Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Abschlussbericht

Raspberry Pi Cluster

vorgelegt von

Ansari, Sam; Schulte, Lukas

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Studiengang:	B. Sc. Informatik
Matrikelnummer, Name:	7260567, Sam Ansari
	7315353, Lukas Schulte
Betreuer:	Jannek Squar

Hamburg, 29. März 2023

Contents

1 Introduction 4

2 Assignment 5

3 The Raspberry Pi 6

4 Software selection 9

4.1 OS 9
4.2 Storage and file system 9
4.2.1 Lustre 10
4.3 Ansible 10
4.4 Spack 11
4.5 SLURM 12
4.6 Monitoring 13
4.6.1 Telegraf 13
4.6.2 InfluxDB 14
4.6.3 Grafana 14

5 Software installation 15

5.1 OS/Initialization 15
5.2 Ansible 15
5.3 Spack 16

6 Installation 17

6.1 Power 18
6.2 Cooling 18
6.3 Expansion 18
6.4 Networking 19

7 Performance testing 23

7.1 HPL 23
7.2 OSU micro benchmarks 24

8 Limitations 26

8.1 Remote management 26
8.2 Network Booting 26
8.3 SD Cards as boot medium 27

9 Conclusion 28

Bibliography 29

1 Introduction

When calculating mathematical problems, researchers nowadays usually use a digital computer. However, they are limited to the calculation capacity of that one computer. For small projects, it may not be a problem to let a digital computer run for several days. Nonetheless, there are calculations, for example for climate research, which would take decades to complete on a single computer. This is why such calculations are done on so-called supercomputers.

A supercomputer is not a single computer that is able to calculate faster; they are a cluster of many computers that are connected together by a network, making it possible to communicate at great speeds. But with the added speed potential comes a great amount of complexity, as the program is executed in parallel on multiple computers instead of running on one computer sequentially. The researcher has to write their software to parallelize correctly and efficiently communicate between computer nodes. Above all, the program should deliver reproducible results that would be the same as its sequential counterpart.

Because of this added complexity, researchers work together with mathematicians and software engineers. Mathematicians help with the correctness of the calculation, and software engineers are responsible for the reproducibility and optimization of the software to be able to run on a cluster of computer nodes efficiently. But before a software engineer is able to help a researcher with the optimization of their software, they first need to have the possibility to explore the options of parallelization and train their skill set without having to claim precious calculation time on a supercomputer. This is the reasoning behind training clusters that are, for example, comprised of 128 Raspberry Pis.

This report starts with the assignment and a description of the Raspberry Pi, after transitioning over to the software selection and installation, followed by the hardware installation. Lastly, performance tests are presented together with the limitations and conclusion.

2 Assignment

To gain experience with parallelizing software, the University of Hamburg offers courses, teaching how to transform a program that runs sequentially on one computer to run in parallel on multiple computers that exchange their calculated data with each other to work together to solve a problem quickly and with the same outcome as the sequential program.

These exercises are done on a variety of small training clusters. However, the size of these clusters is too small for the students to experience the point of diminishing returns, as the amount of speedup due to parallelization is limited to the number of sequential parts in a program. As communication between nodes is done sequentially, it is possible to have too many nodes for a calculation, making the process take longer as the nodes have to wait to communicate more often than if there were fewer nodes. The number of nodes and the speed benefits can be roughly described as a bell curve. The speed of the program increases with the number of nodes until a certain point where there are too many nodes and the speed steadily decreases.

The task was to introduce students to these concepts with the help of a cluster made up of 128 Raspberry Pis (4th Edition, 4GB RAM). It is not important for the cluster to be powerful, as it will be used for educational purposes and demonstrate the speed increase/decrease with different numbers of nodes, making it possible for students to further optimize their programs to include less and more efficient communication between nodes to achieve further speed increases before the point of diminishing returns is met.

This new cluster, called the Pi Cluster, has to be integrated into the existing infrastructure, dictating many software choices. In addition, it is mandatory to offer automation and tools with which operators can quickly administrate the cluster to keep the amount of additional labor for running the Pi Cluster to a minimum.

3 The Raspberry Pi

The Raspberry Pi is a small single-board computer created by the Raspberry Pi Foundation (see Image 3.1). It is designed as an inexpensive device for education and projects with a focus on modularity.

The decision was made in favor of the Raspberry Pi 4 B with 4GB of RAM, which is equipped with a Broadcom BCM2711 SoC comprised of four ARM A72 CPU cores, which use the ARM v8.2 Instruction Set Architecture (ISA) and is manufactured with a 28nm process achieving clock speeds of up to 1.5GHz. As shown in Figure 3.2 it comes with a large variety of I/O but with few if any high-speed interfaces. It is limited to Gigabit Ethernet and its only PCIe Lane is reserved for the USB 3.0 Ports.

At first glance, this decision appears to be not optimal when building a new cluster, as usually, performance and high-performance I/O are very important. 128 Raspberry Pis provide less performance than modern single-node systems (see Section 7.1) and are less power efficient. But choosing Raspberry Pis allows for much higher quantities of nodes for a low cost. As the Pi Cluster is for teaching purposes, the quantity of nodes is much more valuable than the performance of the cluster.

Moreover, just before the decision for the hardware of the new cluster was made the RIKEN Center for Computational Science deployed their Fugaku supercomputer [Fuj] that uses the Fujitsu A64FX ARM v8 processor and took the lead on the Top500 list [1], a ranked list of the most powerful supercomputers in the world and proving ARM-based architecture to be a viable option for high-scale scientific applications. Additionally, an ARM-based cluster is an interesting option because of the new and different libraries and compilers.

With the new Raspberry Pi 4 a completely new type of core was introduced to the Raspberry Pi family. Previous models used so-called *LITTLE cores* from ARM's *big.LITTLE* architecture. This kind of core is designed for low-power applications and lacks a lot of features, improving power efficiency. The A72 architecture is a fully *big core* architecture, designed for high-performance applications. The A64FX from Fugaku the supercomputer uses a slightly modified A76 architecture which is very similar to the A72 but four generations newer. Feature-wise, the A72 architecture from the Raspberry Pi is very similar to the A76 just a lot slower.

With these prerequisites the Raspberry Pis appeared to be the perfect choice for this type of cluster, providing both a platform for testing ARM v8 tools and libraries and allowing for up to 128 nodes to be within the budget.

In the course of this project other, similar single-board computers were released, for example, the Orange Pi [H.b] which comes with more I/O and higher performance cores.

[1]https://top500.org/

Figure 3.1: The Raspberry Pi 4 [H.a]

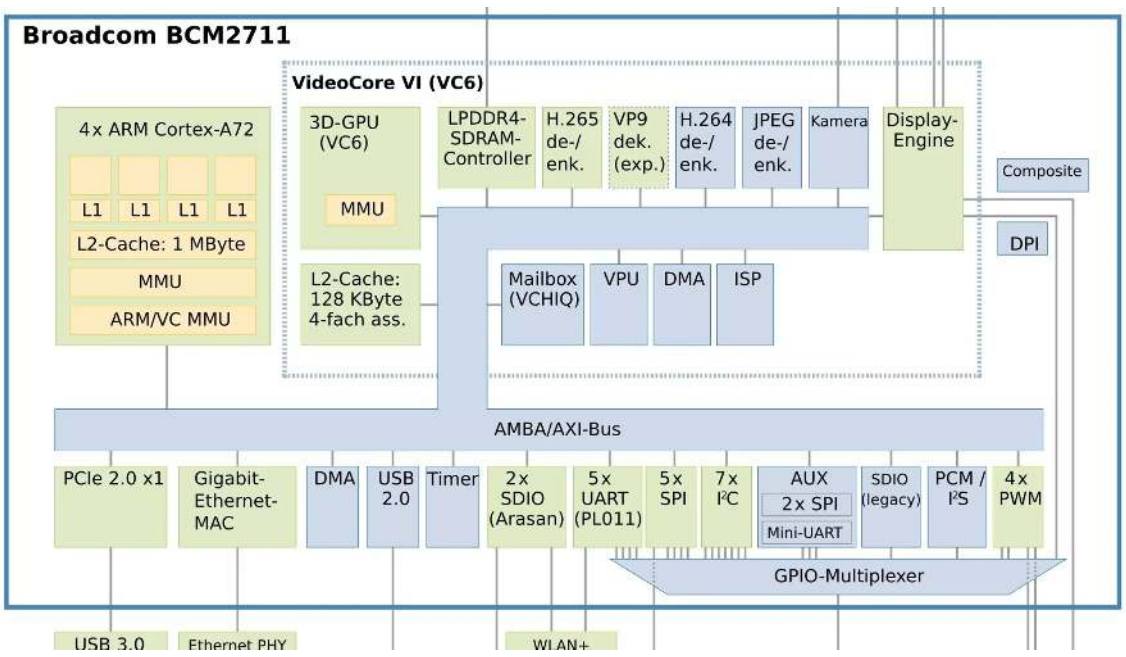


Figure 3.2: Blockchart of the I/O from the Raspberry Pi 4 [Win]

If this project would have started later it would probably be comprised of a different single-board computer with better I/O and performance. Notably, the native support of other storage media like regular M.2 would have been advantageous as explained in 8.3.

A Raspberry Pi differs in many ways from modern computers commonly found. First of all, it uses ARM architecture [Foub] which differs fundamentally from the widespread x86 architecture [Foud]. This means that programs and operating systems that are designed to run on the x86 architecture do not run on ARM architecture natively. But due to the widespread adoption of the ARM architecture, developers are slowly making it possible to deploy their software on ARM-based systems.

Another aspect commonly found on computers, missing from the Raspberry Pi is a comprehensive BIOS [Fouc] which are accessible by the user to make changes. The BIOS on the Pi is very rudimentary and always only loads a simple boot-loader from an integrated EEPROM [Foua]. Even though this EEPROM can be flashed with different software, there is no support for boot loaders like GRUB.

Lastly, the Raspberry Pi has general-purpose input/output (GPIO) pins that can be used for a variety of tasks. It is possible to connect sensors, buttons, and lights and integrate them into programs. This is the reason why the Raspberry Pi is very popular in the maker space.

4 Software selection

Before deploying a new cluster, the thought had to be put into the software that should be used to satisfy the needs of operators and users while offering features found on supercomputers for high-performance computing.

4.1 OS

For the operating system, we choose Ubuntu 20.04 LTS 64-bit. Due to the ARM v8 ISA, there were not many viable options for the operating system. Even though ARM's market share within the server market grows rapidly each year, fully ARM v8 compatible operating systems are still not very common. Additionally, as mentioned in Chapter 3 the boot procedure used in the Raspberry Pis is not compatible with typical boot loaders like GRUB, making many other operating systems incompatible. There are few operating systems with support for this platform. Ubuntu and Rasbian are the only ones with great support, the latter of which is maintained by the Raspberry foundation itself.

Since all the compute nodes within the already existing cluster were running on Ubuntu, Ubuntu 20.04 was a natural choice for this project. Moreover, at the beginning of the project, Ubuntu was the only 64-bit operating system available for the Raspberry Pi. Due to some initial issues building or rather compiling the Lustre file system, experiments were made on CentOS 7.9, Fedora 33, and Fedora 34. None of these solved the problems, but after some work, it then succeeded on Ubuntu. As parallel I/O was very important to the users, the Lustre Client was the most significant factor in this decision.

4.2 Storage and file system

The Raspberry Pis themselves are rather limited when it comes to I/O, which made the overall storage setup more complex as it should be offloaded from the internal SD card to preserve its write cycles and have a large capacity. Luckily, there was already an existing network file system infrastructure in place to solve the complexity.

The OS itself is installed on a simple SD card with a small capacity. Due to the limitations of this kind of storage media, explained in Section 8.3, our goal was to minimize the writing operation on the SD cards as much as possible. Therefore only the root file system is installed onto the SD cards, all other data is stored on other nodes on the network.

The home directories are mounted via an NFS share across the network and are managed globally with the help of the existing LDAP infrastructure. This made the setup rather easy, all that had to be done was integrate the nodes into the existing LDAP and mount the */home* directory.

All installed software is managed by the Spack package manager 4.4 which is installed onto a network share on another node. This means that all software is managed globally by the Spack installation. This network share is mounted on the */opt* directory and the paths are sourced on login.

Moreover, including a parallel file system was very important for the use case of the Pi Cluster. Parallel file systems are heavily used in HPC applications and are important in climate research because of the huge data sets that have to be dealt with. These parallel file systems are rather different from regular file systems allowing for multiple access to a single file from multiple processes on many different nodes. Additionally, the file system itself is typically distributed onto many nodes, increasing the overall throughput of such a system.

Given that there was an existing Lustre file server in the already existing cluster, the Lustre Client has to be installed on the Raspberry Pis to interact with the Lustre infrastructure.

4.2.1 Lustre

Lustre is a parallel, distributed, object-based, open-source file system, designed for high performance and scalability. These properties make it a good choice for large supercomputers, working with a lot of data. As measured by the Top500 [Uli] list, it is indeed used by the majority of the top 100 fastest supercomputers.

It uses a client-server architecture and provides file access to the Lustre clients through the network. In a Lustre system metadata and block, storage is separated onto three different classes of servers, object storage servers, which hold all the actual stored data, metadata servers, which manage index nodes (inodes), file system namespaces and a management server that provides all needed information about configuration and file system registries to the clients.

In this case, there was an existing Lustre infrastructure, so all that had to be done was install the Lustre Client onto the Raspberry Pis. This turned out to be a lot more difficult than it appears. First of all, there was no official support for ARM ISAs. Moreover, hardly any Linux distribution except for Redhat Enterprise Linux is officially supported by Lustre. This means that it was necessary to compile and install a file system client that was neither designed nor tested for the distribution nor the ISA. This meant that there was hardly any documentation available but after a lot of work, the Lustre Client could be automatically compiled and installed with the help of Ansible.

4.3 Ansible

One of the key aspects if not the single most important aspect of this project was automation. The existing infrastructure was installed and managed mostly manually. Some tools, like DHCP and PXE to automate at least some tasks were already used in the cluster but most of the work was done by hand. This worked quite well so far with only small clusters consisting of at most 10 nodes. With this new cluster, one hundred new nodes are added which is simply too much for a manual installation. So a solution was needed for managing and deploying the whole cluster.

Whatever solution had to meet some criteria to be a viable option. Most importantly it must be able to set up the whole configuration, including LDAP, software installation, MPI, and SLURM on different operating systems. Additionally, easy provisioning was an important aspect. Moreover, it has to support different systems and architectures as currently there are three different operating systems in use (Ubuntu 18, Ubuntu 20, and CentOS for the Lustre server) running on two different architectures in all clusters combined. Even though this is beyond the scope of this project, a solution that could be extended to manage all clusters including all existing nodes was preferred.

With these conditions, there are several options that would be able to provide this functionality, some of which are Ansible, Saltstack, and Puppet. The decision was made to use Ansible for a couple of reasons. There are no dependencies to install on the operating system. All that is required is a running sshd daemon and python, both are pre-installed on the base Ubuntu operating system. Then, its agent-less approach consumes no resources on the managed nodes. Even though this is not as scalable in terms of performance since all the instructions are managed by a single server, this is easily enough for this use case as the nodes won't be recreated on a regular basis and the time of deployment is not critical. Moreover, we both had great personal experiences with Ansible prior to this project making it a familiar tool for us to use.

Ansible is an open-source tool that implements infrastructure as code. It is written in Python and comes with no dependencies besides SSH and Python. There must be at least one Ansible deployment server in the network but theoretically, any node could take the role of deployment server because there are no agents necessary. All configurations and changes are initiated by the Ansible server. It generates the Python code, copies it to the remote machines via ssh, executes the code on the remote machines, and then does the cleanup afterward. The user specifies a so-called inventory of nodes on that a command or instruction should be executed on. Then Ansible can both execute a single command on the remote targets or use a Playbook written in YAML that contains multiple commands or functions. Playbooks are designed to be easily readable and configurable and come with many different built-in functions that provide the most important functionality like package management, mounts, network configuration, etc. Playbooks can be idempotent. This is great for when the Cluster is expanded as outlined in Section 6.3.

4.4 Spack

Spack is an open-source package management tool for supercomputers designed to be simple to use interface for managing many different versions of software.

In scientific applications building or compiling the actual software turns out to be a challenging task. These applications typically come with a long list of dependencies like MPI or BLAS and can be built with different build options and compilers, resulting in very different performance characteristics. Even though Ubuntu comes with its own package management tool (apt) it simply cannot provide all these features and multiple versions of a piece of software. Spack was designed for use in large data centers where many different users and machines share a common installation of various software packages with different versions.

Users choose a package and can then specify the version, compiler, all dependencies, and build options. Spack then automatically downloads all necessary packages, sets up compilers, and compiles the whole application. This approach makes it compatible with pretty much any architecture and operating system and even works in heterogeneous environments. Spack itself is nondestructive so it is possible to have multiple installations of the same package with different linked libraries or build options. The user can simply load a configuration and Spack sets up the environment and path so the desired package can simply be used without further adjustments in other software. Moreover, this allows the packages to be installed in any location and fully transparent to the user. If users need more packages they can simply add a local Spack installation to their home directory and build any package without administrative access. All these features make it a good choice for this heterogeneous cluster. The existing nodes will keep all their packages and the Raspberry Pis will get their own ARM v8 compatible binaries. Spack will automatically choose the right binaries for the given architecture.

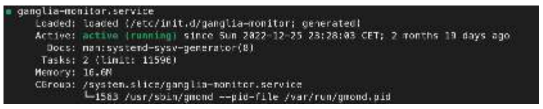
This also makes the deployment process a lot easier because new nodes only need to mount the network file system and then have access to all packages available on the cluster instead of having to install all of the necessary packages on each node individually.

4.5 SLURM

SLURM is as its name suggests a Simple Linux Utility for Resource Management. It is designed to be a scalable cluster management and job scheduling system for Linux-based clusters and is used in the majority of supercomputers in the Top500 list [Ihl]. The whole cluster is managed by a centralized *slurmctl daemon* that handles all requests. Additional nodes can be defined as control daemon to add redundancy but even a single node is easily capable of handling thousands of nodes and jobs simultaneously. All other nodes only need a SLURM daemon running which makes the whole cluster easily scalable. New nodes just need to be added to the control daemons node list and are immediately available for the users.

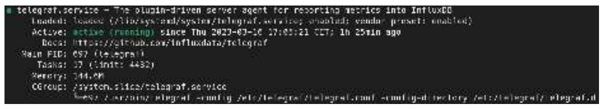
SLURM was already used in the existing cluster, so the Pi Cluster will use the existing control daemon. For testing purposes, the Pi Cluster also ran its own control daemon on one of the Pis which also worked without any issues.

Moreover, its topology-optimized resource selection functionality is of great advantage. As there are too many nodes to provide equal latencies between all nodes and have four sets of nodes that have lower latencies in node-to-node communication. As the testing in Section 7.2 shows there are significant differences in latency when we communicate beyond a single switch. The SLURM daemon can take these into consideration when



```
● ganglia-monitor.service
     Loaded: loaded (/etc/init.d/ganglia-monitor; generated)
     Active: active (running) since Sun 2022-12-25 23:28:03 CET; 2 months 19 days ago
       Docs: man:systemd-sysv-generator(8)
      Tasks: 2 (limit: 11596)
     Memory: 16.6M
     CGroup: /system.slice/ganglia-monitor.service
             └─1583 /usr/sbin/gmond --pid-file /var/run/gmond.pid
```

(a) Ganglia



```
● telegraf.service - The plugin-driven server agent for reporting metrics into InfluxDB
     Loaded: loaded (/lib/systemd/system/telegraf.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2023-03-16 17:03:21 CET; 1h 25min ago
       Docs: https://github.com/influxdata/telegraf
   Main PID: 657 (telegraf)
      Tasks: 17 (limit: 4432)
     Memory: 144.6M
     CGroup: /system.slice/telegraf.service
             └─657 /usr/bin/telegraf -config /etc/telegraf/telegraf.conf -config-directory /etc/telegraf/telegraf.d
```

(b) Telegraf

Figure 4.1: Memory usage of Ganglia and Telegraf as reported by systemd

making scheduling decisions. When possible the scheduler will prefer to allocate nodes with the lowest possible latencies and highest possible throughput.

4.6 Monitoring

A monitoring system is a very important part of every server infrastructure, it allows for identifying misbehaving nodes and can help find system bottlenecks. The existing cluster uses Ganglia as its monitoring system. The decision was made against using ganglia for a couple of reasons. First of all, Ganglia is very old and not actively developed anymore, the last release was in 2015 and therefore about eight years ago. In addition, there are very few metrics that can be monitored with Ganglia. Since the deployment of this cluster will require some modifications to the existing cluster, this was a good opportunity to try out something new that could be used for the entire cluster. The new monitoring system should consist of three main components, a data collection agent, a database, and a visualization tool using Telegraf, InfluxDB, and Grafana respectively. This combination of tools is very popular and widely used in IoT applications and especially on Raspberry Pis. Moreover, A recently published project report at the RTWH Aachen [Sch] compared different extensions for a Job specific monitoring for their SLURM cluster and use a monitoring system with the same components. Grafana as a visualization tool turned out to be the most flexible one.

4.6.1 Telegraf

Telegraf is a plugin-driven open-source data collection agent. It is written in the go programming language and does not have any external dependencies. It uses an agent that regularly applies all installed updates. Generally, there are four different types of plugins: input, process, aggregation, and output [inf]. The Pi Cluster mostly uses input plugins and a single output plugin. The output plugin is an interface for the InfluxDB database and simply sends all collected data to the database.

Input plugins collect any data. By default Telegraf already comes with hundreds of input plugins that can get data from different sources, be it system information like CPU, memory, services, or process data, sensor data like thermals or CPU frequencies or even third-party APIs. Each plugin launches a so-called task that executes a function or program. Since plain text in a JSON format can be used as a generic input plugin, it makes writing your own input plugins very easy and straightforward. For the Pi Cluster, we created our own plugins to monitor SLURM. The other two plugin types allow for data processing before it sends to an output. For the Pi Cluster, it was not necessary to process any data before it was sent.

Unfortunately, even though Telegraf is often used for IoT devices, mostly because of its great compatibility, it has a rather high usage of system resources, notably memory. As shown in Figure 4.6.1 the Telegraf agent used about 130MB to 160MB of RAM. In comparison, the gmod from Ganglia uses just about 16MB. Given that the individual nodes only come with 4GB of RAM this difference is significant.

4.6.2 InfluxDB

As a database to store the collected data, the Cluster uses InfluxDB by InfluxData just like Telegraf. It is a time series database which means that all data is accessed by a time stamp which is very useful for monitoring data. The main reason why InfluxDB is used instead of other databases is its ease of use and compatibility with Telegraf. The intended use case is for using it in a pipeline with Telegraf, InfluxDB, and Chronograph as a visualization tool. Therefore combining both Telegraf and InfluxDB is very easy and can be set up within minutes and works pretty well.

4.6.3 Grafana

For visualization, Chronograph which is the visualization tool from InfluxData was not chosen in favor of Grafana mostly for two reasons. First of all, because of personal experiences with the Grafana dashboard before this project. Second, Grafana has more Features than Chronograph and a lot more plugins developed by the community. Grafana can natively use the InfluxDB database and offers many visualization functions, provides a direct interface to the database, and can alert the user on certain behavior. This functionality might be useful for possible future projects and early warnings.

5 Software installation

Having our software selected for our Raspberry Pi Cluster we move on to actually installing it on the Raspberry Pis. All software installation is handled by Ansible and Spack.

5.1 OS/Initialization

Installing the OS onto the Raspberry Pis isn't as straightforward as one might think. As explained in section 4.1 the base system is installed differently than usual. It can't boot any ISO images which are usually used for installing operating systems. Instead, the OS image is directly written onto the SD cards.

Since there is no support for PXE boot, this had to be done manually which can be a tedious task depending on the size of the cluster. Additionally, during the first SSH connection with the Raspberry Pi, a root password has to be manually entered.

5.2 Ansible

Ansible is at the heart of the operation, as it provides the means of configuring multiple hosts from a centralized point with the help of so-called Playbooks written in YAML. These Playbooks describe what should be changed on which system, enabling a great number of possibilities. As the Playbooks are written in YAML which is very easy for humans to read and comprehend, they also double as documentation. In addition, the Ansible deployment server also holds all of the relevant configuration files and changes for easy serviceability. To have a homogeneous environment, every change was made by Ansible on all nodes and never manually on any single node.

To deploy the Pi Cluster, Ansible did the following steps:

1. **Initialization:** set passwords, exchange SSH keys, set boot sequence, update system
2. **Set OS options:** set hostname, set time, set keyboard layout
3. **Optimize OS:** remove snapd, remove cloud_init, disable WiFi and Bluetooth
4. Install Lustre Client
5. **Integration into Infrastructure:** install LDAP and connect to existing LDAP, configure */etc/fstab* to mount NFS and Lustre mounts for home directories, Spack and Lustre

6. Install and configure SLURM
7. Install and configure Telegraf
8. Install GCC

5.3 Spack

Spack can easily be installed by cloning the GitHub repository to some directory. In this case, the existing NFS share and the existing Spack installation were used to add all the necessary software. Now it was only necessary to compile the software on a Raspberry Pi to be able to run on the ARM architecture. In this case, the design of Spack makes this process very easy since Spack simply creates a new directory for the compiled binaries for Arm v8. After the build phase, all other nodes simply source this Spack installation and automatically choose the build for their architecture.

In theory, it is possible to use the slurm environment to distribute the compiling process which would improve the total installation time. Since we aren't using a lot of packages of which most have many common dependencies a distributed build process wouldn't improve the installation time by a lot. When building only the necessary software for the Pi Cluster needed by our users a single node could build the software in about 90 minutes. Spack also makes it possible to build additional software later.

6 Installation

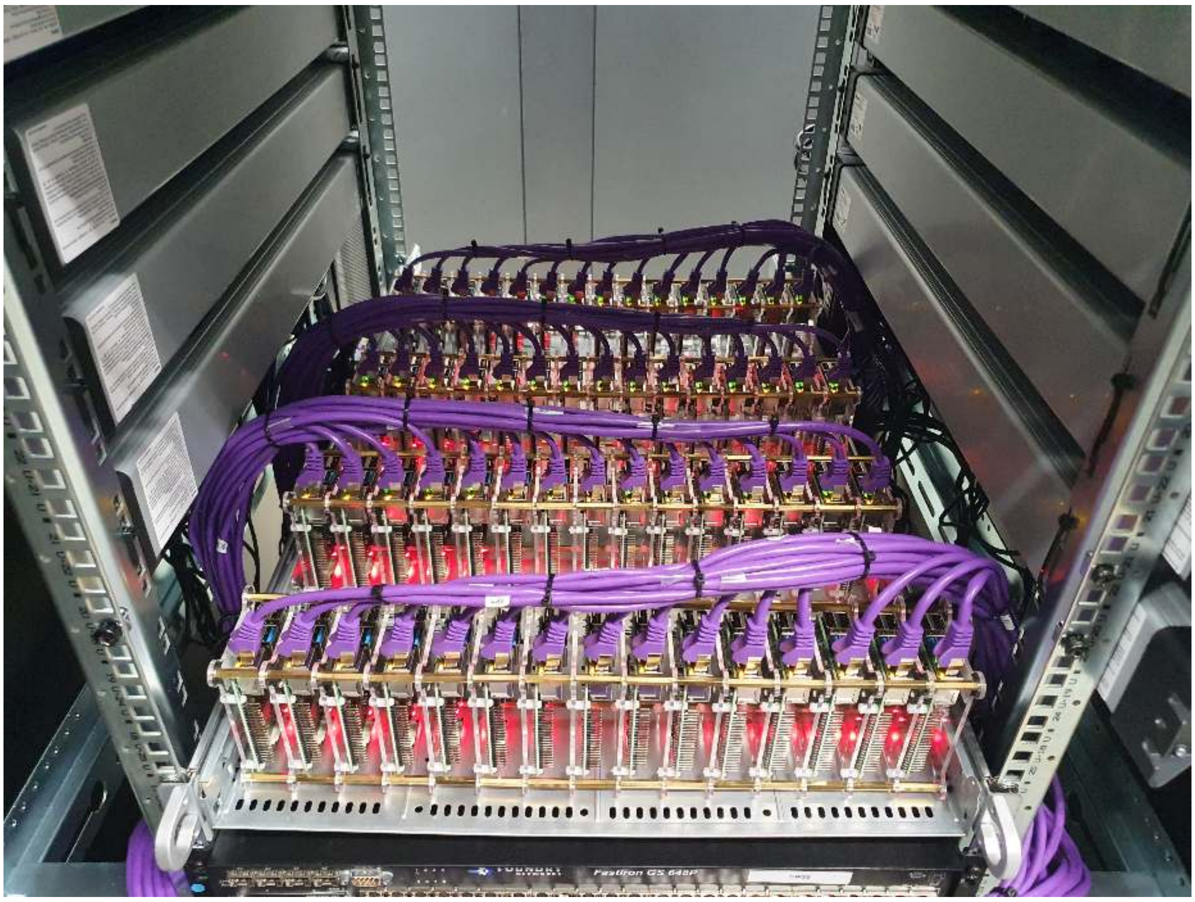


Figure 6.1: One layer of installed Raspberry Pis

As shown in Figure 6.1 acrylic tower kits were used to install the Raspberry Pis. The towers were loaded with 15 Raspberry Pis and laid onto their side for easier cable management and more efficient use of height. They rest on old defective SuperMicro 1U Servers which were completely stripped down yielding a rack-mountable flat surface. With this solution, the Pis have a good amount of airflow due to the open design of the acrylic towers spacing out the Pis. Lastly, this was the most economic solution for mounting many Pis especially compared to dedicated rack mount cases designed for Raspberry Pis additionally using old server components that are repurposed and put back into use. In total there are two trays full of computers.

6.1 Power

There were multiple options to power the Raspberry Pis. One option would have been to use power supply units with multiple or single USB ports and USB cables. Another option would have been to power the Raspberry Pis over their GPIO pins instead of using the USB connector. Additionally, it would have also been possible to power the Pis with Power over Ethernet which is only possible with an additional PoE adapter and a PoE switch which if it fails would also de-energize all of the Pis connected to it. As there was uncertainty as to how much power a Pi would require in this configuration the decision was made to buy the original power supplies. The benefit of using one power supply for every computer is that there is no single point of failure as is the case with using PoE with a PoE switch or power supplies with multiple USB ports.

As the size of the power supplies is large, special power strips were bought with big gaps between the sockets. The sockets are also oriented 90° instead of 45° which is common on power strips in Europe. As visible in Figure 6.1 the power strips are mounted with cable ties onto the sides of the Racks for neat cable management and also serviceability. In addition, it does not block rackmount equipment from being installed.

6.2 Cooling

There were serious concerns about cooling down the Raspberry Pis as they would overheat during testing. Before installing heatsinks or fans powered by the GPIO pins the decision was made to first install the Raspberry Pis in the data center. Fortunately, the spacing between the Pis due to the acrylic towers and the positive air pressure in front of the rack due to the cold aisle forcing the air through the towers and out through the back is enough to keep the Pis cool.

To optimize the airflow and the air pressure rack panels were installed as seen in Figure 6.2. In the future, it would also be possible to install a server above the trays with Raspberry Pis to more directly channel the air towards the back and to achieve a higher air pressure.

6.3 Expansion

During the installation, we noticed that the official power supplies were too large to all fit into the original power strips which were bought for this project. However, instead of waiting for the new solution to power all Pis (see Section 6.1) the decision was made to power up a portion of the Raspberry Pis and split up the integration into two steps. Because of the use of Ansible, it was easy to integrate the second half of the Computers by simply adjusting configuration files and running the Ansible Playbook on all Computers. Installing the missing software on the newly powered-on computers and updating the configuration on the Raspberry Pis which were already powered and already part of a cluster.

This two-step integration showed the great value of automating all processes and that adding additional computers to the cluster is effortless.

6.4 Networking

With a large number of clients networking all clients together is no trivial task. It is important to not overload a switch and also have short delays between clients with as few hops as possible. To tie all Raspberry Pis together two existing 48-Port "Brocade FastIron Edge GS 648P - POE" and two "Foundry Networks FastIron GS648P" switches were used. As the Raspberry Pis have a gigabit connection the switches also had to have gigabit connections to utilize the full potential of the network card in the Raspberry Pis.

Figure 6.3 depicts the network diagram and shows how the clients are wired into the network. Switches 01 and 02 are responsible for the bottom tray of Raspberry Pis and switches 03 and 04 for the top tray and each except switch 04 is connected to two rows of Raspberry Pis not using every port of the switches to balance the load over the switches. There are empty ports between the rows for easier maintenance as the rows are represented by the blocks of connected ports on the switches. Each switch has a connection to the backbone switch for communication with the rest of the infrastructure.

To increase performance and decrease the load on the backbone switch, connections between the switches are made and are shown in blue in Figure 6.3. In addition, it decreases a hop that a client needs if it wants to talk to a client in certain cases. For instance, if client 001 is connected to switch 01 and wants to talk to client 061 which is connected to switch 03 only two hops are necessary for them to communicate. If the connection between switches would not exist switches 01 and 03 would have to communicate through the backbone switch creating overhead on the backbone switch and adding a hop resulting in additional delays. However, the connections between the switches will be implemented in the future, as it is essential for the spanning tree protocol to be enabled, as these connections build a loop and would disturb the network greatly. As this feature is not enabled on all switches in the network, it will have to wait for the next maintenance cycle where the entire network is taken offline and not in use for it to be reconfigured.

To furthermore increase performance it would have been possible to run the switches in a stack and use 10-gigabit per second instead of gigabit between the switches but unfortunately, as these were existing switches that were not bought new, the cables needed to run the switches in a stacked configuration with a 10-gigabit uplink were missing. Given that the cluster is used for trial purposes and is much more intended to simulate the troubles of running an application on a high-performance cluster the limited uplink is not a great problem and much more a limitation with the potential for optimizing not only the amount of communication between nodes but also the size of the messages needed to be exchanged.

To make replacing a Pi more easy and the network more serviceable, the decision was made to not use static IP addresses but to use DHCP with address reservations. With this decision, it is now possible to change the IP addresses of the entire Pi Cluster by changing the configuration file on the DHCP server instead of having to change it on every node.

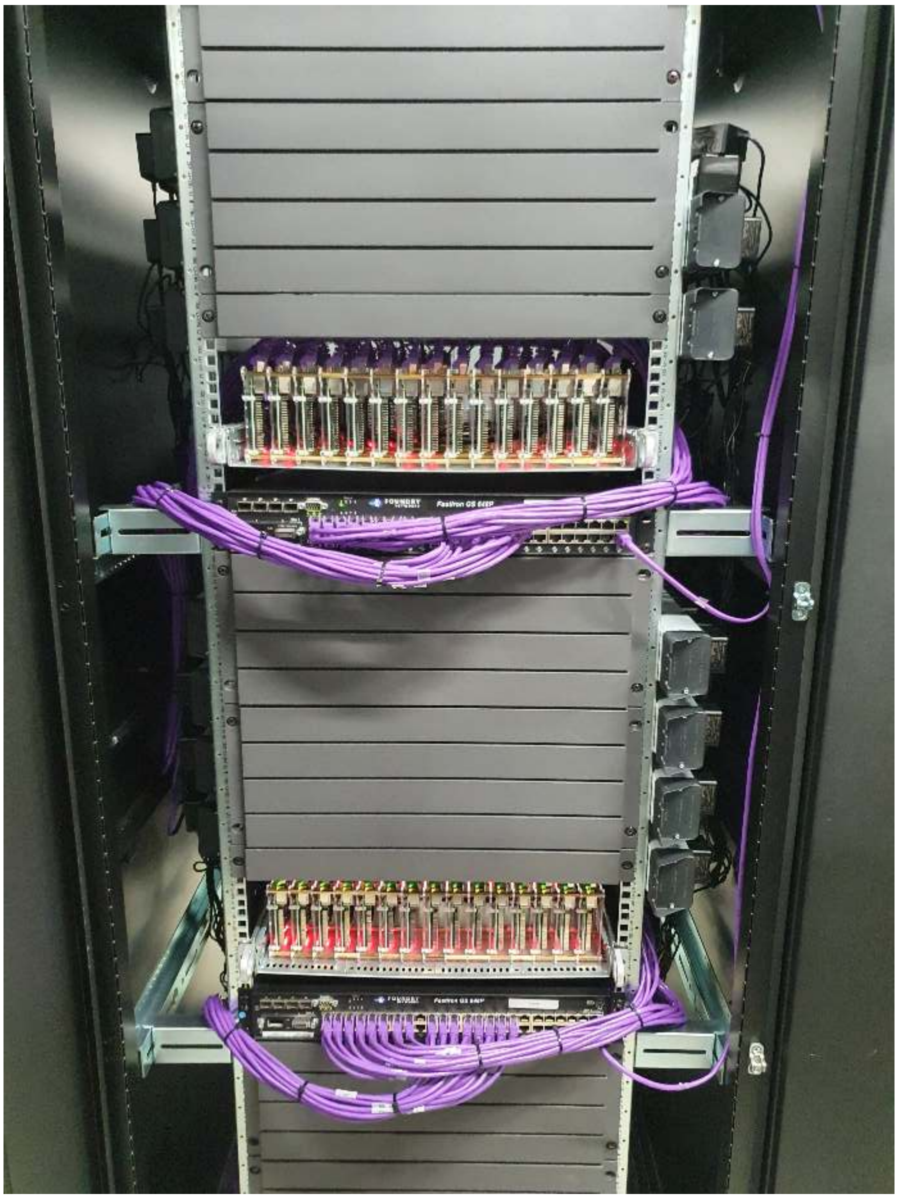


Figure 6.2: Both layers of installed Raspberry Pis and rack panels

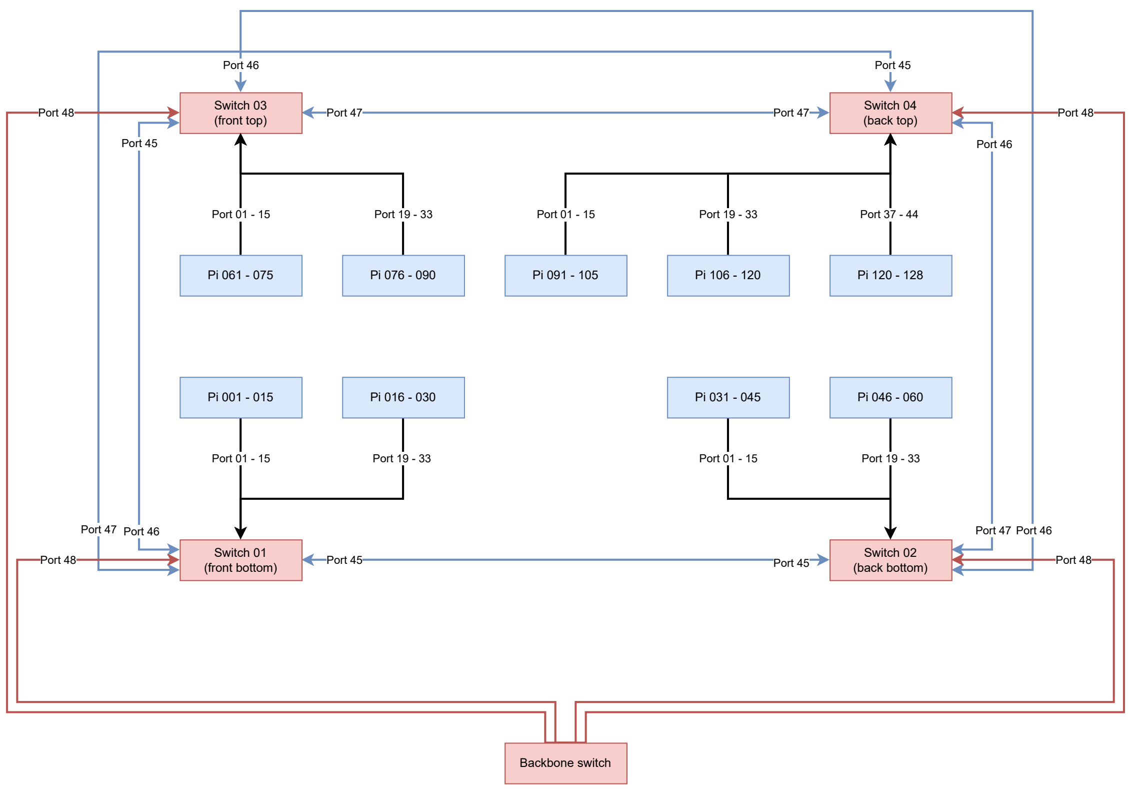


Figure 6.3: Network diagram for the Raspberry Pi Cluster

7 Performance testing

To evaluate the performance and stability of the cluster benchmarks were run on the Pi Cluster. The tests are limited to smaller, so-called micro benchmarks.

7.1 HPL

HPL is a software package that solves dense linear systems in 64-bit precision[1]. The Raspberry Pi 4 uses 4 A72 cores that are running at 1500MHz. Each A72 core has a single 128-bit Neon vector pipeline split into two 64-bit pipelines. With its FMA support, it can perform up to four 64-bit floating point operations in every clock cycle. At 1.5 GHz the theoretical performance is $4\frac{flop}{cycle} \cdot 1 * 10^6 \frac{cycles}{s} = 6 * 10^6 \frac{flop}{s} = 6$ GLOPS per core [Goo]. With 128 Pis and 512 total cores. At 100% efficiency, this would result in 3072 GFLOPS.

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR11C2R4      221952   192     4    31            8096.80             9.0028e+02
HPL_pdgesv() start time Wed Apr 13 15:23:37 2022

HPL_pdgesv() end time   Wed Apr 13 17:38:34 2022

--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=   1.42932391e-03 ...... PASSED
================================================================================
```

The best result was 900 GFLOPs, barely missing the teraflop mark. This means an efficiency of just 29.3%. There could be several reasons for this poor result. First of all, despite all efforts, there might be better configurations of the input file and BLAS Library. OpenBLAS was only used in the tests and the input variables were not tuned all. A better-optimized BLAS library might yield better results.

But the hardware might be an issue too. In the tests, the temperatures were rather low when running the HPL benchmark. The power consumption was very constant throughout the runs which leads to the belief that the Raspberry Pis might be limited by their power limits. This could maybe be resolved with a firmware update.

[1]See netlib.org as reference

Moreover, the network configuration isn't optimal either. When testing a single node it achieved 12 GFLOPs indicating very poor scaling across this high number of nodes.

Compared to other CPUs, this result is pretty weak. A single Intel Xeon 6130, a $1900 16 core CPU from 2017 [Int] easily outperforms the whole cluster by a huge margin with over 1.7 TFLOPs [SP]. The power consumption of individual Raspberry Pis was around 6W. With 128 Raspberry Pis that would be around 768W which makes them not only perform much worse than a single socket system but also more expensive per TFLOP and less efficient.

7.2 OSU micro benchmarks

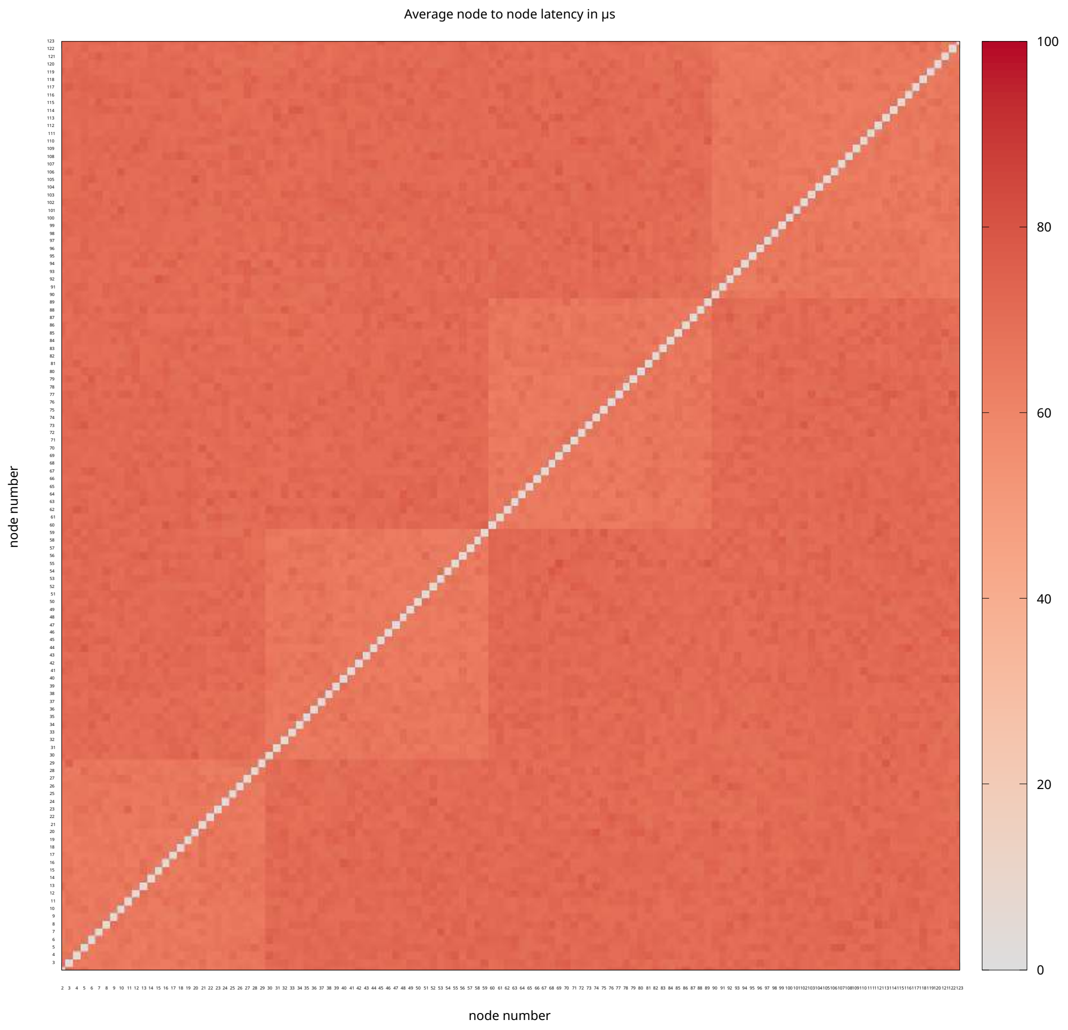


Figure 7.1: Pairwise latency between all nodes in µs

The OSU micro-benchmarks are a collection of numerous MPI-specific benchmarks by the NBCL[2]. Many different Benchmarks for testing the connections were used here. But the most interesting results were produced by the point-to-point latency. Figure 7.2 shows a matrix of the latencies between each node. As discussed in 6.4 a nested topology of switches is used where all the nodes are evenly distributed among all switches which are then all connected to one single switch. This is not optimal as it puts a lot of load onto the backbone switch and adds latency. The weak scaling of the HPL benchmark 7.1 might be caused by this. In the future, a ring topology of the available switches will be implemented, connecting the switches to each other. The advantage of the ring topology is the lower latencies to more distant nodes. The benchmark clearly shows that the number of hops between the nodes has a significant impact on the latencies. It even shows which nodes are connected to which switch by their latency.

[2]https://mvapich.cse.ohio-state.edu/benchmarks/

8 Limitations

Raspberry Pis are very limited when it comes to High-Performance Computing and server applications both in terms of performance and features. Given that performance is not a high priority for this use case other limitations are much more relevant for the Pi Cluster.

8.1 Remote management

Unfortunately, the Raspberry Pis do not have any remote management capabilities. That means that whenever access beyond the operating system is needed, e.g. to force a reset of a node, physical access to the nodes is needed. To aid in serviceability the nodes and marked so one could at least identify the right node within the rack quickly.

Moreover, the whole cluster is designed to fail softly, meaning that the Cluster can lose any amount of nodes without any effect on the other nodes. Since high availability is not necessary in this case, it is enough to provide an overall stable cluster with no redundancy or power backup. However, adding hardware for remote access might be an interesting future project.

Another idea would be to add network-controlled power outputs that can be turned off to reboot not responding nodes.

8.2 Network Booting

All the other servers in the existing network use a PXE which provides their boot image. This makes the installation of the operating system much easier. Unfortunately, the Raspberry Pis do not support this standard, which means that the base operating system must be flashed manually onto the SD cards and the root password set. This causes a lot of manual work whenever a large amount of nodes is added or the base image is updated. Newer revisions of the Raspberry Pis do support some form of network booting, TFTP boot to be exact. This differs quite a lot from PXE boot and requires accompanying infrastructure. When using TFTP boot the node would mount the root file system through the network at boot time. That means the entire root file system would be stored on different machines. But at the current state, the existing infrastructure would not have enough resources to provide all 128 nodes.

8.3 SD Cards as boot medium

Raspberry Pis usually use SD cards as their boot medium. Even though newer revisions do support other boot media, this would not be really applicable to the Pi Cluster. Therefore SD cards have to be used for the operating system which comes with many downsides.

First of all the performance of SD cards is rather low but besides the deployment phase that does not impact the performance because all other data is stored on storage nodes on the network. The reliability of SD cards is a greater concern. SD cards are generally known to be less reliable than other storage media like HDDs or SSDs. Moreover, there is no option for any redundancy.

Should the reliability of the SD cards be very poor, other solutions like USB to M.2 adapters with an SSD could be viable solutions.

9 Conclusion

Over the course of this project, we were able to deploy a whole new cluster based on Raspberry Pis and integrate it into the already existing cluster. Moreover, introducing a new set of monitoring tools and Ansible as an automation tool for the whole infrastructure. The introduction of these new tools provides new possibilities for the operators of the clusters. The addition of automating every step is valuable as new hardware can be easily deployed and the scripts act as documentation for all of the changes made to a node and its configuration. The more in-depth monitoring provides operators with much more data making it also possible to act proactively. These monitoring tools can also be used by the user to gauge a better understanding of parallelization and how to fully utilize all of the performance available.

Despite the poor performance and efficiency both in terms of speed and power, this cluster is a great addition to the existing cluster. The Pi Cluster accomplished the goal of being a serviceable and cost-effective cluster for simulating communication between many nodes. It would have been nice to have extra performance with the cluster but it was never the main consideration. Lastly, besides all the shortcomings of this architecture, the cluster works well and turned out to be reliable over the last year!

It will be interesting to further optimize the performance of the cluster by increasing the network connections between switches and also further optimizations of the operating system. All in all, we are very happy to have been part of such a unique project!

Bibliography

[Foua] Raspberry Pi Foundation. Raspberry pi boot sequence. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#boot-sequence. Accessed on 2023-03-18.

[Foub] Wikimedia Foundation. Arm architecture family. https://en.wikipedia.org/wiki/ARM_architecture_family. Accessed on 2022-07-12.

[Fouc] Wikimedia Foundation. Bios. https://en.wikipedia.org/wiki/X86. Accessed on 2022-07-12.

[Foud] Wikimedia Foundation. x86. https://en.wikipedia.org/wiki/X86. Accessed on 2022-07-12.

[Fuj] Fujitsu. Special feature: Supercomputer fugaku. https://www.fujitsu.com/global/about/resources/publications/technicalreview/2020-03/. Accessed on 2023-03-18.

[Goo] Chris Goodyer. Cortex-a72 maximum theoretical linpack performance r_peak. https://community.arm.com/support-forums/f/high-performance-computing-forum/47220/cortex-a72-maximum-theoretical-linpack-performance-r_peak. Accessed on 2023-03-18.

[H.a] Michael H. Raspberry pi 4 model b. https://de.wikipedia.org/wiki/Raspberry_Pi#/media/Datei:Raspberry_Pi_4_Model_B_-_Side.jpg. Accessed on 2023-03-18.

[H.b] Michael H. Raspberry pi 4 model b. http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-5.html. Accessed on 2023-03-18.

[Ihl] Nick Ihli. Accelerating hpc and ai with slurm and schedmd, page 3. https://slurm.schedmd.com/SC22/Accelerating-with-Slurm.pdf. Accessed on 2023-03-18.

[inf] influxdata. Telegraf. https://www.influxdata.com/time-series-platform/telegraf/. Accessed on 2023-03-18.

[Int] Intel. Intel® xeon® gold 6130 processor. https://ark.intel.com/content/www/us/en/ark/products/120492/

intel-xeon-gold-6130-processor-22m-cache-2-10-ghz.html. Accessed on 2023-03-18.

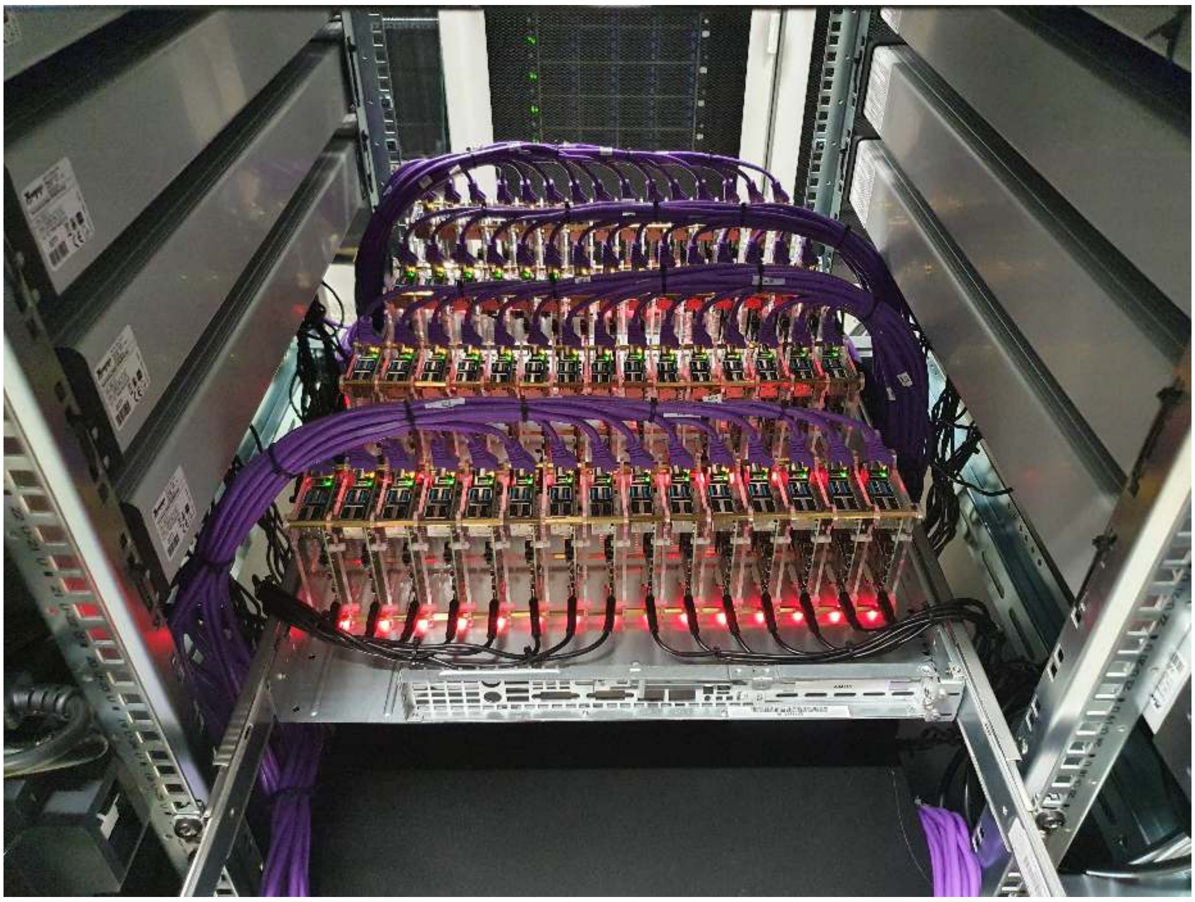
[Sch] Severin Schüller. Job specific performance monitoring on the hpc cluster of the rwth aachen university. https://www.matse.itc.rwth-aachen.de/dienste/public/show_document.php?id=20066. Accessed on 2023-03-18.

[SP] Ashish K Singh Savitha Pareek, Varun Bawa. Hpc synthetic benchmark performance using 2nd generation intel® xeon® scalable processors – stream, hpl and hpcg. https://www.dell.com/support/kbdoc/de-de/000133009/hpc-synthetic-benchmark-performance-using-2-generation-intel-xeon-scalable-p Accessed on 2023-03-18.

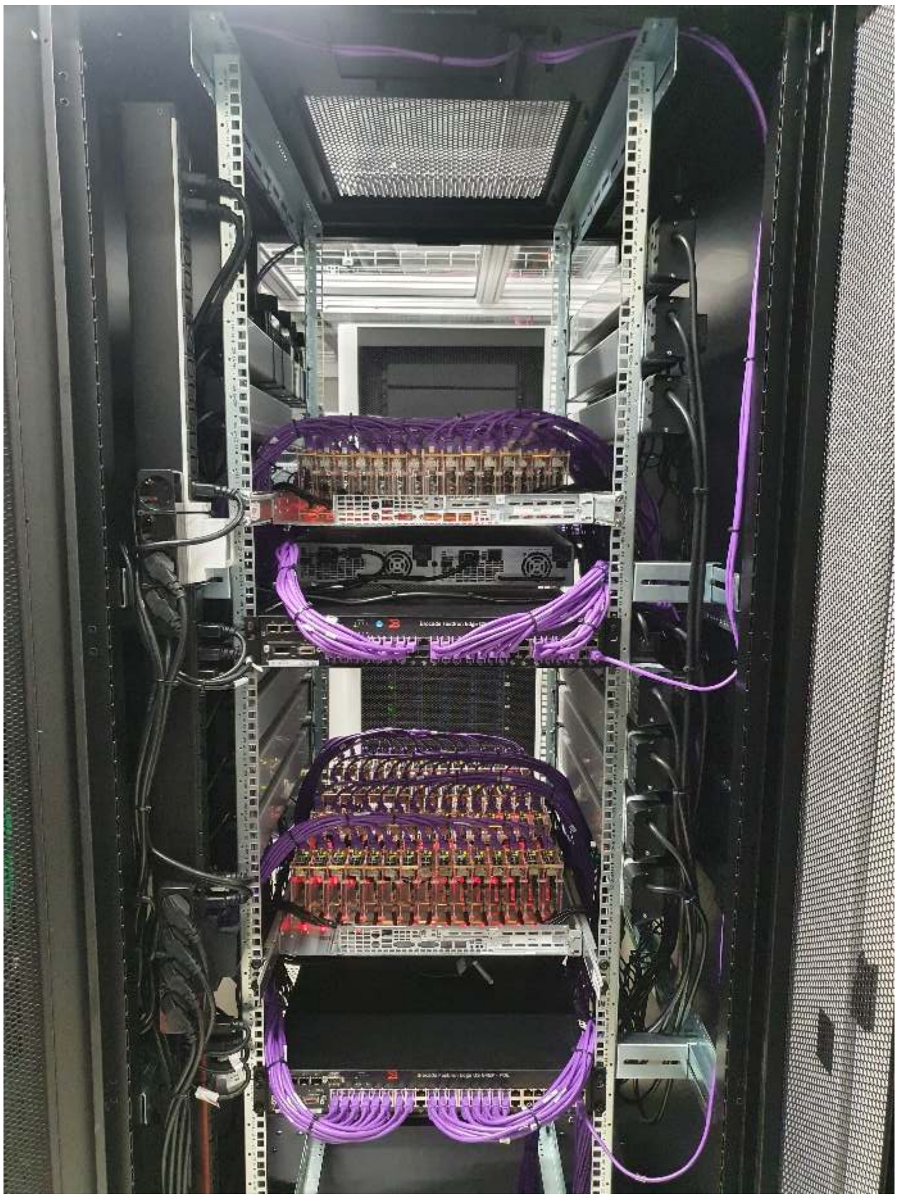
[Uli] UliPlechschmidt. It's lonely at the top: Lustre continues to dominate top 100 fastest supercomputers. https://community.hpe.com/t5/servers-systems-the-right/it-s-lonely-at-the-top-lustre-continues-to-dominate-top-100/ba-p/7109668. Accessed on 2023-03-18.

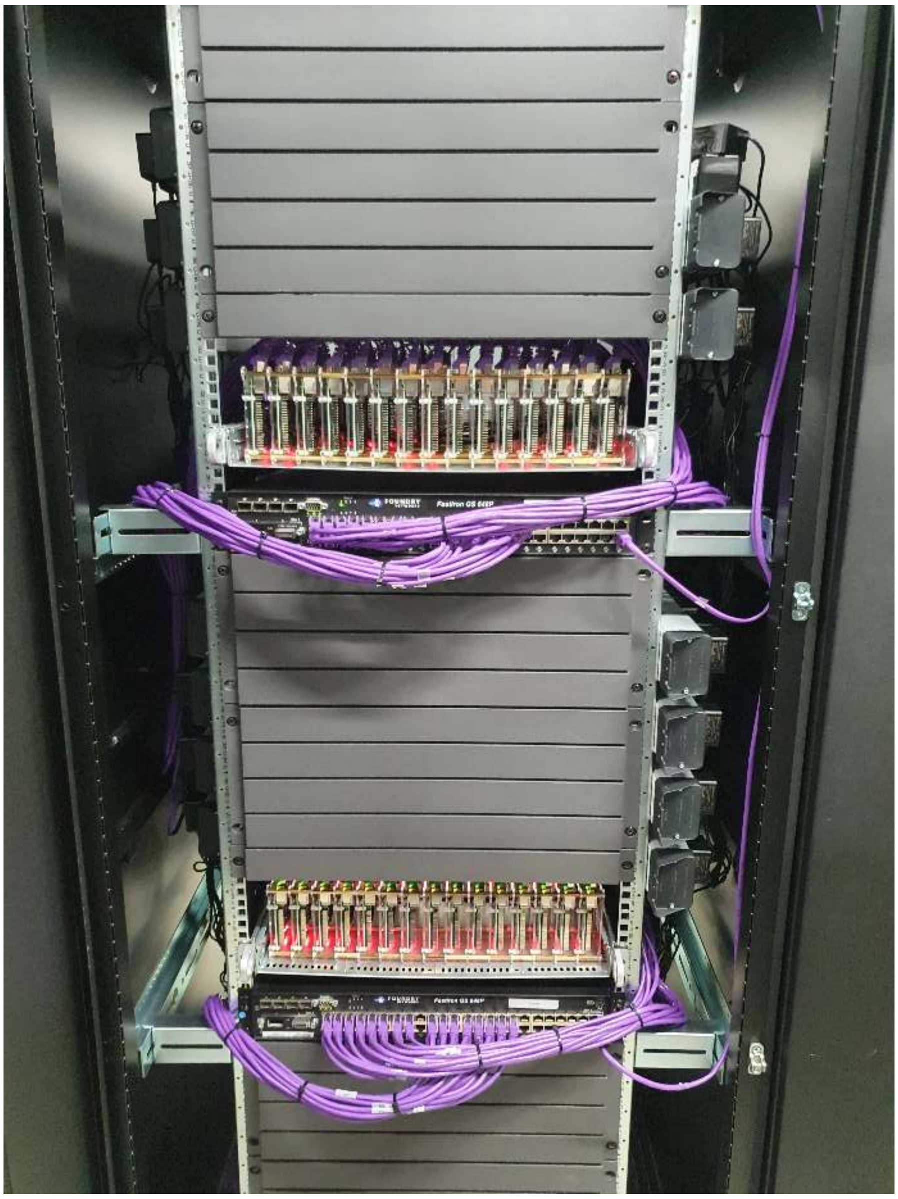
[Win] Christof Windeck. Raspberry pi 4 model b: Blockschaltbild des broadcom bcm2711. https://www.heise.de/hintergrund/Raspberry-Pi-4-Model-B-Blockschaltbild-des-Broadcom-BCM2711-4514399.html. Accessed on 2023-03-18.

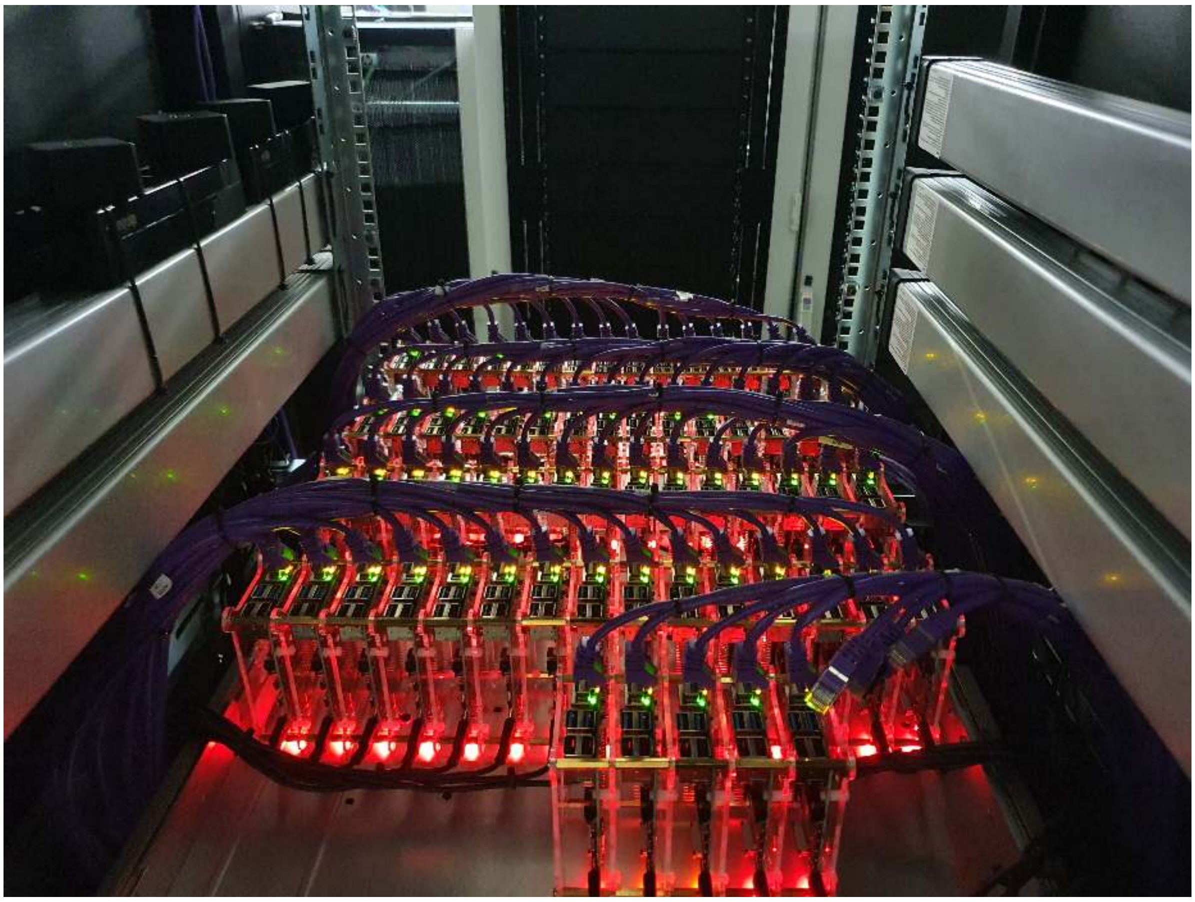
Pictures

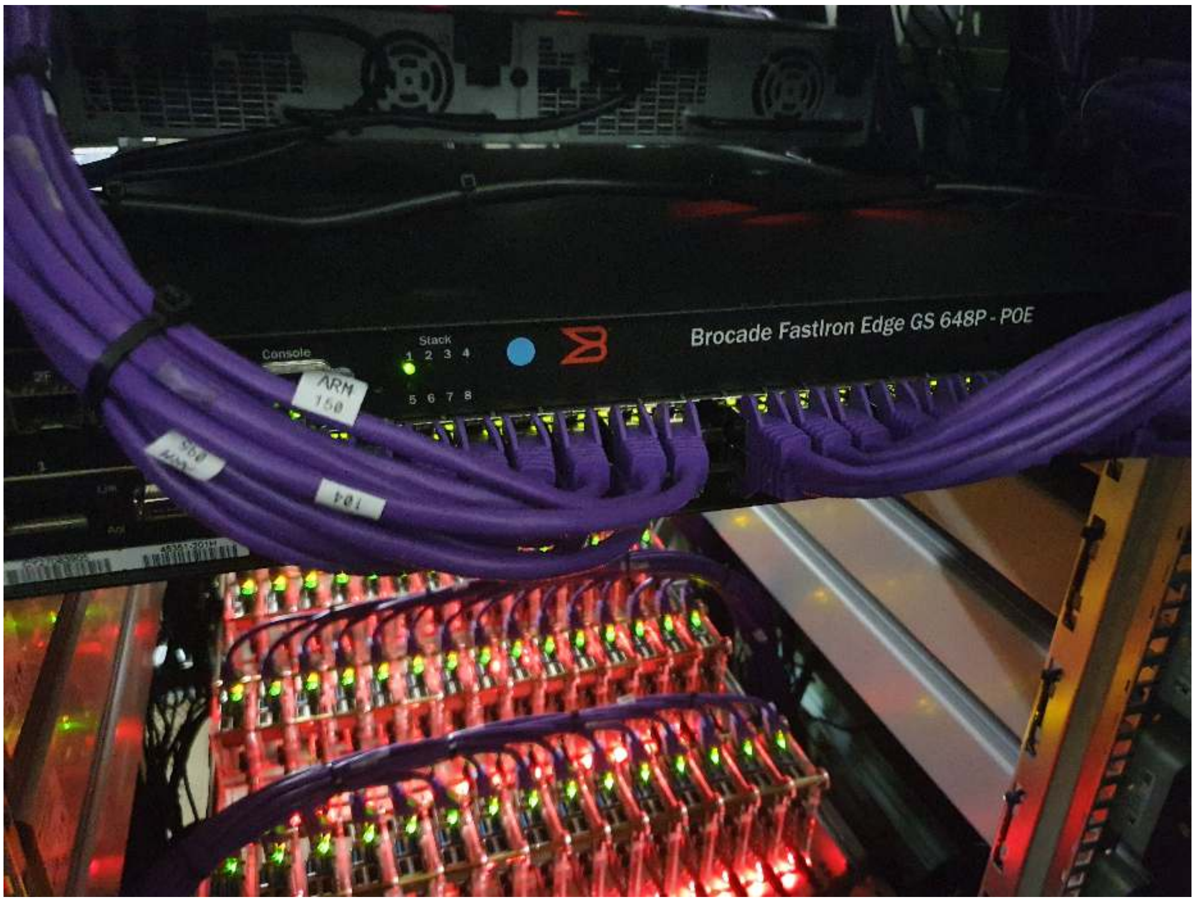


FOUNDRY NETWORKS
FastIron GS 648P




FOUNDRY
FastIron GS 648P




Brocade FastIron Edge GS 648P - POE
Console
Stack
1 2 3 4
5 6 7 8
ARM
150
104