71 "pkill -f stress.sh"
286 fi
```

Listing 2: Learning Algorithm

```
1 import random, pickle, os
2
3 def weighted_choice(weights):
4     totals = []
5     running_total = 0
6
7     for w in weights:
```

```
 8            running_total += w
 9            totals.append(running_total)
10
11        rnd = random.random() * running_total
12        for i, total in enumerate(totals):
13            if rnd < total:
14                return i
15
16
17
18  n=len(open("./hosts.txt").readlines(  )) #number of hosts to
        monitor
19  reward=[0]*n
20  lamda=0.15  #learning parameter, between 0 and 1.
21
22  pmax=0.98  #barrier, limit that any probablity will not exceed
23
24  pmin=(1-pmax)/(n-1)
25
26  #if one is pmax, while the rest, n-1 are at pmin, then the prob
        will sum to 1. pmax+(n-1)*pmin=pmax+(n-1)*(1-pmax)/(n-1)=1
27
28
29  if os.path.exists("./bin/pickle.p"):
30      prob = pickle.load(open("./bin/pickle.p","rb"))
31      p = prob[0]
32      max_change = pickle.load(open("./bin/pickle.m","rb"))
33
34  if not os.path.exists("./bin/pickle.p"):
35      p=[1.0/n]*n
36      max_change = 0
37
38  chosen_index=weighted_choice(p) #p is probablity, roulettte
        wheel
39  #chosen_index=the host that we need to vist.
40  file = open("./bin/calc", "rt")
```

```
41 contents = file.read()
42
43 change_chosen=int(contents)  #this needs to be computed based on
       amount of change, since last visit (absolute value)
44
45 if change_chosen>max_change:
46     max_change=change_chosen  #to keep track of max change for
           normalization
47
48
49 #updating the prob of the chosen wesite, we will increase the
       prpb of the chosen host in a proportional manner to the
       amount of change
50 p[chosen_index]=p[chosen_index]+lamda*(change_chosen/max_change)
       *(pmax-p[chosen_index])
51
52 #I will reduce the prob for all hosts that were not chosen, bcs
       I inreased the prob of the chosen.
53 for k in range(0,n):
54
55     if (k!=chosen_index):
56
57         p[k]=p[k]+lamda*(change_chosen/max_change)*(pmin-p[k]) #
               note (pmin-p[k]) is negative
58
59
60
61
62 #print "iteration", i
63 print(p)
64 print(chosen_index+1)
65
66 pickle.dump([p], open("./bin/pickle.p", "wb"))
67 pickle.dump(max_change, open("./bin/pickle.m", "wb"))
```

Listing 3: Trigger script

```
#!/bin/bash
mem=$(free --mega | grep Mem: | awk '{print $7}')
echo '* * * * * touch ~/MVAC/trigger.sh' | crontab -
if [ $mem -lt 1000 ]
then
 echo "Memory available is less than 1000mb, starting collector"
 echo '' | crontab -
 bash /home/fedora/MVAC/collector.sh
fi
```

Listing 4: Cleanup script

```
#!/bin/bash
sudo podman rm -f influxdb
sudo podman volume prune -f
rm ./export.tar.gz
```

Listing 5: Stress script

```
#!/bin/bash
while :
do
 stress-ng --vm-bytes $(awk '/MemAvailable/{printf "%d\n", $2 *
     0.4;}' < /proc/meminfo)k --vm-keep -m 1 -t 25
 sleep 15
done
```

Listing 6: Logging script

```
#!/bin/bash
while :
do
 date=$(date)
 mem=$(free | awk '{print $6}' | sed -n 2p)
 echo $date $mem >> logfile
 sleep 135
done
```

A.2 Additional Attached Files

Listing 7: Poster

Added as an additional attachment: See poster.png

Listing 8: Prototype

Added as an additional attachment: See prototype.zip