

2 May 2017

An Ansible implementation of a self-configuring Beowulf cluster of Raspberry Pis in a localised environment for the purpose of distributed computing using Open MPI

Theocharis Ledakis

Student ID: 5717206

Supervisor: Dr Carey Pridgeon

Course: BSc Computing (Honours) with Industrial Placement

School of Computing, Engineering and Mathematics, Coventry University

Project repository:

<https://gitlab.com/ledakis/picluster>

300COM / 303COM Declaration of originality

I Declare that This project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

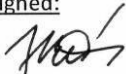
Statement of copyright

I acknowledge that the copyright of this project report, and any product developed as part of the project, belong to Coventry University. Support, including funding, is available to commercialise products and services developed by staff and students. Any revenue that is generated is split with the inventor/s of the product or service. For further information please see www.coventry.ac.uk/ipr or contact ipr@coventry.ac.uk.

Statement of ethical engagement

I declare that a proposal for this project has been submitted to the Coventry University ethics monitoring website (<https://ethics.coventry.ac.uk/>) and that the application number is listed below (Note: Projects without an ethical application number will be rejected for marking)

Signed:



Date:

30 April 2017

Please complete all fields.

First Name:	Theocharis
Last Name:	Ledakis
Student ID number	5717206
Ethics Application Number	P47560
1 st Supervisor Name	Dr. Carey Pridgeon
2 nd Supervisor Name	Prof. Kuo-Ming Chao

This form must be completed, scanned and included with your project submission to Turnitin. Failure to append these declarations may result in your project being rejected for marking.



Certificate of Ethical Approval

Applicant:

Theocharis Ledakis

Project Title:

An Ansible implementation of a self-configuring Beowulf cluster of Raspberry Pis in a localised environment for the purpose of distributed computing using Open MPI.

This is to certify that the above named applicant has completed the Coventry University Ethical Approval process and their project has been confirmed and approved as Low Risk

Date of approval:

16 February 2017

Project Reference Number:

P47560

Table of Contents

a. Abstract	6
b. Acknowledgements.....	6
1. Introduction	7
2. Term definitions.....	8
3. Literature review.....	9
3.1 Introduction	9
3.2 Old cluster systems	9
3.3 Recent approaches to cluster systems with cheap components.....	9
3.3.1 Broberg cluster.....	9
3.3.2 Iridis-pi	10
3.3.3 RPiCluster	10
3.3.4 DeConinck cluster	11
3.3.5 Pi Dramble.....	12
3.4 Conclusion.....	12
4. Method	13
4.1 Project deliverables.....	13
4.1.1 Network independent.....	13
4.1.2 Unattended setup	13
4.1.3 Input will be limited	13
4.1.4 Easy access to the master	13
4.1.5 Documentation	13
4.2 Methodology.....	13
4.3 Why Git?.....	14
4.4 Script injection	14
4.4.1 Choice of initiation for my script.....	14
4.4.2 Implementation	15
4.5 run.sh	16
4.6 IP update scripts.....	20
4.7 Master node scripts	21
4.7.1 Cron files	21
4.7.2 Inventory generator.....	22
4.8 Ansible.....	23
4.9 Python user interface.....	26
4.10 SSH to master and Documentation.....	28
5. Evaluation - Results.....	30

5.1 Deliverables.....	30
5.2 Demonstration	31
6. Project management.....	32
7. Discussion.....	33
8. Reflection on ethics.....	34
8.1 Social aspect.....	34
8.2 Legal aspect/Licencing	34
8.3. Ethics	34
9. Conclusion.....	35
10. Bibliography	36
11. Appendix	38
11.1 Project Proposal.....	38
11.2 Meeting diary.....	42
11.3 Presentation material	56
11.4 Handwritten notes/Diary.....	59

a. Abstract

Although there are projects that combine Ansible with Raspberry Pis, or OpenMPI with Raspberry Pis, there is none to combine the benefits of all those software technologies to automate the deployment of any number of Raspberry Pis with minimum intervention from the user in order to run OpenMPI projects.

This project's aim is to accomplish that and provide a complete package that can be used with little intervention from the end user. The completed project is making use of git and the Bash scripting language to initially establish a method of announcing the nodes' IP addresses to a central location, Ansible to configure at a higher level the swarm from a master node, and Python for the front-end tool that lets the user configure the few parameters needed for the swarm to operate and create the SD card images. Finally, OpenMPI is installed along with an NFS server and NFS clients for the nodes to communicate files. This implementation does not try to initialise the cluster using zeroconf techniques, due to restrictions on Coventry University's network.

The final code has been tested in a small environment of 3-6 Raspberry Pis and is working as expected. The project is also released as an open source repository on the following address:

<https://gitlab.com/ledakis/picluster>

The repository can be cloned and used directly to run the project from the user's account.

Keywords:

- MPI
- Raspberry Pi
- Ansible
- Automation
- Beowulf cluster
- Distributed computing

b. Acknowledgements

I would like to thank my supervisor Dr Carey Pridgeon for the continuous help and insights to the questions and problems I have had during this project, as well as sponsoring my trip to a developer conference and for providing the hardware, Raspberry Pis and power supplies. The completion of the problem could not have happened without the support of my good friend Manolis Kiagias in some highly technical matters.

Finally, I am grateful to the open source community that has produced such high-quality software, namely Linux, Debian, OpenMPI, Ansible, Raspbian, SSH, Python, Bash and others. I hope the resulting code of this project is going to be used by the community and cherished.

1. Introduction

The research question for this paper as explained in the Detailed proposal (Appendix 11.1) is:

How could the process of deploying a Raspberry Pi cluster in a local network be automated in order to achieve minimal configuration from the user?

The purpose of this project is to offer a set of tools that can easily and with little configuration and attention from the user, configure a cluster of Raspberry Pis using open source software and aiming to be modular, by using the Ansible automation framework.

Specifically, I will discuss ways and methodologies to manipulate the initialisation scripts of the Raspbian OS so that the nodes can start organising into a cluster, scripting of the automation of git updates back to GitLab and, as mentioned, creation of the necessary Ansible playbooks that continue from where the bash scripts finish and install and configure any software the user requires. For this project OpenMPI and NFS will be demonstrated, but because of the modular nature of Ansible many more tools can be added on the fly.

A potential reader of this report needs to have basic shell/bash scripting experience along with some knowledge of how Linux systems work, and an understanding of clusters and basic networking. OpenMPI is only used as the final example of the working cluster, so experience with this is valuable only to run the final product and program with a distributed set of runners in mind.

2. Term definitions

A list of definitions that are going to be used throughout this report.

- Raspberry Pi
A single board computer that was developed by the Raspberry Pi Foundation to be used in education in 2012. Since then, because of its low price (about £24) it has been used in many hobby projects in the community and has been a complete success. Two more versions with upgraded specs have been released to succeed the original.
- MPI and OpenMPI
Message Passing Interface is a system designed to enable programmers to create programs that can run on a distributed environment, like a cluster.
- Beowulf
Beowulf cluster is a cluster consisting of identical nodes, usually old commodity hardware.
- NFS
Network File Storage
- OS
Operating System
- Ansible Galaxy
A portal with collections of Ansible playbooks, made by the community.
- Playbook
The Ansible playbook is a set of scripts made in YaML format that are run by Ansible to achieve a variation of tasks, from administrative to scripting.
- Bash
The default shell for Unix systems. An extension to the original Bourne Shell, now called "Bourne Again SHell".
- Asciicast
A screencast, not with video but with the shell. This tool records the terminal session and can replay the STDIN/STDOUT to viewers, which makes an excellent tool to demonstrate shell software.
- dd
A Unix command that is a short, and a play of cc (Carbon Copy). It copies the origin to the destination as an exact copy because it copies in a lower level. Because of that, to use it for accessing system devices such as disks, super user privileges are required.

3. Literature review

3.1 Introduction

In this section I will discuss what other findings have led to forming my research question. The research on literature will begin by older cluster examples used in the industry and academia and then focus on smaller teams with less budget that achieve the same effect by using the Beowulf approach. Finally, I will explain how the current literature offers solutions that offer either automation but no integration with MPI or just MPI clusters that have to be setup manually.

3.2 Old cluster systems

Computer clusters is not a recent computing development, but instead it has been developed and utilised for many decades. There has been a plethora of big cluster systems used mostly by the military, but most importantly by universities and large tech companies. Good examples of such systems are the computing system of CERN (Bahyl et al. 2003, CERN 2012), the National Science Foundation's "TeraGrid" (NSF 2005), large supercomputers like the IBM Watson or smaller Beowulf deployments as the "Chiba City" or "Jazz", 256-node and 350-node respectively clusters at the Argonne National Laboratory (Gropp et al. 2003). Such systems are used mainly in research because they offer a cost effective solution to problems that require high performance and fault tolerance according to Sterling (2002).

Academics (Becker et al. 1995), small companies (Gardner 2014) and individuals (Stephenson 2005) have as well created clusters with a handful of nodes that were based on cheap hardware and used open source software to run. In 1995 and 2004 the software that Becker (1995) and Gardner (2004) used was very much like the ones that are in use today, an open source operating system (FreeBSD), OpenMPI and NFS, making them good choices to demonstrate the result of the working cluster in this project.

3.3 Recent approaches to cluster systems with cheap components

Although the software used for communication between the nodes has not changed (MPI/NFS) the approach to deploy the nodes and the hardware has.

Since the Raspberry Pi foundation released the first generation of the computer it has been used not only in primary schools (the target audience, initially), but used heavily by hobbyists and academics to showcase various projects. This was an effect of the low cost of the device and the very good Linux support, initially from Debian, one of the most popular Linux distributions, and then by Archlinux, Ubuntu and others.

3.3.1 Broberg cluster

The first attempt to run OpenMPI on the Raspberry Pi platform was by Broberg (2012) who, at that time, had to compile the binaries for the software to work as it was not included in the distribution's repositories. That of course is a very technical approach and is so time consuming that he had to run the compilation inside a virtual machine emulating the arm architecture on his personal computer.

Since then, the binaries for OpenMPI have been released in the official Raspbian repositories and can be used like any other package available there.



Figure 1: Iridis-pi (Cox et al. 2014)

3.3.2 Iridis-pi

A year later Professor Cox and his team at the university of Southampton have published a paper about Iridis-pi (Cox et al. 2014). Iridis-pi (Figure 1) is a 64-node cluster made of Raspberry Pis that has been configured manually to run LINPACK, OpenMPI and other benchmarking software. In this paper, they propose that the cluster of Raspberry Pis is ideal for academic purposes and argue that:

“The small size, low power usage, low cost and portability of the “Iridis-Pi” cluster must be contrasted against its relatively low compute power and limited communications bandwidth (compared to a contemporary, traditional HPC cluster), making this architecture most appropriate as a teaching cluster.” (Cox et al., 2014, p7)

One of the goals of starting this project has been the fact that Coventry University offers a parallel and distributed computing module, 370CT, that is based on OpenMPI. The module would definitely benefit from a way to help the lecturers deploy a cluster of cheap machines easily, to demonstrate the principles of distributed computing in workshops.

3.3.3 RPiCluster

Another cluster demonstration was the one named RPiCluster from Kiepert (2013), who has made a 32-node cluster and demonstrated its distributed processing capabilities in his report. Kiepert is using the Archlinux distribution which, as he suggests, is more efficient. Unfortunately, he is compiling the source code for his MPI software (MPICH), which, for the purpose of this project, was not acceptable as it is requiring substantial technical knowledge and time from the user. Furthermore, his approach configures manually the images needed for the nodes, adding complexity

to the end result. I aim to remove this complexity from the user and automate as many components as possible to deliver a working cluster with OpenMPI installed and ready to accept commands.

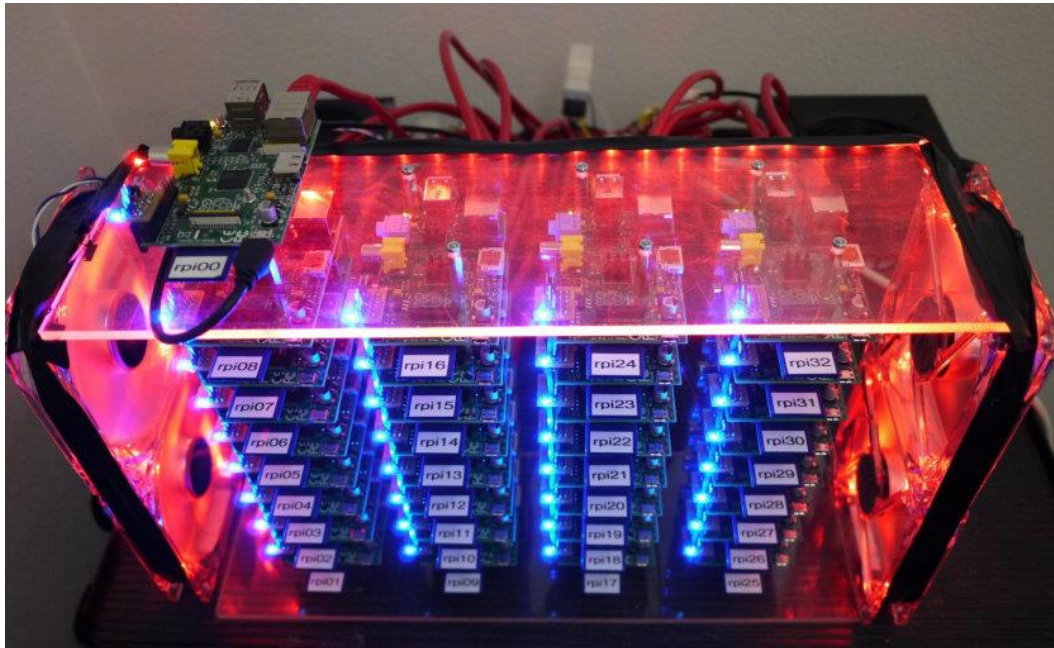


Figure 2: RPiCluster (Kiepert 2013)

Kiepert's paper shows that a cluster of Raspberry Pis can be organised in a compact box and powered from the IO headers instead of what one would normally do, which is power the device from the micro-USB port. He has demonstrated that he can pack 32 compute nodes in an enclosure similar to a desktop case as can be seen in Figure 2. In this project, I am going to be deploying Raspberry Pis in the usual way, connecting only the USB port to the mains and the Ethernet port to access the university network. This is a choice taken to make the process of creating a cluster as simple as possible for the user.

3.3.4 DeConinck cluster

Adam DeConinck has created an implementation of an Ansible-configurable Raspberry Pi cluster on his GitHub repository (DeConinck 2013, Nvidia and DeConinck 2013). His approach is very similar to what I aim to have installed at the end of the configuration of the cluster.

He uses Raspbian to operate the nodes, OpenMPI and NFS for the computation and Ganglia to monitor the system. His guide shows very good understanding of how to set up cluster systems with Ansible and that has been a source of inspiration for my own implementation.

One interesting step in his guide is the fact that he is using a different Ansible playbook for the master node (called "headnode") and another for the slave nodes.

Unfortunately, DeConinck's approach, like the previous ones mentioned, does not have user automation in mind for the initial configuration part. His guide, explains that the user has to clone SD cards, which is perfectly acceptable for the purposes of "as less user interaction as possible" that I want to achieve, but he is creating simple Raspbian images and does not include any specific

element in the system that would enable him to learn the booted Raspberry Pi's IP address. He just enables SSH and finds the IP using network mapping tools. Instead, he could add enough scripts/text files in the SD cards to let a central location know their network IP address, maybe even download and run scripts from that central location that the user has control of.

DeConinck's example as I have mentioned seems to be the closest to what I have in mind for a cluster that requires minimum intervention from the user (only the SD card imaging process is needed) and manages to configure itself to have an MPI, NFS software installed via Ansible playbooks.

3.3.5 Pi Dramble

Jeff Geerling has published his own implementation of a Beowulf cluster of Raspberry Pis that use Ansible to set up the high-level software. He is using Raspbian and Ansible to set up the nodes to serve a Drupal website through a High Availability cluster (Geerling 2015).

His project offers very good documentation on how to set up the Raspbian OS with Ansible and he has packaged a library of Ansible playbooks to the Ansible Galaxy repository.

On the other hand, Geerling's project does not automatically configure itself and does not install the software I require for this project (OpenMPI and NFS).

3.4 Conclusion

There are many good approaches to creating clusters with Raspberry Pi using many different tools and with many different software installed on the systems.

The old cluster examples I have talked about all use heavily customised software to configure and automate processes on the nodes manually. Broberg, Iridis-pi, RPiCluster and the DeConinck clusters all successfully install and configure OpenMPI for the purpose of distributed computing, but fail to achieve minimum interaction from the user, by requiring manual network configuration on each node. Finally Pi Dramble both does not automate the network setup and installs a LEMP stack instead of the MPI that I require.

These projects have shown that the Raspberry Pis can be used in a cluster to enable educators and students to discuss effectively how distributed computing works, test algorithms and programs on such a cluster with a very low budget.

Although the academics that work with distributed systems usually have good knowledge on how to deploy those systems, this is not always the case and it can be a very difficult experience if they must find their way out of a problematic setup. Based on the previous findings I will be making a set of tools that will set up a Beowulf cluster of Raspberry Pis using Ansible to set up the OpenMPI software automatically and with as little as possible intervention from the user.

4. Method

4.1 Project deliverables

After discussing with my supervisor on the findings that are discussed in the Literature review conclusion (Chapter 3.4), I have compiled a list of what this project aims to deliver and, by doing that, solve the problem discussed earlier.

4.1.1 Network independent

The scripts and the automation must be network agnostic. This means that the user will not have to ask a network administrator for any particular information and/or special access, such as, for example, static IP addresses or the ability to broadcast. The cluster will use the standard network stack of the Raspbian OS, which will act like any other machine connected to the network and get a standard IP address. That IP will be then communicated to a central location (git repository) and will thus allow each node to know about the rest.

4.1.2 Unattended setup

After imaging the SD cards the user should not have to do any other work for the system to get ready, apart from plugging in the devices. As long as the nodes have network (and internet) access to the git repository then the system will start setting itself up.

4.1.3 Input will be limited

Although I aim for as little user effort and interaction as possible, unavoidably they will have to perform a few trivial operations before the automation begins. The user will need to pass the git repository address, the private and the public key to access that repository to the script that creates the SD cards for the nodes. As an extra feature, I would like to add a question that lets the user choose if the generated SD card is for a master node or not.

4.1.4 Easy access to the master

The final feature that is required by this project is that the user will be able to connect easily to the master node and run OpenMPI commands/programs. As an extra feature, the user will be able to run extra Ansible playbooks if they wish to further extend the functionality of their cluster.

4.1.5 Documentation

Although not a feature, this is one of the deliverables for this project especially because it is going to be open source and potentially used by the community and further enhanced and tweaked.

4.2 Methodology

As I have explained earlier, the project will be mostly a compilation of programs and scripts that manage to set up the cluster using automation tools entirely. Thus, the primary research method will be building/programming and investigating the results to see if the deliverables are met. Along with the programming a lot of additional research for the specific technologies will be conducted to find the solutions that I find best for this project.

The investigatory part of the build methodology has been conducted in the third chapter and conclusions have been made about what is missing from the current literature that I am going to try and fill. In the following sections I will discuss how I have developed each of the several components of this cluster project and at the end how all work together.

4.3 Why Git?

As I have explained earlier, one of my goals is for the cluster to be able to share the nodes' IPs to a centralised location/service which then would be used by Ansible. This service needs to be running at least initially for the Raspberry Pis to auto-configure.

Having a custom full stack application running just for this purpose is a waste of programming time and since much simpler solutions exist it adds unnecessary complexity for the user. The suggestion from my supervisor in one of the early meetings was to use git to handle the orchestrating part, which was an interesting idea for me to explore.

SaaS providers like GitHub and GitLab offer exactly what we need, a 24/7 free service that can handle multiple connections efficiently (that is mostly because git itself is efficient in transmission of data (Walters 2012)) and work over SSH/HTTPS to overcome potential network restrictions.

An extra bonus is that they are very popular and offer a useful interface to work on. A lot of developers and many academics and students who are programming know of, or have used, at least one of the two SaaS providers I am considering.

Between the two I chose GitLab because they offer free private repositories which would prove useful to users that do not want to disclose their cluster's IP addresses to the public.

The repository I have created for this project can be found on this address and easily be cloned:

<https://gitlab.com/ledakis/picluster>

The simplicity of the repository cloning using the web interface adds to the overall user experience and removes a fair amount of friction at the very beginning of the process. For the scripts to work, the user will clone and download his copy of the repository and initiate from there.

4.4 Script injection

This part has been the most interesting and was a steep learning curve for me because I had to learn Bash scripting and I had to do it as the first thing because the rest of the steps depend on this. When a Linux system boots, it will initiate several core subsystems and then it will start other higher level functions.

4.4.1 Choice of initiation for my script

In my research, I needed to find a way to make sure I can inject a piece of code somewhere along that process so that my script can then run and start the automation. I have found several ways to do this:

a. Create my own service for the new system and have it start after the boot process.

This seemed like a good approach in the beginning, but eventually I concluded that it would take a lot of time to learn how to properly create a system service on Debian OS. Furthermore, I still did not know of a method to inject the script that installs the system service to the system. A chicken/egg problem which was discarded from the beginning.


```
1 [Unit]
2 Description=Service module for the picluster
3 Documentation=https://gitlab.com/tledakis/picluster
4
5 [Service]
6 Type=simple
7 ExecStart=/home/pi/picluster/service.sh
8 ExecStop=pkill service.sh
9 StandardOutput=null
10 Restart=on-failure
```

Figure 3: An attempt to create a service definition file

The only way to make this work would be to create my own custom images of Raspbian that have the service installed already; this is not one of this project's intentions because the Raspbian developer team release new versions based on Debian which means it is a release cycle that updates every couple of months and would in effect stop updates to the users of this cluster project until I update my own images.

b. Try the systemd method

systemd is the new initiation system that (almost) every major Linux distribution uses and has replaced init (the old system). RedHat (2015) and DigitalOcean (2015) provide extensive documentation on how to add "systemd unit files" which can be programmed to be services among other things, but again, given the limited time allocated to this task I chose to look for an easier, simpler solution.

c. Try init

init is the old good and tried system that has worked for decades in the Unix/Linux world and still works in many applications. It is still being preferred among traditional Unix communities like the FreeBSD developers. I was inclined to use this from the beginning just because it was so simple, for a script to be injected in the initiation system I only need to add it to the `/etc/rc.local` script that runs last after every other init system (Raspberry Pi Foundation 2014a).

d. Try cron

Cron is the internal scheduler for the Unix/Linux systems and can be programmed to do anything we want it do. It can be told to run scripts at set times or even run after a successful reboot using the `@reboot` option (Raspberry Pi Foundation 2014b). After trying to find a way to inject the script into cron I found that it is extremely hard and not suggested because I would be touching files that are meant to be used by cron only.

The choice was to go with the init approach as it was the simplest and thus the less error prone. The rest of them would involve additional learning that I could not afford.

4.4.2 Implementation

Now that I had an approach I wrote the script that needs to be injected into the `rc.local` file. One caveat of this is that any script that resides in the `rc.local` file needs to finish running successfully or the boot process will halt there.

```
21 if ! crontab -upi -l | grep "run.sh" >/dev/null; then
22     crontab -upi /boot/picluster/picron
23 fi
24
25 exit 0
```

Figure 4: The end of the rc.local script

```
1 */2 * * * * /boot/picluster/run.sh
```

Figure 5: The picron file

As shown on figure 4 the rc.local script in order to exit properly and in time, contains a simple command to add the picron file into the pi user's crontab. This was by design so we would get both the successful script exit and we would be utilising the very powerful cron service that I wanted to use as I have mentioned earlier at choice "d". Also, after careful consideration, I added the conditional statement to check if picron directive is already in the crontab and skip inserting it in that case. This would prevent inserting the directive every time the system reboots.

On figure 5 the single line content is shown that will be injected to the crontab. The */2 * * * * means it will run this command on every minute that is divisible by 2, thus every second minute.

4.5 run.sh

This is the most important file that orchestrates everything in the system. It is designed to make simple checks (because it runs very often) and then quit and save its parameters until next time it runs.


```

1  #!/usr/bin/env bash
2
3  # init vars
4  PI_check_master=1
5  PI_init_repo=0
6
7  PI_conffile=/home/pi/.piclusterrc
8
9  # init keys, conffile
10 if [ ! -f "$PI_conffile" ]; then
11     cp -f /boot/picluster/conffile.sh $PI_conffile
12     if [[ ! -d "/home/pi/.ssh" ]]; then
13         mkdir /home/pi/.ssh
14
15         fi
16         cp -f /boot/picluster/priv.key /home/pi/.ssh/id_rsa
17         chmod 600 /home/pi/.ssh/id_rsa
18         cp -f /boot/picluster/ssh_config /home/pi/.ssh/config
19         chmod 644 /home/pi/.ssh/config
20         cp -f /boot/picluster/pub.key /home/pi/.ssh/id_rsa.pub
21         chmod 644 /home/pi/.ssh/id_rsa.pub
22         cp -f /boot/picluster/pub.key /home/pi/.ssh/authorized_keys
23         chmod 600 /home/pi/.ssh/authorized_keys
24         cp -f /boot/picluster/gitconfig /home/pi/.gitconfig
25         chmod 644 /home/pi/.gitconfig
26         PI_init_repo=1
27     fi
28     # conffile exists by now
29     source $PI_conffile

```

Figure 6: run.sh first part

As can be seen on figure 6 it begins by initiating crucial variables for the script's logic, the "PI_check_master" and the "Pi_init_repo". The check master is set to True in the beginning because I want the script to be checking if it is the master, until it has other evidence that it is not. This way I can ensure that the master check will run at least the first time the fresh system boots and then when it determines its master status it will turn the checking off. The "PI_conffile" file contains the saved environment variable from previous runs, and as shown its location is ~/.piclusterrc.

The init repo variable is meant to let the script later to clone the repository to the system or not.



```

1  export PI_repo_addr=git@gitlab.com:ledakis/picluster.git

```

Figure 7: conffile.sh

The big if statement checks if the .piclusterrc exists and if it doesn't it means this is the first time the script runs. On figure 7 is shown the conffile that is generated by the SD imaging tool and contains

the repository address. This file becomes `.piclusterrc` and later on (line 29, figure 6) is loaded. In the condition block the rest to be done apart from the repository information, is of course the SSH keys. The keys are copied from the boot partition where were saved by the SD imaging tool that I am going to discuss later. Finally, the `init_repo` variable is set to true as we are certain that this is a first run of this system.

```
31  if [ ! $(which git) ]; then
32      sudo apt update
33      sudo apt -y install git ansible vim
34  fi
35
36  if [ ! -d /home/pi/picluster ]; then
37      PI_init_repo=1
38  fi
39
40  # init repo
41  if [ "$PI_init_repo" -eq "1" ]; then
42      if [[ -d /home/pi/picluster ]]; then
43          rm -rf /home/pi/picluster
44      fi
45      git clone -q $PI_repo_addr /home/pi/picluster
46  fi
47
```

Figure 8: `run.sh` second part

This is another interesting part of the run script (figure 8). It checks if the `git` command exists and runs the package manager (`apt`) to update and install Ansible and `git` which are essential, and a text editor.

What follows is another check, if the repository directory does not exist, it will set the `init_repo` variable again to True. I have added this after testing the system and when for some reason the `.piclusterrc` file was not written (IO problem for example) it would fail later because the repository was not there to run files from. This has happened a few times and after investigation it appears the SD cards sometimes fail to read files with the multiple partitions the Raspberry Pi uses.

The next condition checks if the `init_repo` variable is true and then checks if the repo is there. If it is, it will remove it by force and clone it again. This has been added to prevent failures in IO as mentioned before.

```

48 # init master, master runs ansible scripts
49 if [ "$PI_check_master" -eq "1" ]; then
50     bash /home/pi/picluster/node-master.sh
51     PI_check_master=0
52 fi
53
54 # ip update
55 bash /home/pi/picluster/up2git.sh
56
57 # by now this should end, saving vars
58 export -p > $PI_conffile
    chmod -x $PI_conffile

```

Figure 9: run.sh third part

Finally, the script will initiate the master checking script and set check variable to False so that it won't have to run that script again.

```

8 boot_master_file="/boot/picluster/master"
9 mac_addr=$(cat /sys/class/net/eth0/address | tr -d ":")
10
11
12 if [ -f $boot_master_file ]; then
13     echo $mac_addr > /home/pi/picluster/master
14     git -C /home/pi/picluster/ add /home/pi/picluster/master
15     git -C /home/pi/picluster/ commit -m "new master"
16     git -C /home/pi/picluster/ push origin master
17     if ! crontab -upi -l | grep "masterRun.sh" >/dev/null; then
18         (crontab -upi -l ; cat /boot/picluster/mastercron) | crontab -upi -
19     fi
20 fi
21

```

Figure 10: the node-master checking script

Figure 10 shows how the master check is performed. The SD card creation tool creates an empty file called master into the boot partition (more about the boot partition files follows). If the file exists, it means this node is the master and will add the mac address inside a file called "master" in the repository and upload it so that it is known to others (and the user). Then it will add the special master crontab using the same technique that I have used for the normal node, as explained earlier. In a similar manner, the master cron script runs every three minutes instead of two for the simple node. Worth mentioning is line 19 (figure 10) where the existing crontab is concatenated with the new directive into one. This ensures the run.sh script will continue to run and won't be overwritten by this operation.

At the end of "run.sh" (figure 9) the environment variables that have been created in this script are saved so they can be used the next time it runs. And just before that happens the script that updates the IP address to the repository is run, which I will discuss next.

4.6 IP update scripts

From “run.sh” the script that initiates the IP updating mechanism is “up2git.sh”.

```
up2git.sh 622 Bytes
1  #!/usr/bin/env bash
2
3  git -C /home/pi/picluster/ remote update
4  repo_update_needed=$(/home/pi/picluster/git_up2date_needed.sh)
5
6  if [ $repo_update_needed ]; then
7      git -C /home/pi/picluster/ fetch origin master
8      git -C /home/pi/picluster/ reset --hard FETCH_HEAD
9      git -C /home/pi/picluster/ clean -df
10 fi
11
12 ip_changed=$(/home/pi/picluster/ip2file.sh)
13
14 if [ $ip_changed ]; then
15     git -C /home/pi/picluster/ add /home/pi/picluster/ip$(cat /sys/class/net/eth0/address | tr -d ":")
16     git -C /home/pi/picluster/ commit -m "new ip for $(cat /sys/class/net/eth0/address | tr -d ':')"
17     git -C /home/pi/picluster/ push origin master
18 fi
19
20
```

Figure 11: up2git.sh

The “up2git.sh” script as shown in the figure 11 updates only the remotes of the repository (to preserve bandwidth and reduce system load) and then updates the repository if it has changed on the git server.

Then it checks if the system’s IP address is different than the one in the repository and if it is, it updates the repository with the new IP. The method I use to store the IPs was an attempt (after discussion with my supervisor) to create unique files for each IP, per system. The only thing that is unique and easily accessible on each machine is the Ethernet port’s mac address. So, I have used that as the file name of the file that contains the IP of the system.

```
git_up2date_needed.sh 239 Bytes
1  #!/usr/bin/env bash
2
3  update_needed=0
4
5  local_repo=$(git -C /home/pi/picluster/ rev-parse @)
6  remote_repo=$(git -C /home/pi/picluster/ rev-parse origin/master)
7
8  if [ $local_repo != $remote_repo ]; then
9      update_needed=1
10 fi
11 echo $update_needed
```

Figure 12: git_up2date_needed.sh

Figure 12 shows the mechanism that checks for updates on the remote repository using “git rev-parse” which returns the SHA1 of the HEAD’s commit (Chacon 2014). Because the SHA1 is a string it is easy to check if the two strings match and if not, force the update.

```

6 curr_ip="$(ip address show | awk -F '[ /]+' '/inet / && $3 != "127.0.0.1" {print $3}')"
7 ip_file="/home/pi/picluster/ip/$(cat /sys/class/net/eth0/address | tr -d ":")"
8 changed=0
9 if [ ! -f $ip_file ]; then
10     echo $curr_ip > $ip_file
11     changed=1
12 elif [ "$curr_ip" != "$(cat $ip_file)" ]; then
13     echo $curr_ip > $ip_file
14     changed=1
15 fi
16
17 echo $changed

```

Figure 13: ip2file.sh

In figure 13 above is shown the script that determines if the IP address has changed. It uses the “ip” command which was found after trying to use a better approach than the “ifconfig” command that created implications when run. The “awk” language will take the piped content and remove the lines that don’t contain “inet” for the internet interface and will also remove the loopback address (127.0.0.1).

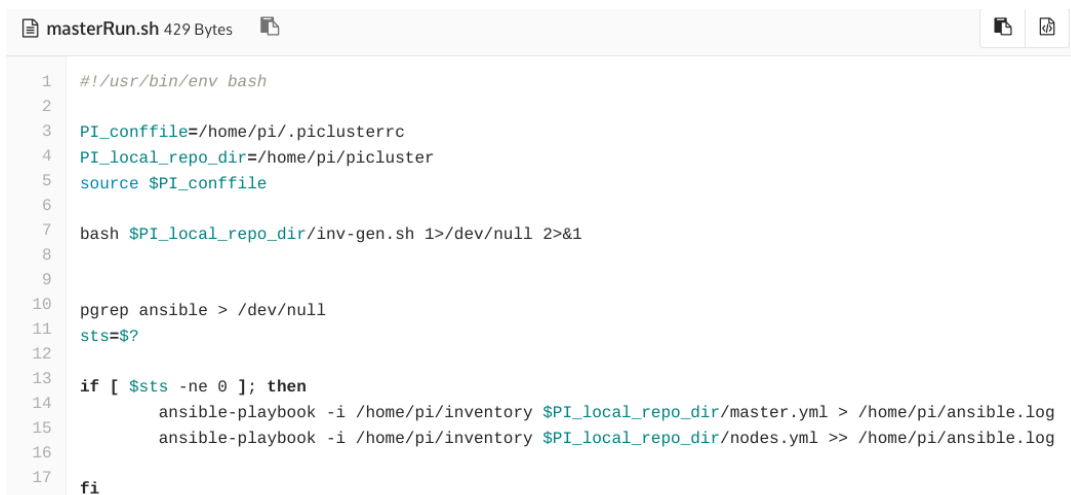
This implementation will work even if something goes wrong and, for example, the update is unsuccessful. The next time the cron runs “run.sh”, the repository will be reverted to the version that is on the remote and then “ip2file.sh” will again identify that the IP needs to change and will commit and push the change.

4.7 Master node scripts

So far the “simple” node scripts have been explained, and by this point one can understand how each node will be able to communicate its address and get the rest of the addresses. Also, because of the way the master scripts are injected, the master node is no different than the rest apart from a single empty file (located in /boot/picluster/master) initially.

4.7.1 Cron files

As shown on figure 10, the way the master script runs is identical to the regular scripts, by utilising the cron service. The content of the “masterRun.sh” script is shown in figure 14.



```

1 #!/usr/bin/env bash
2
3 PI_confdir=/home/pi/.picluster
4 PI_local_repo_dir=/home/pi/picluster
5 source $PI_confdir
6
7 bash $PI_local_repo_dir/inv-gen.sh 1>/dev/null 2>&1
8
9
10 pgrep ansible > /dev/null
11 sts=?
12
13 if [ $sts -ne 0 ]; then
14     ansible-playbook -i /home/pi/inventory $PI_local_repo_dir/master.yml > /home/pi/ansible.log
15     ansible-playbook -i /home/pi/inventory $PI_local_repo_dir/nodes.yml >> /home/pi/ansible.log
16 fi
17

```

Figure 14: masterRun.sh

This script uses a similar approach to the run.sh script, by loading the local variables file (~/.picluster) and runs some checks before executing the Ansible playbooks.

4.7.2 Inventory generator

For Ansible playbooks to work, an inventory file containing the list of nodes to perform the commands on, is needed. This is generated by the “inv-gen.sh” script (figure 15).

```
11
12 echo "[master]" > $PI_local_repo_dir/inventory
13 cat $PI_local_repo_dir/ip/$(cat $PI_local_repo_dir/master) 2>/dev/null >> $PI_local_repo_dir/inventory
14 rm $PI_local_repo_dir/ip/$(cat $PI_local_repo_dir/master)
15
16 echo "[nodes]" >> $PI_local_repo_dir/inventory
17 cat $PI_local_repo_dir/ip/* 2>/dev/null >> $PI_local_repo_dir/inventory
18 cat $PI_local_repo_dir/ip/* 2>/dev/null > /home/pi/mpiNodes
19
20 echo "
21 [nodes:vars]
22 ansible_ssh_user=pi
23
24 [master:vars]
25 ansible_ssh_user=pi
26 " >> $PI_local_repo_dir/inventory
27
28 cp $PI_local_repo_dir/inventory /home/pi/ansible-inventory
29
```

Figure 15: inv-gen.sh

The “inv-gen.sh” script will initially put the master node’s IP address to the top of the inventory file and add it to the “master” group (which only contains itself). Then it will remove the IP file before concatenating all other IP files in the “ip” directory. I found this the easier and most clean way to do it, in order to avoid using complex regular expressions that I would need if I wanted to exclude a single file (Unix Stackoverflow 2015). Removing the master’s IP file is insignificant, as the next time the “run.sh” runs the “up2git.sh” (figure 11) it will reset the local repository back to the remote’s version.

Finally, it adds a useful variable for Ansible to perform without problems (ansible_ssh_user). The new inventory file is saved both in the repository’s directory but also into the home directory of the master node to avoid being overwritten by any other change.

At this point all the shell scripts that are used have been explained so I will be moving on to the Ansible section.

4.8 Ansible

The Ansible part was another challenging part of the project. I have spent a lot of time researching (Heap 2016) how it works in order to make the playbooks close to the specification. One problem I only found after I wrote the playbooks was that the Ansible version I was using to write playbooks on my computer was the current release (version 2.3) instead of the old version of Ansible that is available in the Raspbian repository (version 1.7).

This led to complications as the recent release contains several notations and modules that were not included in v1.7 (which is dated August 2014 (Python Software Foundation 2014)). One example of such feature is the “become” directive used in tasks that will use clever algorithms to become the super user on the node that is configured to perform administrative tasks.

Of course, when I got notified of the error I figured that the problem was with the outdated version and found a solution for the 1.7 release. Instead of using “become” I used the “sudo” directive in the playbook that achieves the same result.

```
nodes.yml 124 Bytes
1 ---
2 - name: run base config
3   hosts: nodes
4   remote_user: pi
5   sudo: true
6   roles:
7     - core
8     - openmpi
9     - nfsClient
```

Figure 16: nodes.yml

```
master.yml 86 Bytes
1 ---
2 - hosts: master
3   user: pi
4   sudo: yes
5   roles:
6     - core
7     - nfs
8     - openmpi
```

Figure 17: master.yml

The “node.yml” shown in figure 16 runs the base configuration (figure 18), installs the OpenMPI library with mpi4py (a library to run Python scripts written for MPI, figure 19) and then installs and configures the NFS client.

Similarly, the master playbook (figure 17) runs the same roles but with NFS being different, for the master node it installs the NFS server and configures it accordingly.

The roles are explained below.

```
1 ---
2 - name: enable ssh on boot
3   service:
4     name: ssh
5     enabled: yes
6
7 - name: update apt
8   apt:
9     update_cache: yes
10    cache_valid_time: 7200
```

Figure 18: common role playbook

```
1 ---
2 - name: install the required packages for OpenMPI
3   action: apt package={{item}} state=installed
4   with_items:
5     - openmpi-bin
6     - openmpi-checkpoint
7     - openmpi-common
8     - openmpi-doc
9     - libopenmpi-dev
10    - python-mpi4py
```

Figure 19: OpenMPI role playbook

The common playbook runs on all the nodes, including the master and ensures the SSH daemon is enabled, and then updates the package manager if the local cache is older than two hours (the value is in seconds) (Shah 2015).

The OpenMPI playbook that runs on all the nodes as well, installs only the necessary packages for me to showcase that the OpenMPI can run.

To achieve this, OpenMPI software needs to be installed, SSH connectivity needs to be established and a shared storage drive needs to be used for all the nodes to be able to get the scripts to run. So far OpenMPI and SSH are working with no issues and what is left to configure is the shared folder over the network.

The best approach for this, I concluded, was NFS, mostly because it is straightforward to install and configure and is suggested by other OpenMPI tutorials as the software that should be used for such installations (Geerling 2015, DeConinck 2013).

```
1 ---
2 - name: install the required packages for NFS
3   action: apt package={{item}} state=installed
4   with_items:
5     - nfs-kernel-server
6     - nfs-common
7
8 - name: create shared directory for NFS
9   file: path=/share state=directory mode=777 owner=root group=root
10
11 - name: NFS config file
12   action: template src=exports.j2 dest=/etc/exports
13
14 - name: rpcbind is running
15   action: service name=rpcbind state=started enabled=yes
16
17 - name: NFS is running
18   action: service name=nfs-kernel-server state=started enabled=yes
19
20 - name: exportfs
21   action: command exportfs -a
```

Figure 20: the master node NFS playbook

When I began writing this Ansible playbook I did not know what I needed to create for an NFS setup to work. I have checked tutorials online and the Ubuntu documentation (Ubuntu Help 2014) but the most helpful resource was the FreeBSD handbook, particularly the chapter explaining NFS (Swingle and Rhodes n.d.). Of course, that chapter did not provide me the specific commands I needed for Ansible, but using my experience of Unix and the guide from the handbook I compiled a list of what the playbook needs to do in order to complete the setup.

After installing the required packages, a folder needs to be created that is going to be shared, I chose to use the `/share` path, which makes it easy to identify from the user that runs MPI commands.

Shown in figure 21 is the `exports.j2` template that will create the necessary `/etc/exports` file which is required by NFS to give access to other network clients. The template is filled by a “magic” variable of Ansible that inserts the list of the group “nodes” specified in the inventory (Shah 2015). The resulting file can be seen in figure 22. This file is generated and updated every time the playbook runs which means that it will always be kept up to date with new nodes joining the cluster.


```

exports.j2 81 Bytes
1  {% for host in groups['nodes'] %}
2  /share {{host}}(rw,no_root_squash)
3  {%endfor %}

```

Figure 21: exports.j2 template

```

pi@raspberrypi:~ $ cat /etc/exports
/share 192.168.1.166(rw,no_root_squash)
/share 192.168.1.67(rw,no_root_squash)
/share 192.168.1.213(rw,no_root_squash)
/share 192.168.1.211(rw,no_root_squash)
/share 192.168.1.110(rw,no_root_squash)
pi@raspberrypi:~ $

```

Figure 22: the resulting /etc/exports file

Finally, the playbook ensures “rpcbind” and the NFS services are enabled for the system and then runs the “exportfs” command which updates NFS with the new information in the “/etc/exports” file (Kirch and Brown n.d.).

```

1  ---
2  - name: shared directory for nfs
3    file: path=/share state=directory mode=777 owner=root group=root
4
5  - name: mount the share on the remote
6    action: command mount {{groups.master}}:/share /share

```

Figure 23: the NFS client playbook

The playbook for the NFS clients is much simpler, because it only needs to connect to the remote service and mount it. Again, I am using the same path for the shared folder under “/share”. The groups.master notation, similarly to the exports.j2 template (figure 21) inserts the single master IP and mounts the remote folder to “/share”.

At this point all the intelligence behind the automation of the auto-configuration of the cluster has been explained. Figure 24 shows at a higher level how this process works. To summarise:

The nodes get the repository from GitLab, then they update back with their IP address.

The master gets the updated repository with the list of the IPs, and then sets them up using Ansible.

Then the user can run MPI scripts from the master directly.

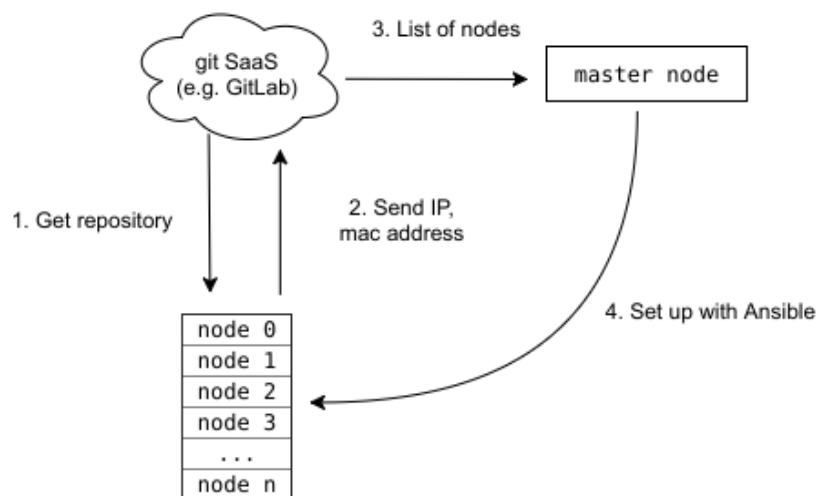


Figure 24: Automation diagram

4.9 Python user interface

The final part of the development process was to create the necessary scripts that will guide the user to create the SD cards for the nodes.

After significant research and deliberation with my supervisor we concluded to just make a simple shell interface that will work, because of time constraints. Since the project is open source I, and others will be contributing new features at a later time, and a graphical interface could be one of them.

For the interface scripting I chose to use Python because I found Bash quite challenging and I am better prepared to use my existing knowledge of Python that I have gained in the University. Another factor to influence my decision was that Python makes it much easier to manipulate strings, that I am going to use in the interface as I will demonstrate next. Being an advocate of newest technologies I also have decided to use Python3 instead of the old Python2, which is “end of life” marked and should not be used in the industry anymore. The script does not have any special dependencies that are not already included in the standard Python3 installation.

The sole purpose of the “initSD.py” script is to let the user easily create the SD cards for both the master and the slave nodes. The only thing that is required before running the script is for the user to have downloaded and extracted the Raspbian zip to their home directory and have an SD card plugged in their system. During the run time of the script it will wait for user input in various places, thus letting the user decide and take action per their needs.

The script will first try to guess which Unix device already connected to the system that looks like a potential target to write to (figure 25). This usually is a device connected to a USB port and not a serial connection (even the included SD readers are recognised by the system as USB devices so that should work with no issues). The list of active mounts of (and including) the device will be displayed for the user to choose. A “safeoptions” mechanism has been included to prevent accidental writes to the main disk device thus destroying the user’s OS installation.

```
68 drivescmd="ls -al /dev/disk/by-path/*usb*"
69 findcmd="find -P ~ ! -path '*Trash*' -and -name '*raspbian*.img'"
70
71 print()
72 msg={
73     'main':'Check the list of usb mounted drives above, please pick the one NOT ending in ',
74     'linux':'\na number such as sdb1, but instead the one without it e.g: sdb',
75     'darwin':'\ns1/s2 and so on, but instead the one without it, e.g: disk2',
76     'main2': '\nIf you make a wrong choice the sd card will not work.'
77 }
78 if not sys.platform in msg.keys():
79     sys.exit('platform not supported yet')
80
81 safeoptions=list()
82 for line in run(drivescmd)[0]:
83     drive=line.split("/")[-1]
84     safeoptions.append(drive)
85     print("/dev/"+drive)
86
87 if len(safeoptions)<1:
88     sys.exit('Error finding a suitable device to write on')
89
90 print(msg['main']+msg[sys.platform]+msg['main2'])
91
92 print('enter here:')
93 input1=input("/dev/")
94 if input1 not in safeoptions:
95     sys.exit('choice not in the list, will exit')
96
```

Figure 25: Device selection

Afterwards, the script will look into the home folder and find any files that contain “raspbian” and end in “.img” to be used (figure 26). A list will be displayed for the user, to choose which file they want to write to the SD card. This way the tool will be working with future updated versions of Raspbian and no hardcoded version needs to exist in the source code, making it more agile.

```

99
100 # will prepare for dd now
101 msg="Select the img file you want to write to "+"/dev/"+input1
102 print(msg)
103 print()
104
105 choices={}
106 for number,line in enumerate(run(findcmd)[0],start=1):
107     choices[number]=line
108     print(str(number)+" | "+line)
109 input2=input("enter the correct number: ")
110 # print(choices.keys())
111 try:
112     input2=int(input2)
113     if input2 not in choices.keys():
114         sys.exit('choice not in the list, will exit')
115 except:
116     sys.exit('input was not a number, will exit')
117
118 ddcmd='sudo dd if=%s bs=4M of=%s' % (choices[input2],'/dev/'+input1)
119 print()
120 print('will run this:')
121 print(ddcmd)
122 agreed=input("""do you agree? Check the command again and make sure
123 it is correct!\n Proceed? y/N: """)
124

```

Figure 26: Raspbian image selection

The next dialog (figure 26) will display to the user what the script wants to run (the full dd command) along with a notice before agreeing.

When the user accepts, the script will start the dd copy, requiring the sudo password, and take around five minutes, depending on the SD card writing speed (figure 27). Then when it is finished in order to be able to properly mount the new partitions, the system’s partition table needs to be reloaded, using the “blockdev” command (Maroudas 2016).

```

125 if agreed == 'y':
126     print('You might get asked your sudo password:')
127     print("""The dd command will take a few minutes to finish depending
128 on the size of the image and the SD card speed. You will get a
129 confirmation when finished.""")
130     print('Unmounting /dev/%s'%input1)
131     unmount(input1)
132     print()
133     run(ddcmd)
134     print()
135     print('finished')
136     print('now reloading the partition table')
137     print()
138     cmd='sudo blockdev --rereadpt /dev/%s' % input1
139     run(cmd)
140     print('finished reload, moving on to mount')
141     print()
142 elif agreed=='skip':
143     pass
144 else:
145     print('Aborted by user')
146     sys.exit(0)

```

Figure 27: dd process and partition reloading

Following the previous process, the script copies the SSH keys from the user’s “.ssh” directory (of course after their confirmation) and intelligently finds the git repository URL from the current repository this script is running on. It displays the result for user confirmation/alteration before it writes the variables to the SD card.

A final question to the user is whether the SD card is going to be used on a master node. The choice is as simple as a “y” or just hitting the return key.

All the questions asked in the “initSD.py” script I have decided to add one by one, by what has been necessary at the time and with feedback from my supervisor on what an academic would want to have in such a script. Thus, I am confident I am not using an overly complex structure for this script (since it is just a question/action format) and more importantly, I do not tire the user.

4.10 SSH to master and Documentation

The “sshmaster.sh” script is a small single line script purposed to make it easy for the user to connect to the master node as soon as it is up and running.

By default, this contains the following command (figure 28) which, when run directly from inside the repository on the user’s machine can directly connect to the master server. An AsciiCast example will be included as a link in the evaluation section.

A screenshot of a terminal window titled "sshmaster.sh" with a close button "x" in the top right corner. The terminal shows a single line of code: "1 ssh pi@\$(cat ip/\$(cat master))|". The line number "1" is highlighted in a light grey box. The code uses shell variables and command substitution to dynamically determine the IP address of the master node.

Figure 28: sshmaster.sh script

The user can start imaging the SD cards by navigating to the “python-scripts” directory and executing the initSD.py script by running “python3 initSD.py”. The script will guide the user and gracefully exit if problems arise.

The documentation on how the files work and explaining the repository structure has been written inside the repository’s Readme file, written in markdown. This way the whole project can be in one location, and can be easily organised using git.

The markdown viewer that is included in the web interface of GitLab and GitHub further help present the Readme document in a much cleaner way, as can be seen in figures 29 and 30.

The picluster repository will have the necessary files for the ansible-backed raspberry pi beowulf cluster project.

File structure of the repository

`/ip/`

Contains the generated IP (individual) files for all nodes (including master).

`README.md`

This file you are reading, contains all relevant information about the project.

`ip2file.sh`

Saves the current IP of `eth0` to the `/ip/<eth0-MAC-ADDRESS>` file, and overrides it, if exists.

`up2git.sh`

Pushes to git the new IP that has been generated by `ip2file.sh`.

`/local-vars/`

Directory that will contain all the variables for the individual deployment each time. For example the MAC address of the master node, the public SSH key that the master will use, etc.

`/master`

File which will contain the MAC Address of the designated master node. On fresh repositories this file will not exist, but will be generated from the first node that saves its IP in `/ip/` directory. If the file exists (either because the Pi is not the first to boot and add its IP or because the user has added the file manually) it will not be touched.

Figure 29: Readme, part 1

`/boot/picluster/master`

Exists only on the designated master sdcard/node. Will be checked by the `/node-master.sh` script.

`/boot/picluster/priv.key`

The private key that ought to have write access to the git repository. Otherwise nothing will work. This key is going to be used for regular SSH as well. It is going to be copied to `/home/pi/.ssh/id_rsa` automatically.

`/boot/picluster/pub.key`

The public key for the aforementioned private one. It is going to be copied to `/home/pi/.ssh/id_rsa.pub` automatically.

Processes

Raspberry Pi boot process

After the Debian process finishes, the scripts in `/etc/rc.local` will run. In those scripts there will be initialisation procedures that will get the configuration from `/boot/picluster.conf` and parse it accordingly.

The script will also copy the ssh keys to `/home/pi/.ssh/`

Given there is internet connection the script will clone the git repository into `/home/pi/picluster` (now referred to as `$LOCALREPODIR`) and decide if it is a master and add the master file in the root of the directory. Then it will add its IP in `$LOCALREPODIR/ip/` and commit-push back.

Then it will set up a process to be added as a service in the system so that it never gets killed.

That service will from now on handle everything and the initial script can terminate.

If the Pi is configured to be a master, it will initiate the master script.

Master

Figure 30: Readme, part 2

5. Evaluation - Results

To evaluate this project, we need to consider the original research question. The title of this report is:

“An Ansible implementation of a self-configuring Beowulf cluster of Raspberry Pis in a localised environment for the purpose of distributed computing using Open MPI”

So, the research question is, as defined in the beginning, if we can create a set of tools that will enable any number of Raspberry Pis to automatically configure using Ansible and successfully set themselves up for OpenMPI usage.

5.1 Deliverables

After completing the development and testing phase of this project I can acknowledge that this goal has been achieved. To elaborate, I will go over the list of deliverables defined after the literature review and in the beginning of the method chapter (section 4.1 Project deliverables).

a. Network independency

One of the important deliverables of this project was that the cluster had to be network agnostic, meaning that it should be able to self-configure regardless of the network.

As I have explained in the 4.6 chapter, the IP update scripts will have nothing to do with the network stack of the system, but only get the current IP address and share it over git. So, as long as any other system on the same network (desktops, laptops, smartphones) can get an IP address through DHCP, so can the Raspberry Pis. The IP then will be updated with the repository and shared to the rest of the cluster. In addition, if the network also allows direct connections using IP address (like SSH connection) then the cluster will have no problem configuring, as it is only using SSH protocols to communicate to git and between the nodes.

b. Unattended setup

This was delivered as well by the project. The scripts described in sections from system start-up injection to bash updating repositories and IP addresses through cron, and even the higher-level configuration being done by Ansible, all converge to a truly unattended setup. The user only has to prepare the devices physically (plug in the SD card, the power adapter and the Ethernet cable), which of course is unavoidable. The rest is being done automatically.

c. Little input from the user

This deliverable asked for as little input from the user as possible in the initial stage of creating the SD card. This, as well has been met, as the user only answers simple questions, like which device contains the SD card, which image to write to it and if the SD card is destined to a master node. The rest of the questions can be skipped with an enter as they have default values hardcoded, or guessed intelligently as shown in the 4.9 Python chapter.

d. Easy access to the master

This, although an easy task, is one of the most useful deliverables as it lets the user quickly and with no frustration connect to the master node. As shown in the last (4.10) section of the methodology and as I have recorded in an AsciiCast later this has been implemented and is as easy as running a single script, “sshmaster.sh”. All the SSH keys are already there on all nodes (including the master) and the connection is seamless and with no errors.

e. Documentation

Documentation has been written in the front of the web interface of the git repository on GitLab in form of a Markdown document called Readme. This serves the double purpose of being very visible (as it is the first thing displayed in the repository) and being easy to edit and update by being a simple and elegant mark-up language.

5.2 Demonstration

The previous discussion has argued that the deliverables have been met. To demonstrate this I also have recorded two “Asciicasts” which, as explained in the 2. Term definitions, are a variation of screencasts but displaying only the presenter’s shell, making them an excellent choice for the task.

The following URL points to the Asciicast that demonstrates how the user can create a new SD card using the Python tool:

<https://asciinema.org/a/5zn8w9c3p4pdffc86co86kfx8>

After the user creates the necessary node SD cards and the single master SD card using the Python script demonstrated above, then they will turn the devices on and let them auto-configure. The process does take at least 6 minutes depending on the Internet bandwidth of the network as all the nodes when booted start to download the updated package repository list and install Ansible, git and vim as explained in 4.5 run.sh section.

During this time, of course, the user can be watching how the git repository changes as the updates will be regular commits there. When the master has submitted the master commit, the user will be able to connect to it and start using the cluster (even if not all nodes are active yet).

The following URL points to the Asciicast that shows how the user can easily connect to the master node and run OpenMPI commands successfully. In the demonstration, I have used a python script that prints “hello world” from each node in the cluster.

<https://asciinema.org/a/4liwwumq4trab65091w3thxtk>

6. Project management

In order to efficiently develop the required software for this project I initially planned the 14 weeks I had available in a simple way as I have demonstrated in the project proposal (Appendix 11.1). In the proposal, I planned each step of the process that I had envisioned then in its small-compact time slot, more like how an agile team using the Scrum framework would have worked. The project was split into small sprints and after each cycle I would mark the task off the product backlog and continue to the next in line.

Since I was working alone and not as part of a team, my focus was at the current sprint and not the previous or next ones, making the process as clean as possible and requiring a certain amount of dependencies met before I could proceed to the next step. Based on the initial plan I had also created a simple Gantt chart (figure 31) that shows exactly how each task does not overlap with others (apart from 2-3 occasions that the tasks would be easy to do at the same time, like the test fix and documenting near the end of the timeline).

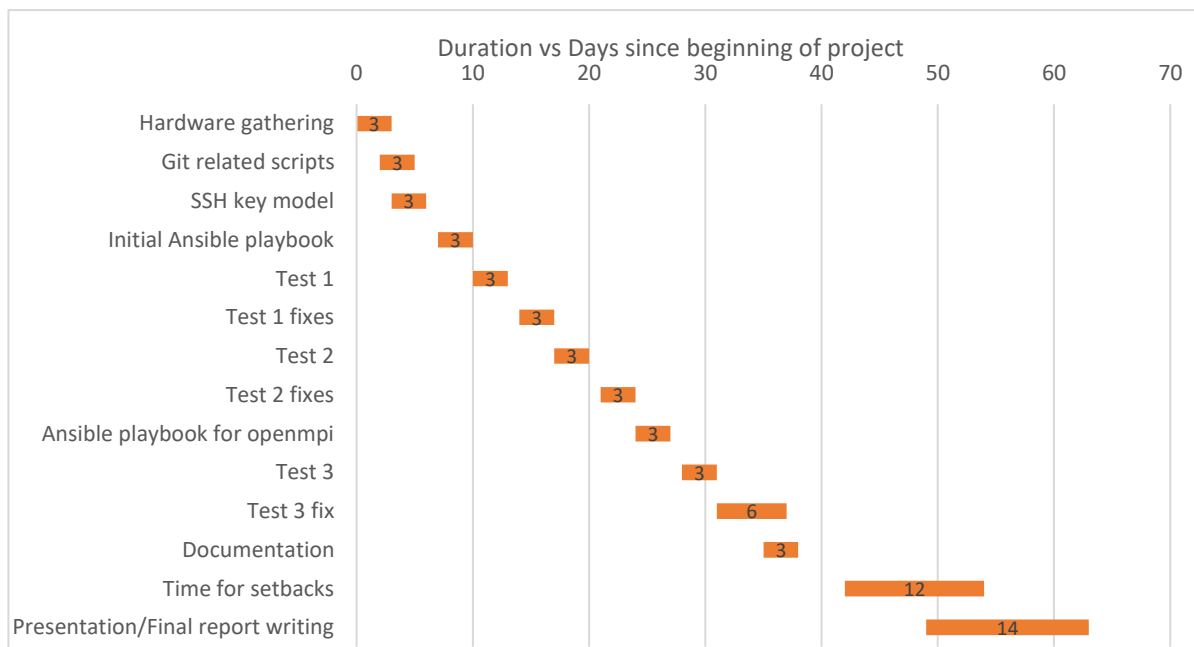


Figure 31: Initial Gantt chart

The initial plan albeit being thorough, did not include difficulties encountered over time. It appears I have underestimated the time it would take to learn about the Ansible and the testing of the Python script. These tasks required considerable more time than allotted and as a result I had to use the “time for setbacks” period to fix them.

Over the final weeks and after the presentation meeting with my supervisor I have tried to follow his instructions to focus mostly on the cluster working rather than too complicated and specialised parts, such as a better interface or an additional zeroconf method of configuring.

After the presentation meeting I took the chance to view the project from a different perspective in order to find how I would prioritise best (a list of handwritten notes I took while discussing with the supervisor after the presentation can be found in the Appendix’s handwritten notes section 11.4 figure 2). From the 302CEM module I was introduced to the MoSCoW model (Rasmusson 2010) of prioritising tasks and managed to adjust the rest of the tasks to it.

7. Discussion

This project began as a question in the university's computer club: "Why can't we use all the Raspberry Pis we have to demonstrate clusters to students?"

That was an intriguing question that led me to discuss with my supervisor ways I could accomplish it. Soon the conclusion was made, that we would not be able to auto discover the nodes because of the university's network restrictions. Eventually, this led me asking my supervisor if I could research and implement a way to make this possible.

By the end of this module I am happy to announce that the project was a success and the university will be able to use the Raspberry Pis we have available to demonstrate easily with no special attention a cluster of Linux machines running MPI software.

Although everything seems well and good, the project did have a small number of setbacks that led to re-organising priorities and plans and deciding which feature wouldn't be implemented after all. Of course, being a supporter of the open source ideology that was not a problem, as the project will be shared publicly and everyone will be contributing in the future, including me.

One of the things that are not included in the current version of the code is the lack of an one-line unattended SD card imaging program. While, as demonstrated, it is a fairly easy process to create the cards, I would like to add in the future the option to run the "initSD.py" script with a number of arguments that will automatically start doing the work instead of asking the user for information. After all, this tool could be used by technical people that would prefer to input the information before the script even starts.

Another thing that I wish I had more time to implement is a better user interface. I initially planned for a "curses" like interface but that part alone would cost several weeks of trial and error and problematic attempts. Again, this is added in the To-do list for the project in the future.

For the time being the Python script has been tested against Linux systems, specifically Ubuntu ones. I am confident that it will work with Fedora as well, but since I do not personally own a macOS device yet, I won't be able to develop for that efficiently, and will rely on other people's bug submissions or even fixes. I do not plan to develop on Windows for the time being because of the lack of a Unix shell, or until I learn PowerShell like I do Bash.

As described this project relies on the SaaS git provider GitLab. For larger installations, the number of requests sent to their service will scale in a proportional manner, and that might become a problem for the provider. I have not yet hit that limit with six devices running and requesting git updates every two minutes amounting up to 180 requests per hour (6 nodes times 30 runs per hour).

For each node added in the cluster one can expect 30 more requests per hour, which can easily scale since the very purpose of this project is to provide for an easily scalable cluster. To overcome this problem a user can host their own repository or use the university's own GitHub server (github.coventry.ac.uk).

Another limitation of this project this far is the fact that Ansible runs every three minutes on the master. I would like to make it run only once or twice so that it will be more efficient. To overcome overlapping problems if one Ansible is not finished in the three-minute window, I have added a

check on the masterRun.sh script to check if the process “ansible” is running and continue only if it is not.

8. Reflection on ethics

8.1 Social aspect

As I have mentioned earlier, the target audience for this project is academics in higher education that want to use clusters running MPI software to demonstrate distributed programming to students. More specifically, Coventry University was the main target so that the project would be used in the parallel programming module taught to final year students. This project does not interfere with other networks and as such cannot be considered harmful to anyone, but instead is promoting education by making teaching materials (for example a cluster) easier to set up.

The project is based on open source software, a community effort and thus I would not feel morally intact if I did not give it back freely and openly so that others may benefit from it.

8.2 Legal aspect/Licensing

As already clarified, the project makes use of open source libraries, programming languages and operating systems. It is meant to work with such software and takes advantage of the openness of said products.

Although I could skip the part of choosing a licence for the project’s code, that would make the work copyrighted, non-free and not available to usage without my prior consent, as has been pointed out by Richard Stallman (2015).

Furthermore, I am not bound by the licences for the software that my scripts use, because I am not using any of their code in my own work. I am merely running scripts on systems that use other open source software. This gives me the right to choose my own licence at this point.

Because I did not want the users of my scripts to be bound by any open source licence I chose to use the “Unlicence” aiming to rid the work from any restriction and obligation (Unlicence.org n.d.).

8.3. Ethics

I could not find any fault on the ethical aspect for this project since it does not involve personal information of any kind, it does not try to hack computers and does nothing that is not already shown in the public repository that the user can load and inspect.

9. Conclusion

In this project, I have demonstrated how I created a set of tools that work together to enable the user to set up and configure an OpenMPI cluster of Raspberry Pis using Ansible and git with minimum intervention.

The evaluation of this project returned positive results that the cluster starts to configure promptly and automatically when the devices are powered on and connected to the internet. The set-up time varies depending on the network bandwidth but usually is in the timeframe of 6 - 10 minutes for each Raspberry Pi connected.

10. Bibliography

- Bahyl, V., Chardi, B., Van Eldik, J., Fuchs, U., Kleinwort, T., Murth, M., and Smith Cern, T. (2003) *Installing, Running and Maintaining Large Linux Clusters at CERN*. [online] available from <<https://arxiv.org/pdf/cs/0306058.pdf>> [30 April 2017]
- Becker, D.J., Sterling, T., Savarese, D., Dorband, J.E., Ranawake, U.A., and Packer, C. V. (1995) *BEOWULF: A PARALLEL WORKSTATION FOR SCIENTIFIC COMPUTATION* [online] available from <<http://www.phy.duke.edu/~rgb/brahma/Resources/beowulf/papers/ICPP95/icpp95.html>> [30 April 2017]
- Broberg, R. (2012) *OpenMPI on Raspberry Pi* [online] available from <<https://rhinohide.wordpress.com/2012/02/26/openmpi-on-raspberry-pi/>> [30 April 2017]
- CERN (2012) *The Grid: Software, Middleware, Hardware*. [online] available from <<https://cds.cern.ch/record/1997392>>
- Chacon, S. (2014) *Pro Git*. Berkeley, CA New York, NY: Apress, Distributed to the Book trade worldwide by Spring Science+Business Media
- Cox, S.J., Cox, J.T., Boardman, R.P., Johnston, S.J., Scott, M., and O'Brien, N.S. (2014) 'Iridis-Pi: A Low-Cost, Compact Demonstration Cluster'. *Cluster Computing* [online] 17 (2), 349–358. available from <<http://dx.doi.org/10.1007/s10586-013-0282-7>>
- DeConinck, A. (2013) *Ansible Scripts for My Raspberry Pi Cluster* [online] available from <<https://github.com/ajdecon/ansible-pi-cluster>> [30 April 2017]
- DigitalOcean (2015) *Understanding Systemd Units and Unit Files | DigitalOcean* [online] available from <<https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files>> [1 May 2017]
- Gardner, G. (2014) *Mini-Itx Cluster* [online] available from <<http://www.mini-itx.com/projects/cluster/>> [30 April 2017]
- Geerling, J. (2015) *Raspberry Pi Drabble* [online] available from <<https://github.com/geerlingguy/raspberry-pi-drabble>>
- Gropp, W., Lusk, E., and Sterling, T. (2003) *Beowulf Cluster Computing with Linux*. 2nd edn. Cambridge, Mass: MIT Press
- Heap, M. (2016) 'Appendix A. Installing Ansible'. in *Ansible: From Beginner to Pro* [online] Berkeley, CA: Apress, 159–162. available from <http://dx.doi.org/10.1007/978-1-4842-1659-0_10>
- Kiepert, J. (2013) *Creating a Raspberry Pi-Based Beowulf Cluster*. [online] available from <http://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf> [30 April 2017]
- Kirch, O. and Brown, N. (n.d.) *exportfs(8) - Linux Man Page* [online] available from <<https://linux.die.net/man/8/exportfs>> [1 May 2017]
- Maroudas, E. (2016) *#825340 - Sfdisk: Invalid Option -- 'R' on Restoredisk Mode - Debian Bug Report Logs* [online] available from <<https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=825340;msg=2>> [30 April 2017]
- NSF (2005) *\$150 Million TeraGrid Award Heralds New Era for Scientific Computing | NSF - National Science Foundation* [online] available from

- <https://www.nsf.gov/news/news_summ.jsp?cntn_id=104248> [30 April 2017]
- Nvidia and DeConinck, A. (2013) *How One NVIDIA Built a Tiny Server Cluster Out of a Slice of Raspberry Pi* | *The Official NVIDIA Blog* [online] available from <<https://blogs.nvidia.com/blog/2013/07/17/raspberry-pi/>> [30 April 2017]
- Python Software Foundation (2014) *Ansible 1.7 : Python Package Index* [online] available from <<https://pypi.python.org/pypi/ansible/1.7>> [1 May 2017]
- Rasmusson, J. (2010) *The Agile Samurai : How Agile Masters Deliver Great Software*. Raleigh, North Carolina: The Pragmatic Bookshelf
- Raspberry Pi Foundation (2014a) *Rc.local - Raspberry Pi Documentation* [online] available from <<https://www.raspberrypi.org/documentation/linux/usage/rc-local.md>> [1 May 2017]
- Raspberry Pi Foundation (2014b) *Scheduling Tasks with Cron - Raspberry Pi Documentation* [online] available from <<https://www.raspberrypi.org/documentation/linux/usage/cron.md>> [1 May 2017]
- RedHat (2015) *Creating and Modifying Systemd Unit Files* [online] available from <https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/sect-Managing_Services_with_systemd-Unit_Files.html#tabl-Managing_Services_with_systemd-Service_Sec_Options> [1 May 2017]
- Shah, G. (2015) *Ansible Playbook Essentials*. Birmingham: Packt Publishing
- Stallman, R. (2015) *Re: Please, No GitHub* [online] available from <<https://lists.gnu.org/archive/html/discuss-gnustep/2015-12/msg00182.html>> [2 May 2017]
- Stephenson, J. (2005) *The Humidor Cluster* [online] available from <<http://slipperyskip.com/page10.html>> [30 April 2017]
- Sterling, T. (2002) *Beowulf Cluster Computing with Linux*. Cambridge, Mass: MIT Press
- Swingle, B. and Rhodes, T. (n.d.) *Network File System* [online] available from <<https://www.freebsd.org/doc/handbook/network-nfs.html>> [29 April 2017]
- Ubuntu Help (2014) *Network File System (NFS)* [online] available from <<https://help.ubuntu.com/14.04/serverguide/network-file-system.html>> [1 May 2017]
- Unix Stackoverflow (2015) *Text Processing - Cat Files except One - Unix & Linux Stack Exchange* [online] available from <<https://unix.stackexchange.com/questions/246048/cat-files-except-one>> [1 May 2017]
- Unlicence.org (n.d.) *Unlicence.org » Unlicence Yourself: Set Your Code Free* [online] available from <<https://unlicence.org/>> [2 May 2017]
- Walters, C. (2012) *Efficiency of Git versus Tarballs for Source Code Transmission and Storage over Time* | *Colin Walters* [online] available from <<https://blog.verbum.org/2012/06/08/efficiency-of-git-versus-tarballs-for-source-code-transmission-and-storage-over-time/>> [30 April 2017]

11. Appendix

11.1 Project Proposal

300/303COM Detailed Project Proposal

First Name:	Theocharis
Last Name:	Ledakis
Student Number:	5717206
Supervisor:	Dr Carey Pridgeon

SECTION ONE: DEFINING YOUR RESEARCH PROJECT

1.1 Detailed research question

How could the process of deploying a Raspberry Pi cluster in a local network be automated in order to achieve minimal configuration from the user?

1.2 Keywords

MPI; Raspberry Pi; Ansible; automation; Beowulf cluster; distributed computing

1.3 Project title

An Ansible implementation of a self-configuring Beowulf cluster of Raspberry Pis in a localised environment for the purpose of distributed computing using Open MPI.

1.4 Client, Audience and Motivation:

This project's client is Dr. Carey Pridgeon, who has requested for a set of tools that can be used in higher education environments to easily set up any number of Linux nodes. This cluster will have open-source software configured and its purpose will be to be used as teaching material as well as distributed computing system.

One of the key software required by the client is open MPI to be running on all nodes for computation purposes. Furthermore, as the project question suggests, the final product will require the minimum amount of user configuration to implement. This leads to the second major requirement, a way for the individual nodes to make themselves known to the rest, irrespective of local network policies that possibly prevent message broadcasting.

The intended audience is Coventry University students that study modules involving parallel/distributed computation as well as the academics that deliver those modules.

The implementation that is going to be followed during this project is unique in nature and I strongly feel that will provide considerable help to users that want to create their own cluster of Linux nodes. This will not be restricted to Raspberry Pis but rather can be applied to any Linux machine running on the most popular architectures (arm, i386, amd64). The fact that git repositories will be used makes this project useful for environments that restrict broadcasting, and thus provides the ability to configure the cluster without advanced technical knowledge.

1.5 Primary Research Plan

The implementation plan will be consisted by the following tasks; a tablet showing the designated weeks those tasks are planned to be completed follows:

1. Initialize the 8GB SD cards for this project, with Rasbian, correct permissions, updates, appropriate users. Create a repository on the University's GitHub or on GitLab, share access with supervisor. Get the project's hardware from supervisor (at least 16 Raspberry Pis, chargers, cables). Research and make a request for a switch to try them later on.
2. Write necessary scripts/configuration to advertise the nodes' information on a shared git repository.
3. Create a security model using ssh client-server keys to implement a trust model between all the nodes.
4. Create an Ansible playbook that will be used to implement all initial steps to configure the images. There will be one or more playbooks that can all be used on the same node (for example, node and master node).
5. Test 1: Deploy on three Raspberry Pis and check the code works (they manage to establish connection between them). Each node should be able to get the table of addresses/hostnames of the rest in reasonable time and each can ssh to each other using the preset ssh keys.
6. If there are problems with the first test, allowing three days' time to solve them.
7. Test 2: Deploy on 16 or more Raspberry Pis with one acting as a master node. All should, in reasonable time, discover the master node and advertise their address/hostname to the repository. The master node should be able to ssh to all of them using the preset ssh keys.
8. If test 2 introduces problems, allow a week to fix.
9. Create Ansible playbook that will install and configure open-mpi as well as the rest of the software needed for the purposes of the project. Deploy to a master and a regular node.
10. Test 3: Re-image the (previously tested) Raspberry Pis from test 2 to include the Open MPI tests and test the following:
 - a. SSH connectivity between the nodes (most important is master <--> node)
 - b. Open MPI configuration working with all nodes in the cluster.
11. If there are problems with test 3, allow a week to fix them.
12. Write documentation for the project. The documentation will include a network map, a guide for deploying the images to SD cards, applying Ansible playbooks and additional information on how to add extra software/scripts to the cluster.
13. Allowing two weeks' time for unexpected problems/setbacks. If there is enough time, research and implement an optional alternative way for local node discovery using avahi-daemon (a zeroconf implementation) for networks that allow broadcasting. (Johns 2002)
14. Write project report and submit.

Plan table:

Week	Date	Plan Steps
1	06/12/2002	1,2,3
2	13-19/2	4,5
3	20-26/2	6,7
4	27/2-5/3	8,9
5	06/12/2003	10,11
6	13-19/3	11,12
7	20-26/3	13
8	27/3-2/4	13
9	03/07/2004	Presentation
--	8-28/4	14

SECTION TWO: ABSTRACT AND LITERATURE REVIEW

2.1 Abstract

The purpose of this study is to showcase a set of custom tools created to enhance the process of configuring any size clusters of nodes running Unix/Linux Operating Systems. In order to present the results as well as perform testing, Raspberry Pi computers will be used in a local University network that restricts broadcasting. The tools created will be usable in any Linux system and licensed under an open-source license. The outcome of this project is expected to be a fully functional product that will be used inside Coventry University for deploying clusters with minimum interaction from the user.

2.2 Summary/Mini Literature Review

Distributed computation

Distributed/parallel processing is not a recent computing development; it has been around for at least 20 years. It was conceived to battle demanding computational problems that required large amounts of processing power. The computers of the 90s were not capable to deliver, so computer scientists came up with the idea to split the computing parts of problems into small chunks and let slave nodes do the legwork in order to speed up the process. (Gropp et al. 1996)

In recent years, even though the computational power has increased tremendously, more and more projects in science need to distribute work load horizontally to achieve faster results. From the big server farms that host large companies' "cloud" applications to the very laptop we use in daily life the principles of distributing load are being applied to, very successfully, make the user's experience better. This lead to introducing this subject to the higher education curriculum and a big boost of the demand for documentation and specialised implementations of clusters and MPI software. Despite this demand, there is a small amount of projects trying to simplify the process of deploying distributed systems for computational purposes, which is the main motive in starting this project.

Raspberry Pi

The use of the Raspberry Pis is chosen because this device costs about 30 GBP and has been introduced to schools and universities all over the UK and the world. These factors help showcase the effectiveness of the final product by introducing a considerable number of Linux machines that will successfully be able to auto configure themselves into a working cluster. There will be knowledge gained from several other similar projects, and the final result will be a combination of the good parts of each implementation. Some noteworthy projects are:

- The Iridis-pi (Cox et al. 2014)
- The 2003 mpich implementation from (Karonis et al. 2003)
- The MPI specification (Gropp et al. 1996)

Useful sources of information will be the following Ansible books as well:

- Ansible Playbook Essentials (Shah 2015)
- Ansible: From Beginner to Pro (Heap 2016)

2.3 Bibliography (key texts for your literature review)

- Raspberry Pi foundation (n.d.) *SSH (Secure Shell) - Raspberry Pi Documentation* [online] available from <<https://www.raspberrypi.org/documentation/remote-access/ssh/>> [3 February 2017]
- Johns, H. (2002) *Understanding Zeroconf and Multicast DNS - O'Reilly Media* [online] available from <<http://archive.oreilly.com/pub/a/wireless/2002/12/20/zeroconf.html>> [3 February 2017]
- Shah, G. (2015) *Ansible Playbook Essentials* [online] Packt Publishing. available from <<https://www.amazon.com/Ansible-Playbook-Essentials-Gourav-Shah-ebook/dp/B00Z6VXSPW%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3DB00Z6VXSPW>>
- Karonis, N.T., Toonen, B., and Foster, I. (2003) 'MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface'. in *Journal of Parallel and Distributed Computing* [online] vol. 63 (5). 551–563. available from <<http://arxiv.org/abs/cs/0206040>> [3 February 2017]
- Heap, M. (2016) 'Appendix A. Installing Ansible'. in *Ansible: From Beginner to Pro* [online] Berkeley, CA: Apress, 159–162. available from <http://dx.doi.org/10.1007/978-1-4842-1659-0_10>
- Cox, S.J., Cox, J.T., Boardman, R.P., Johnston, S.J., Scott, M., and O'Brien, N.S. (2014) 'Iridis-Pi: A Low-Cost, Compact Demonstration Cluster'. *Cluster Computing* [online] 17 (2), 349–358. available from <<http://dx.doi.org/10.1007/s10586-013-0282-7>>
- Geerling, J. (n.d.) *IT Automation with Ansible on a Cluster of 6 Raspberry Pi Computers | Opensource.com* [online] available from <<https://opensource.com/life/16/2/cluster-computing-with-ansible-and-raspberry-pi>> [2 February 2017]
- Bondi, A.B. (1998) *Network Management System with Improved Node Discovery and Monitoring*. United States: Google Patents. available from <<https://www.google.com/patents/US5710885>>
- Techterms (n.d.) *Grid Computing Definition* [online] available from <https://techterms.com/definition/grid_computing> [3 February 2017]
- Gropp, W., Lusk, E., Doss, N., and Skjellum, A. (1996) 'A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard'. *Parallel Computing* [online] 22 (6), 789–828. available from <<file://www.sciencedirect.com/science/article/pii/0167819196000245>>

11.2 Meeting diary

2nd of February 2017

Supervisor: Dr. Carey Pridgeon

Student: _____ Theocharis Ledakis _____

Date of meeting: _____ 02 Feb 2016 _____

Key topics Discussed:

- Proposal question reviewed
- Discussed the implementation method
- Discussed deliverables
- Clarified one of the terms used in the proposal (distributed vs parallel computing)
- Agreed on the project title and implementation plan structure

Individual action points for next meeting (no more than 3):

- Finish proposal and deliver
- Submit ethics form
- Meet and get the hardware needed (Raspberry Pis) from Carey's office

9th of February 2017

Supervisor: Dr. Carey Pridgeon

Student: Theocharis Ledakis

Date of meeting: 09 Feb 2016

Key topics Discussed:

- Project progress catch-up
- Discussed about the repository I will be using, it will be GitLab
- Talked about the networking and how many I will be using this at the beginning

Individual action points for next meeting (no more than 3):

- Complete ethics form
- Start working on the framework of my literature review
- Have something working next week

Record of individual actions completed + notes:

Regarding the literature review:

- Things that inspired the project for lit review.
- Not just specifically about the project.
- Break into subject areas.

Check into 'ganglia' software to be added to the Raspberry Pi images.

Date of next meeting: 16 Feb 2019

Acknowledgement from Supervisor



Carey Pridgeon

Thu 09/02, 10:53

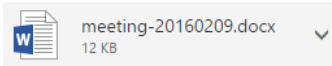
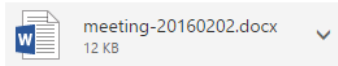
Theocharis Ledakis 

Yes I agree to these meeting record forms.



Theocharis Ledakis

Thu 09/02, 10:51



2 attachments (25 KB) [Download all](#) [Save all to OneDrive - Coventry University](#)

Hi, do you agree with the attached forms about the meetings on 2nd February and 9th of February?



Theo

16th of February 2017

Supervisor: Dr. Carey Pridgeon

Student: Theocharis Ledakis

Date of meeting: 16 Feb 2016

Key topics Discussed:

- Project progress catch-up
- Reviewed ethics form
- Talked about ways to identify the individual Raspberry Pis on the cluster

Individual action points for next meeting (no more than 3):

- Do work as agreed on the proposal for the individual weeks' timetable
- Start working on the literature review

Record of individual actions completed + notes:

As a second way of identifying the Raspberry Pis on the cluster, the status and power LEDs on them could be used to blink and show the numeric ID in a binary format. Try to design a plan for this.

Date of next meeting: 23 Feb 2019

Acknowledgement from Supervisor
Meeting notes



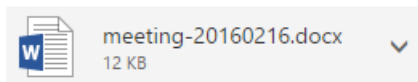
Carey Pridgeon
Thu 16/02, 10:51

agreed



Theocharis Ledakis
Thu 16/02, 10:51
Carey Pridgeon ✓

Sent Items



Download Save to OneDrive - Coventry University

Hi, these are the notes from today, agreed?

Theo

24th of February 2017

Supervisor: Dr. Carey Pridgeon

Student: Theocharis Ledakis

Date of meeting: 24 Feb 2016

Key topics Discussed:

- Introduction to Prof. Chao
- Key points for next meeting
- Discussions on initial work plan

Individual action points for next meeting (no more than 3):

- Do more work on the literature review
- Begin working on the dissertation report's structure

Record of individual actions completed + notes:

Prof. Chao might have some useful information for the development of this project.

Date of next meeting: 2 March 2017

Acknowledgement from Supervisor



Carey Pridgeon

Fri 24/02, 14:24

agreed

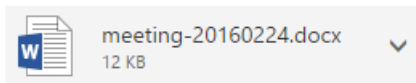


Theocharis Ledakis

Fri 24/02, 14:24

Carey Pridgeon ↕

Sent Items



Download Save to OneDrive - Coventry University

Meeting notes



2nd of March 2017

Supervisor: Dr. Carey Pridgeon

Student: _____ Theocharis Ledakis _____

Date of meeting: _____ 2 Mar 2016 _____

Key topics Discussed:

- Discussed project progress
- Plan to add more work to the report

Individual action points for next meeting (no more than 3):

- Do more work on the literature review

Record of individual actions completed + notes:

Date of next meeting: 9 March 2017

Acknowledgement from Supervisor

Re: Meeting notes



Carey Pridgeon

Thu 02/03, 10:45

Theocharis Ledakis ↕

agreed

From: Theocharis Ledakis <ledakist@uni.coventry.ac.uk>

Date: Thursday, 2 March 2017 at 10:44

To: Carey Pridgeon <ab0475@coventry.ac.uk>

Subject: Meeting notes

Agreed?



9th of March 2017

Supervisor: Dr. Carey Pridgeon

Student: _____ Theocharis Ledakis _____

Date of meeting: _____ 9 Mar 2016 _____

Key topics Discussed:

- Discussed project progress
- Literature review points

Individual action points for next meeting (no more than 3):

- Progress on the report structure

Record of individual actions completed + notes:

Literature review, discuss about:

- How it helps academics, teachers in their work
- Comparison of puppet VS Ansible
- Educational benefits of the hands-on approach:
 - Benefits of deep learning VS surface learning
 - Beneficial to younger pupils to be exposed to distributed algorithmic thinking
- Identify the need for the project to create an easy to use tool to deploy the SD cards.

Date of next meeting: 16 March 2017

Acknowledgement from Supervisor

Re: Meeting notes



Carey Pridgeon

Thu 09/03/2017 10:34

To: Theocharis Ledakis ↗

Yeah, probably

From: Theocharis Ledakis <ledakist@uni.coventry.ac.uk>

Date: Thursday, 9 March 2017 at 10:33

To: Carey Pridgeon <ab0475@coventry.ac.uk>

Subject: Meeting notes

Meeting notes, is it ok?

Theo

16th of March 2017

Supervisor: Dr. Carey Pridgeon

Student: Theocharis Ledakis

Date of meeting: 16 Mar 2016

Key topics Discussed:

- Project presentation
- Choice of Ansible as the main automation tool

Individual action points for next meeting (no more than 3):

- Progress on Ansible playbooks

Record of individual actions completed + notes:

Date of next meeting: 23 March 2017

Acknowledgement from Supervisor

Re: Meeting notes 16/03/2017



Carey Pridgeon

Thu 16/03/2017 19:09

To: Theocharis Ledakis ↗

I think so, very possibly.

From: Theocharis Ledakis <ledakist@uni.coventry.ac.uk>

Date: Thursday, 16 March 2017 at 11:13

To: Carey Pridgeon <ab0475@coventry.ac.uk>

Subject: Meeting notes 16/03/2017

Agreed?

24th of March 2017

Supervisor: Dr. Carey Pridgeon

Student: Theocharis Ledakis

Date of meeting: 24 Mar 2016

Key topics Discussed:

- What is going to be required from for the final report
- Key presentation points for next week
- Talked about the second supervisor

Individual action points for next meeting (no more than 3):

- Have the structure of the final report ready to showcase?
- Be able to showcase the project working on Raspberry Pis

Record of individual actions completed + notes:

- Detailed discussion about the delivery of project and how it is going to progress after graduation
- Talk about failure scenarios (if for example the master dies) and recoverability of the cluster

Date of next meeting: 30 March 2017

Acknowledgement from Supervisor

Re: meeting 24th march



Carey Pridgeon

Fri 24/03/2017 15:18

To: Theocharis Ledakis ↗

No!!!!

Well ok, yes

From: Theocharis Ledakis <ledakist@uni.coventry.ac.uk>

Date: Friday, 24 March 2017 at 15:15

To: Carey Pridgeon <ab0475@coventry.ac.uk>

Subject: meeting 24th march

Do you agree with the meeting notes?

11.3 Presentation material

The presentation on the 30th of March 2017 to my supervisor was a short demonstration of the code I have had prepared then (the bash scripts to automate the IP updating) and the Ansible playbooks mostly due to the nature of the project being technical.

This was discussed in the last meeting before the presentation on the 24th of March as is shown on the diary section above this.

Evidence of the technical demonstration of me running the cluster software can be seen in the commit logs of the repository that day. Specifically, after the commit with hash 9117594d002d3f3e99b1ad54371d33adc6ee0a8b that can be found in the following list of commits:

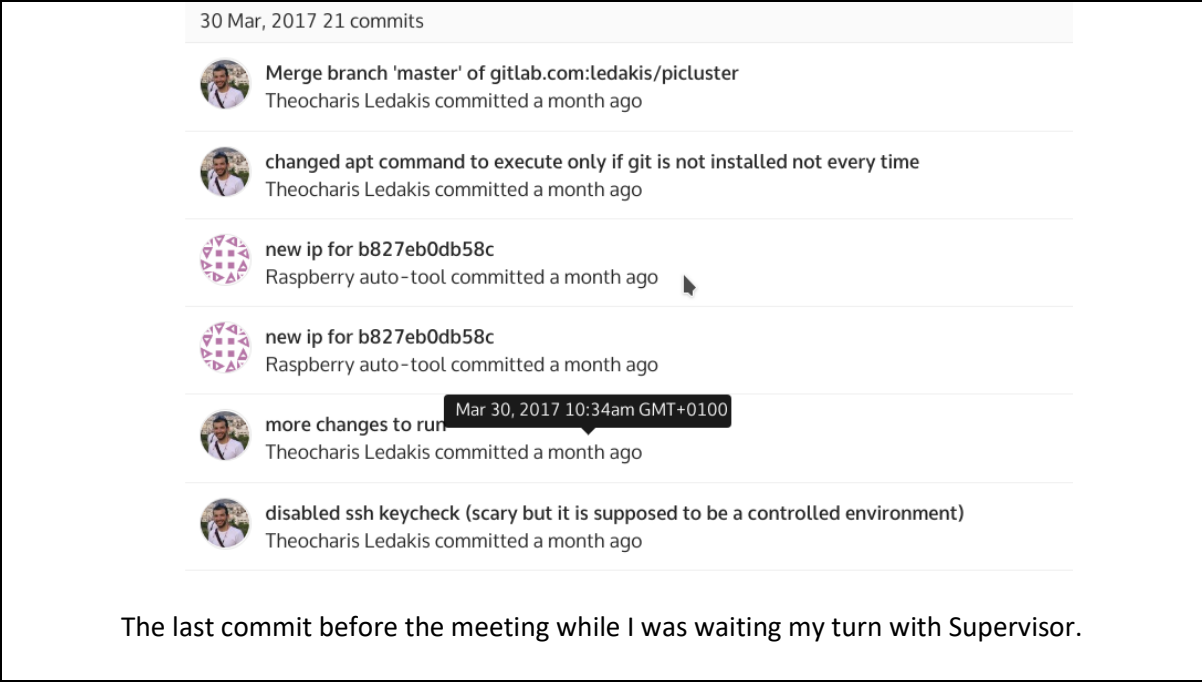
<https://gitlab.com/ledakis/picluster/commits/master>

(the individual commit can be found at:







<https://gitlab.com/ledakis/picluster/commit/9117594d002d3f3e99b1ad54371d33adc6ee0a8b>)

It can be seen, that the next commits, at the time of the presentation meeting, were made by my scripts as I was showing the supervisor the way the IP updating scripts work.

The following figures in addition to the URL I have provided above provide evidence for the presentation that took place on the 30th of March with me and the supervisor.



30 Mar, 2017 21 commits

-  Merge branch 'master' of gitlab.com:ledakis/picluster
Theocharis Ledakis committed a month ago
-  changed apt command to execute only if git is not installed not every time
Theocharis Ledakis committed a month ago
-  new ip for b827eb0db58c
Raspberry auto-tool committed a month ago
-  new ip for b827eb0db58c
Raspberry auto-tool committed a month ago
-  more changes to run
Theocharis Ledakis committed a month ago
-  disabled ssh keycheck (scary but it is supposed to be a controlled environment)
Theocharis Ledakis committed a month ago

Mar 30, 2017 10:34am GMT+0100

The last commit before the meeting while I was waiting my turn with Supervisor.

30 Mar, 2017 21 commits



Merge branch 'master' of gitlab.com:ledakis/picluster
Theocharis Ledakis committed a month ago



changed apt command to execute only if git is not installed not every time
Theocharis Ledakis committed a month ago



new ip for b827eb0db58c
Raspberry auto-tool committed a month ago



new ip for b827eb0db58c
Raspberry auto-tool committed a month ago

Mar 30, 2017 10:52am GMT+0100



more changes to run
Theocharis Ledakis committed a month ago

The next commit was by the automation tool that auto-committed when I demonstrated the IP updating scripts to my supervisor.

The feedback I received from my supervisor after our meeting is shown in the following figure.

presentation feedback



Carey Pridgeon

Thu 30/03, 11:20

Theocharis Ledakis 

interesting presentation, good work evidenced

good research backed work presented

Looks like this will result in a useful education product

Figure: feedback email received from supervisor after the presentation meeting

11.4 Handwritten notes/Diary

apt ~~update~~ upgrade.

get master from /boot

get config from /boot

clone repo

import keys + directions

start services and wait →

- pcluster service
- open mpi
- ganglia.

manage user → sudoers + ssh creds.

ssh enabled → custom port maybe? (get from conf)

Figure 1

2 solutions

1) Have my own image file with everything ready and just edit a config file through the python tool
(FASTER TO IMPLEMENT)
not elegant

2) The python tool injects the necessary files to the raspbian images ~~and~~ and the scripts run automatically on the startup process

Elegant solution, will take considerably more time to implement. → Also, future-proof? in regards to later-raspbian releases.

3) Compile "daemon" software for arm → ship it

Figure 2: Feedback I have written down while discussing with supervisor after the presentation regarding on whether to ship a Raspbian image or let the Python script do it.

So when master boots → it will kill all
openmpi processes on nodes.

~~maybe~~ also when it boots it will ~~remove all other~~
or clear the ip directory in order to ~~let other~~
~~nodes~~ keep it clean.

or there can be a flag telling it to do so
in /boot/master file.

Figure 3

Things
Variables to be careful of:

- 1) eth0 should be an option configured by the user or by the script automatically.
- 2) ssh option on boot partition to enable sshd → "ssh" file
- 3) What happens when the script asks the root password in order to write to the init folders/cron.d?
- 4) I need root access anyway if I am writing with dd !!
- 5) Emails of admins to send to.

How is the initial setup going to take place?

- git repository address.
- designate the master node somehow
(in git repo → /master file will have the mac address of the designated master)
The first pi will make itself a master unless the "master" file exists.

Clue: the initialisation tool can create a proper file that will contain everything.

Possible solution: rc.local file starting the daemon to do everything.

→ Cronjob not ideal a daemon better

Figure 4

Task list

- Ansible: - install core stuff (ssh keys, config node, etc) ✓
- install openmpi, openmpi interface ✓
- add users
↓
- bash: - ip → git ✓
- git cron ✓
- master file discovery ✓
- ips dir → inventory gen ✓
- 1st time init.d initialisation. ✓ (rc.local)
- Python tool: - curses interface X
- python packaging into 1 executable X
- all dialogs implementation. ✓
- docs: - documentation about: → ansible playbooks
 - → how ip's are created ✓
 - how the inventory is created. ✓
 - how the user should ~~use~~ configure things.

Figure 5

- Terminal interface that initialises options for the sd images. This will burn the images to the sd cards as well or will just pack everything into a .img file.
- Needs to be easy to install, eg run a python file after installing the requirements
or, I could just pack the whole thing into a single zip that contains everything needed. Even ansible!
- The script could just inject the necessary files in a raspbian imaged sd card directly.
- It can recognise the mounted cards and ask the user where to add the necessary files.
The first time it runs it can run from init.d and then insert the necessary scripts into cron.

Figure 6

What the system needs to do (Deliverables):

- be network agnostic. the user does not know the particulars of the network and does not care. ~~the~~ the cluster will use ~~the~~ the standard network system on the Raspbian to connect (get an ip from dhcp)
 - after imaging the SD card, ~~do~~ do no other work to configure the network (apart from physically setting up the nodes obviously)
 - the system will start configuring using the designated git repo and ssh keys ~~and the~~
 - the user will be able to SSH to the master and run openMPI commands.
- + documentation.

Figure 7