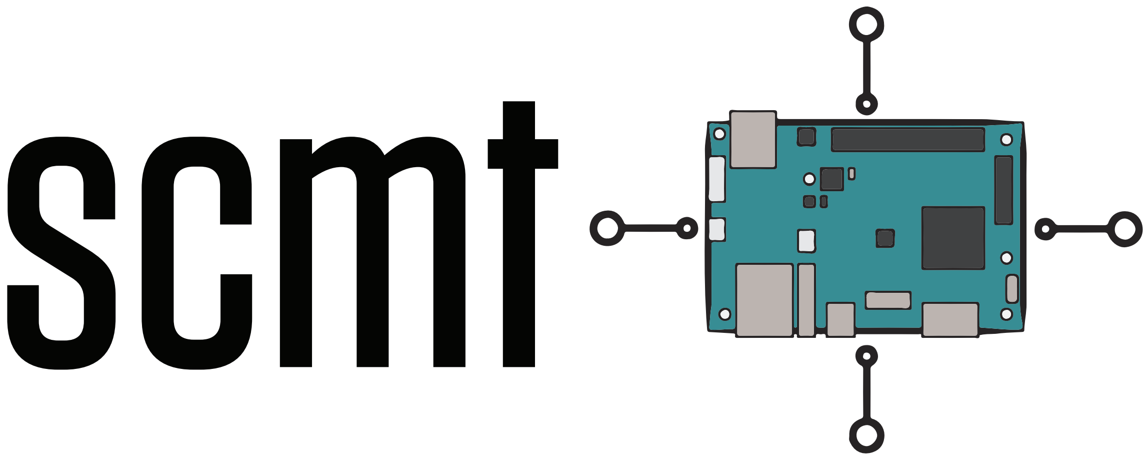
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



SuperK Cluster Management Toolkit

Plug-and-play management for single-board computer clusters

Bachelor of Science Thesis in Computer Science and Engineering

Magnus Åkerstedt Bergsten
Anders Bolin
Eric Borgsten
Elvira Jonsson
Sebastian Lund
Axel Olsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

SuperK Cluster Management Toolkit
Plug-and-play management for single-board computer clusters

Magnus Åkerstedt Bergsten
Anders Bolin
Eric Borgsten
Elvira Jonsson
Sebastian Lund
Axel Olsson

Examiner: Arne Linde

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: Logotype for our prototype software, SuperK Cluster Management Toolkit. Created by Magnus Åkerstedt Bergsten and Sebastian Lund.

Department of Computer Science and Engineering
Göteborg, Sweden June 2016

Abstract

Computer clusters have become increasingly important in fields such as high-performance computing and database management due to their scalability and lower economic cost compared with traditional supercomputers. However, computer clusters consisting of x86-based servers are expensive, highly power-consuming, and occupy significant space.

At the same time, the availability of cheap but performant ARM-based single-board computers has steadily increased, with products such as *Raspberry Pi* and *Odroid* becoming competitive in performance with traditional servers.

This report explores the creation of clusters using these devices, e.g. what benefits and downsides there are compared to x86-clusters. In particular, lower economic cost and decreased power consumption are compelling benefits. However, the subject of building clusters from ARM-devices is not a mature field, and there is a lack of tools and research on the subject.

We present the *SuperK Cluster Management Toolkit*, a prototype software suite for managing clusters of single-board computers. In addition, we discuss problems particular to clusters of single-board computers and contrast different considerations in the design of cluster management software.

Keywords: cluster computing, single-board computer, cluster management, Odroid, ARM

Sammandrag

Datorkluster har fått en ökad vikt inom områden såsom *high-performance computing* och databashantering på grund av deras skalbarhet och lägre ekonomiska kostnad jämfört med traditionella superdatorer. Dock är datorkluster bestående av x86-baserade servrar dyra, har hög energikonsumtion och upptar betydande utrymme.

Samtidigt har tillgången till billiga men högpresterande enkortsdatorer ökat i stadig takt och produkter såsom *Raspberry Pi* och *Odroid* börjar nå en prestandamässigt konkurrenskrafitg nivå gentemot traditionella servrar.

Den här rapporten utforskar skapandet av kluster bestående av dessa datorer, bl.a. vilka fördelar och nackdelar som finns jämfört med x86-kluster. Specifikt är lägre ekonomisk konstnad och minskad energikonsumtion tilltalande fördelar. Dock så är ämnet att bygga kluster av enkortsdatorer inte ett välutforskat område, det finns en avsaknad av verktyg och forskning inom ämnet.

I rapporten presenteras *SuperK Cluster Management Toolkit*, en prototyp-mjukvarulösning för hantering av kluster bestående av enkortsdatorer. Därtill diskuterar vi problem som är specifika till kluster av enkorsdatorer och kontrasterar olika alternativ i hur klusterhanteringsmjukvaror kan designas.

Nyckelord: datorkluster, enkortsdator, klusterhantering, Odroid, ARM

Acknowledgements

We would like express our gratitude towards our supervisor Vincenzo Gulisano for his dedication and support to us and our project. We would also like to thank Arne Linde for his efforts in helping us with ordering hardware and answering administrative questions. We would also like to show our appreciation to Marina Papatriantafilou for her continuous enthusiasm and moral support.

Glossary

APT	Advanced Packaging Tool: package manager for Debian-based Linux distributions.
DHCP	Dynamic Host Configuration Protocol: a network protocol for distributing configuration parameters, e.g. IP addresses.
HPC	High-Performance Computing
MPI	Message-Passing Interface: a programming interface for distributed applications.
NPB	NAS Parallel Benchmarks, commonly used benchmarking suite for parallel computation, developed by NASA.
PVM	Parallel Virtual Machine: an older programming interface for distributed applications.
SCMT	SuperK Cluster Management Toolkit: the software suite we present in this report.
SSH	Secure Shell: A secure network protocol for remote shell sessions.
SSI	Single-System Image: the quality of a computer cluster that acts towards users like one single system.

Contents

Chapter 1

Introduction

The demand for high computational power is huge in many fields, e.g. in order to run experimental algorithms and or complex large-scale data processing. Achieving this with a single computer system is unfortunately rather expensive, as it may require uncommon, or even custom-produced computers to achieve acceptable run-times. One alternative to decrease the cost of heavy data processing is to concurrently perform parts of the processing on many affordable, off-the-shelf computers at the same time. Similar solutions can be used to increase the reliability of many applications, where the system as a whole continues to work even if individual computers malfunction.

1.1 Cluster computing

A *computer cluster* is a network of computers connected to act as one, single system, as a way of achieving increased computational ability, reliability, or both [1]. Each computer in a cluster is referred to as a *node*. The *cluster architecture*, i.e. the way the nodes are organised and communicate must be considered, depending on the purpose of the computer cluster. Nodes can be physically close to each other or distributed over a large area. The management of the cluster can be handled by one particular node, which in that case may be referred to as a *master node*. However, a computer cluster can also be managed with more than one master node or no master node at all. Nodes without special responsibilities aside from contributing to distributed applications may be referred to as *compute nodes*.

Computer clusters can be used for a variety of purposes. The presence of multiple nodes makes for good fault tolerance, as it achieves redundancy. This is used for services that require high availability. As more nodes connect to a cluster, the greater the computing power becomes, this can be used to perform complex calculations, such as weather forecasting [2].

A computer cluster is also a way of achieving high computational capacity with low-cost, off-the-shelf computers, which is usually cheaper than buying a high-performance computer. As such, computer clusters have the advantages of high availability, high performance, scalability, and low cost. However, clusters do have disadvantages as well. In order to make a cluster act as a single system, some management solution is necessary, and preparing a cluster for use can be time-consuming if each node requires manual setup. Another disadvantage is that the energy consumption of a traditional computer cluster generally increases faster than the computing capacity with the addition of nodes [3]. One technique of avoiding this effect is to use energy efficient hardware [3].

1.2 Characteristics of single-board computers

A single-board computer is a computer which is mounted on a single circuit board. Single-board computers such as *Odroid* [4] or *Raspberry Pi* [5] are among the most affordable computers purchasable today with sufficient processing power for effective clustering [6]. *Odroid* and *Raspberry Pi* have ARM processors, I/O ports, RAM, a memory card slot and a DC power jack, which is typical for most commercial single-board computers. ARM processors offer higher power efficiency compared to x86 processors [7]. These single-board computers are, in other words, functional general-purpose computers with lower energy consumption compared to a traditional server. However, in order to achieve a cluster with competitively large processing power, a large amount of such devices is necessary.

1.3 Challenges of using single-board computers in a computer cluster

Setting up a large cluster manually would mean repeating the same work for each connected node. *Cluster management* applications, which to various degrees automate this process, exist for clusters consisting of traditional, x86-based servers, for example, *OpenSSI* enables the addition of new nodes with minimal set-up for x86-based Linux clusters [8]. However, the availability of such tools for single-board computers is somewhere between limited and non-existing.

Other projects exploring clustering single-board computers also faced this challenge. In *the Bolzano Raspberry Pi Cloud Computing Experiment* [9], a cluster consisting of 300 *Raspberry Pi* nodes was set up. To avoid the need to configure each node manually, a software tool to handle basic configuration automatically was implemented within the project.

1.4 Project purpose & goals

The purpose of this project is to explore methods for simplify the set-up and maintenance of a cluster of single-board computers. This is done by developing a software prototype called **SuperK Cluster Management Toolkit (SCMT)** and testing it on a single-board computer cluster. The development process focuses on having support for three central areas:

1. Cluster management: automatic set up and configuration of all nodes in the cluster.
2. Monitoring: tracking the state of each node in the cluster.
3. Cluster applications: support for common cluster use-cases.

For usability and further development, we will build SCMT in a modular way and provide a user guide which will explain how the software works and how to use it.

We will also explore the efficiency of executing parallel programs on a cluster of single-board computers by running benchmark tests on differently sized single-board computer clusters.

1.5 Limitations

In order to limit the scope of the project to what can be completed in a reasonable amount of time, it is necessary to narrow down what exactly the research and development should focus on. Additionally, the limitations must be narrow enough that it is possible to develop and test on the hardware available to the project group. For these reasons, SCMT is built on the following assumptions:

- Devices all run a recent version of the operating system *Ubuntu* (15.04).
- There must be one device in the cluster which can act as a gateway between the cluster and external networks (including the Internet).
- All devices use the same processor architecture (homogeneous cluster).

- Due to limited access to single-board computers, the cluster on which we test SCMT consists of no more than eight *Odroid XU4*s.

In practice, the first limitation is not so inconvenient as it may appear since it corresponds to how *Odroid*-devices are pre-configured when ordered.

1.6 Related work

The *Open SSI Cluster Project* [8] started in 2001 and had the goals of creating SSI software consisting of open source components. The result was *OpenSSI*, a cluster management software solution for x86-based servers. *Rocks Cluster Distribution* [10] is another similar project where software to enable users to easily set up their own cluster was created. As with *OpenSSI*, the software from *Rocks* is not compatible with ARM processors. *OSCAR* [11] is a project that, as with *OpenSSI* bundles different open source software to create cluster management software for x86-based servers.

Chapter 2

Problem description

There are many problems and challenges one is faced with when creating, managing, and using a computer cluster. In this section, we detail these problems, along with some alternatives which may be used to solve them.

2.1 Cluster architecture

There is no clear choice of architecture for a computer cluster - there are many alternatives, and which is most suitable depends on factors such as:

- **The size of the cluster**: i.e. the number of nodes. As the number of nodes grow, so does the workload of managing the cluster, and the network traffic for communication between management software and compute nodes. Additionally, in a larger cluster, having a single master node managing the entire cluster may be undesirable as it becomes a single point of failure for the cluster as a whole.
- **Performance versus robustness trade-offs**: a cluster may be constructed to avoid or lessen the impact of failures (via e.g. redundancy such as multiple master nodes) at the cost of increased resource utilisation.
- **Cluster use-cases**: whether the cluster is intended to run distributed application over all nodes, or to run several applications distributed over different subsets of the cluster.

There are also a number of hardware considerations which must be present in the cluster design:

- **Compute nodes**: the computers that do the actual work the cluster is intended for [11], [10].
- **Cluster network**: the computers comprising the cluster must have some way of communicating with each other [11], [10].
- **File sharing**: provide file sharing system to the nodes of the cluster, for example, the Network File System (NFS) is a commonly used protocol [11], [10].
- **Network gateway**: used to connect and separate the cluster from the outside world. This allows the cluster to have relaxed security considerations internally, since all traffic into and out of the cluster must pass through the gateway [11], [10].

A simple solution regarding file sharing and the gateway is to provide them on the same server. This is indeed the method used by OSCAR, a project with similar aims as SCMT [11].

Other designs distribute parts of this, for example, it is possible to distribute the file system over several servers - perhaps even over every single node.

For the network, an Ethernet network set-up is typically used. Additionally, a secondary network may be used for high-performance communication [11].

2.2 Cluster management

All nodes must be correctly configured. This entails e.g. network configuration, so that nodes may communicate, making sure all needed software is installed across all nodes, and the state of the entire cluster must be tracked.

Cluster management can be used to create the illusion that, from the users point of view, the cluster is one single system. This is called *single-system image* (SSI) [12]. SSI is accomplished by automatically managing software installation, configuration and program invocation across all nodes in the cluster.

This may be implemented as a master service running on a master node which tracks the state of all nodes in the cluster and invokes commands on nodes on certain events [11], [13].

An important issue in cluster management is how the master node invokes commands on the compute nodes, and how the nodes execute these commands. One solution would be to install a service on each node which can execute a pre-programmed set of commands. This service, running on all nodes in the cluster, responds to requests to update configuration, install software, and execute programs. Such requests are sent from the master service, completing the single-system image [10].

2.3 Enabling distributed computing

One of the most compelling use cases for a computer cluster is distributed computing. However, even if a cluster is connected and properly configured, creating and running distributed programs is not a trivial task. There are several issues:

- Task scheduling: how and when processes are invoked on what compute nodes
- Coordination: distributed processes normally have to work to collaboratively to achieve some goal: this requires e.g. message passing between processes.

Given these problems the project should in some way enable users to in a straightforward way run distributed applications.

2.4 Monitoring

Computer clusters are large, complex systems. As the size and complexity of a cluster increases, so does the propensity for failures and indeed the difficulty of solving them. This is natural; the more nodes a cluster has, the more points of failure and the more work must be done to find where the error occurred.

To lessen the impact of these problems, a monitoring solution may help. Monitoring helps users keep track on the state of the cluster, e.g. by providing logs or graphs which may show abnormal behaviour. Additionally, monitoring software may help with determining precisely which node has failed very quickly after it happens. This allows users to inspect the failing node without blindly searching the entire cluster, or worse, leaving the user entirely unaware that something has gone wrong.

As stated in the project purpose, SCMT should provide support for cluster monitoring by the use of existing monitoring solutions.

2.5 Scalability issues

Ideally, the performance of a computer cluster should increase linearly with the number of nodes. Indeed, this should, in theory, be achievable for any highly parallel application.

However, in practice, some factors may limit the scalability. A cluster's scalability is dependent on how well the network and cluster management software handles the increased workload that comes with more nodes. As the network load approaches saturation; or as the load on the cluster management software approaches 100% of CPU-time or memory usage on the nodes on which it is run, adding new nodes will not scale well.

In particular, network limitations are often a major source of limited scalability in real systems, as applications usually need to share significant amount of data. This may result in a linear increase in network bandwidth usage as nodes are added into the cluster [14].

Chapter 3

SuperK Cluster Management Toolkit - Overview

We present the SuperK Cluster Management Toolkit, SCMT, as a prototype solution for the problem of managing clusters of single-board computers.

3.1 Cluster architecture

SCMT currently assumes the cluster will have one master node and several compute nodes. The master node is responsible for tracking the state of the cluster, invoking commands on compute nodes, acting as a gateway to the clusters, hosting the file-server, and routing network traffic. This massively reduces the complexity of the toolkit itself and is a proven architecture for managing modestly sized clusters (less than around 100 nodes) [10], [11]. However, as mentioned in section 2.1, this may limit scalability and definitely introduces a single-point of failure for all the aforementioned tasks. Further work would be to expand SCMT to enable distribution of master node tasks, which would improve both scalability and robustness; or to allow back-up master nodes to take over all master node tasks should the primary master fail.

3.2 Software components

SCMT is written in the *Go* programming language, together with scripts in various languages, predominantly *Bash-script*. As can be seen in figure 3.1 there are two main parts of the software: the *Daemon* and the *Invoker*. The *Daemon* is a part of SCMT that will be running as a background process. Users pass commands to SCMT which may then pass them to the Invoker. Two examples of SCMT commands that are passed to the Invoker are:

- **register-device**, which registers a node
- **install-plugin**, which enables and installs given plugin on the master and all nodes

The invoker then sends data over a TCP channel to a package called *Invoked*, a package in the Go programming language is a modular part of a program. The purpose of Invoked is to receive packets and execute actions based on the instructions, the *Invoked* package is handled by the *Daemon*. Invoked calls numerous packages needed to handle the tasks that occurs when managing a cluster. A short explanation of the packages used by *Invoked* follows:

- **Master** handles the installation of plugins and initialisation scripts on the master and devices and do also keep track of their state, the package uses the **Devices** package to do actions on each device.

- **Database** is used to handle the cluster state database. Other packages use the database package in order to query the database.
- **Heartbeat** is a package to track which devices are accessible and which are not by continuously pinging all devices in the cluster.

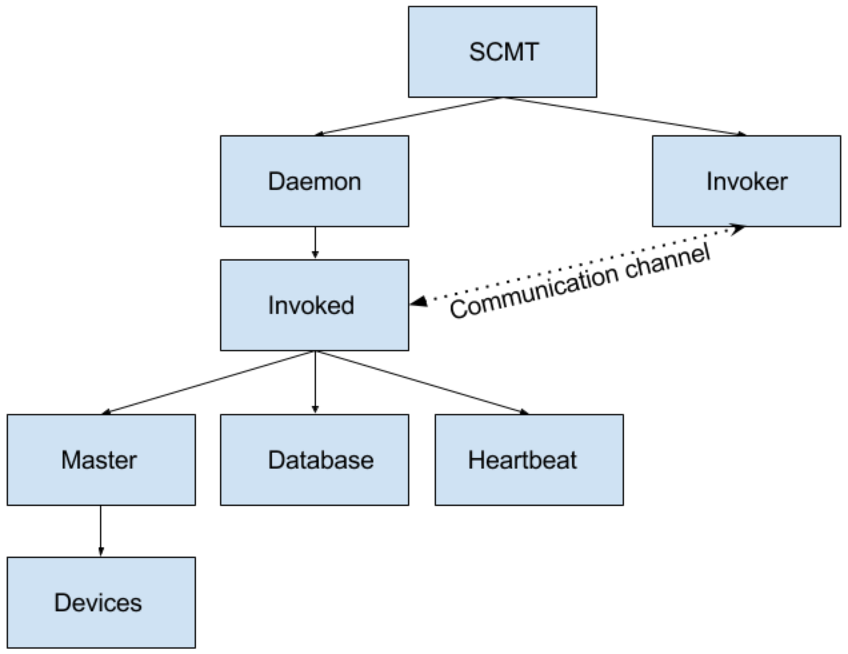


Figure 3.1: Overview of SCMTs software architecture

3.3 User guide

Together with the SCMT software suite, we also provide a user guide to help users understand how to use the software and how it works. The user guide is supplied together with the SCMT software itself.

Chapter 4

Setting up the test cluster

A part of the project purpose is to test SCMT in a single-board computer cluster. This chapter will describe the process of setting up a single-board computer cluster: both the hardware we used and how we mounted the cluster is covered.

4.1 Hardware

We used the following hardware to create our test cluster:

- Seven *Odroid XU4* devices and one *Odroid U3*, used as the compute nodes.
- A desktop PC with *Ubuntu* installed on a virtual machine, used as the master node from which SCMT is run.
- A 16-port gigabit ethernet switch, used to connect the *Odroids* and master node to each other.
- A modified PSU, power supply unit, used to power the *Odroid XU4s*.
- Nine twisted-pair ethernet cables of various length.
- A 10 outlet power strip.

The power supply is modified so that it provides seven 5 Volt DC jacks, supplying up to 30 A, which is more than enough for the amount *Odroid XU4* devices in the cluster.

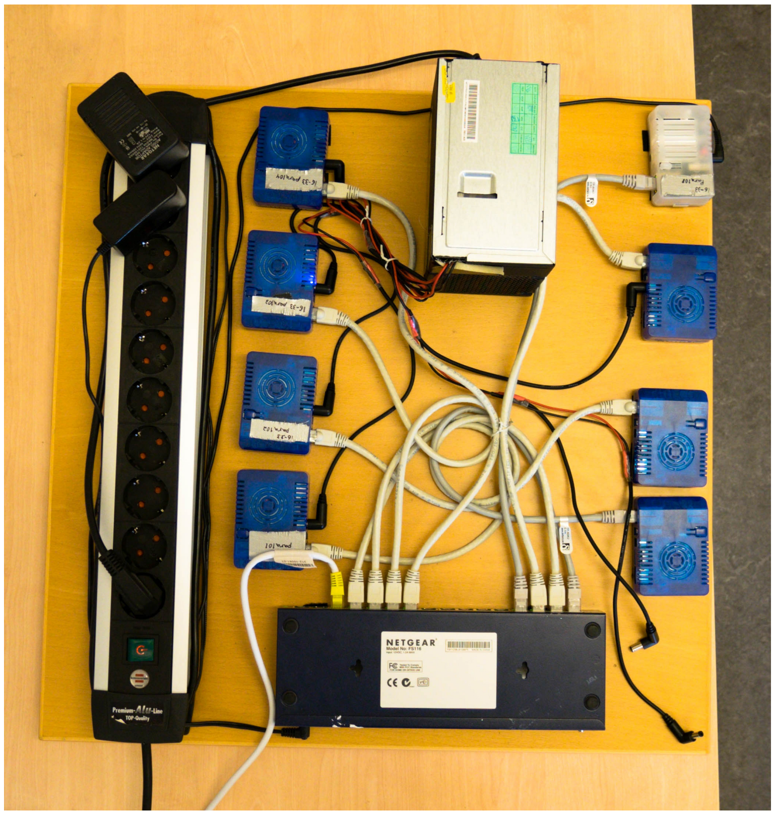
We chose to use Odroid XU4 devices, as we had several of these available at the start of the project.

4.2 Physical mounting

In order to achieve a well-structured mounting layout where cables and the addition of new devices are easily handled, a well thought-out design of the physical mount is required. *Odroids* are delivered disassembled to a certain degree. Although the *Odroids* used in this project did not require much assembly, they still needed to have the memory card (*eMMC*) mounted on the *Odroid* circuit board and have the circuit board well secured within a plastic casing.

These encased *Odroids* were then mounted on top of a wooden surface using velcro-tape, as can be seen in figure 4.1c, with enough space to connect both power and network cables. The devices were placed to enable access ones that needed to be replaced, or for other reasons be physically accessed. The ethernet network was routed through a switch which was also mounted on the board. The PC PSU powering each node can be seen in 4.1b.

The PSU was then connected to power strip, which was also attached to the surface. This way the computer cluster was portable and not bound to a single place, as can be seen in figure 4.1a.


(a) Overview of the prototype cluster

(b) PSU that powers the connected devices

(c) Velcro-tape used to attach connected devices to the board

Figure 4.1: Physical mounting for our prototype, using *Odroid* devices. For larger pictures, see appendix A.

Chapter 5

Area 1: Cluster management

SCMT should provide support for cluster management. This problem was introduced in section 2.2. Furthermore, since we intend to automate the process of setup and configuration, further complexity is added to the problem. This chapter presents how cluster management is implemented in SCMT.

5.1 Distribution of data - shared file systems

It is necessary to share data between the computer nodes within the cluster: both data needed as input to computations, and data needed to manage the nodes. Two alternatives for sharing data in clusters are:

Network File System: Network File System (NFS) is a shared file system that can be used to share files over the network. The principle is simple: directories are mounted seamlessly into the directory structure of any Unix system.

Lustre: Lustre is a parallel file system that can be distributed over several devices. It is designed for scalability and performance.

NFS is older than Lustre and has been in development since 1989 [15]. This makes NFS more mature compared to Lustre (which was first released in 2003 [16]), there have been four new versions and iterations of the first protocol [17]. This makes it fair to assume that NFS is more stable than Lustre.

Lustre was designed to run on large computer clusters [18], and to be able to store large amounts of data. With scalability and performance in mind, Lustre works very well on large systems. It also has the advantage of being able to store very large files distributed over several devices [18].

Despite the listed benefits of Lustre, we chose NFS, due to its simplicity and the fact that it covers all the needs our project requires. Lustre is complicated to set up, and consists of several individual services maintaining the file system over several devices, which was deemed unnecessarily complicated for our prototype.

5.2 Dynamic distribution of network addresses

When managing a large number of nodes, it is necessary to distribute network addresses and identities automatically. In order to achieve dynamic distribution, the *Dynamic Host Configuration Protocol (DHCP)* protocol is ideal, as its purpose is to distribute network addresses dynamically. Dynamic distribution removes the need to manually assign a certain network address to any given node.

There are, however, some downsides in dealing with nodes dynamically. Dynamically assigning network addresses makes it difficult to physically identify any particular device in a large cluster, as there is no correlation between address and physical location. In the event any device malfunctions, and needs to be replaced or repaired, the user will not be able to find that device using only the network address.

There are several applications that run on a Linux-systems which manage devices using the DHCP protocol: *DHCPD* and *Dnsmasq*. In this project we chose to use DHCPD, due to a feature which allows script execution at certain events. This allows us to send a request containing newly connected device's IP-address and MAC-address to our software whenever a DHCP lease is renewed. The software will then determine if the new lease is a new device which requires configuration, or an already configured device which does not require further action, by looking up its MAC-address in the database.

5.3 Device detection

The first step of getting a fully automated setup process is the detection of connected devices. This is in order to detect when a new device connects for the first time, or when an already connected device disconnects (e.g the device malfunctions or is otherwise prevented from working correctly).

There is a certain set of *events* that need to be captured in order to automatically handle devices:

- **Connection:** when a new device connects to the cluster. The newly connected device should be initially configured to work with the cluster, and added to the device database. This needs to be done before any jobs can be scheduled on the device.
- **Reconnection:** when a device is connected to the cluster, depending if it is a newly connected device or a previously connected device it should then, if needed, be upgraded or treated as a new connection.
- **Disconnection:** when a device disconnects from the cluster. The user should be informed that a device disconnected from the cluster. This could be due to hardware or software faults, or that the device has been misconfigured.

All these basic events make up the backbone of the automation process. Capturing these events is vital if the cluster should be able to manage an arbitrary amount of devices dynamically.

In *SCMT* we use *DHCPD* to assist in detection of the **connection** and **reconnection** events (see section 5.2). This works by configuring DHCPD to run a script or a program when a new lease is handed out or renewed. Configuring this is simple, it only requires adding one line to the original DHCPD configuration.

```
subnet ... netmask ... {
  ...
  on commit {
      execute("/usr/bin/scmt", "register-device",  mac, ip);
  }
  ...
}
```

This, in turn, invokes the registration process in the daemon of SCMT. See figure 5.1 for an overview of the registration process of a new device.

Disconnection is detected via the *Heartbeat* package (see section 3.2). If a node is disconnected, at the next heartbeat it will be detected as unreachable and the disconnection event is invoked.

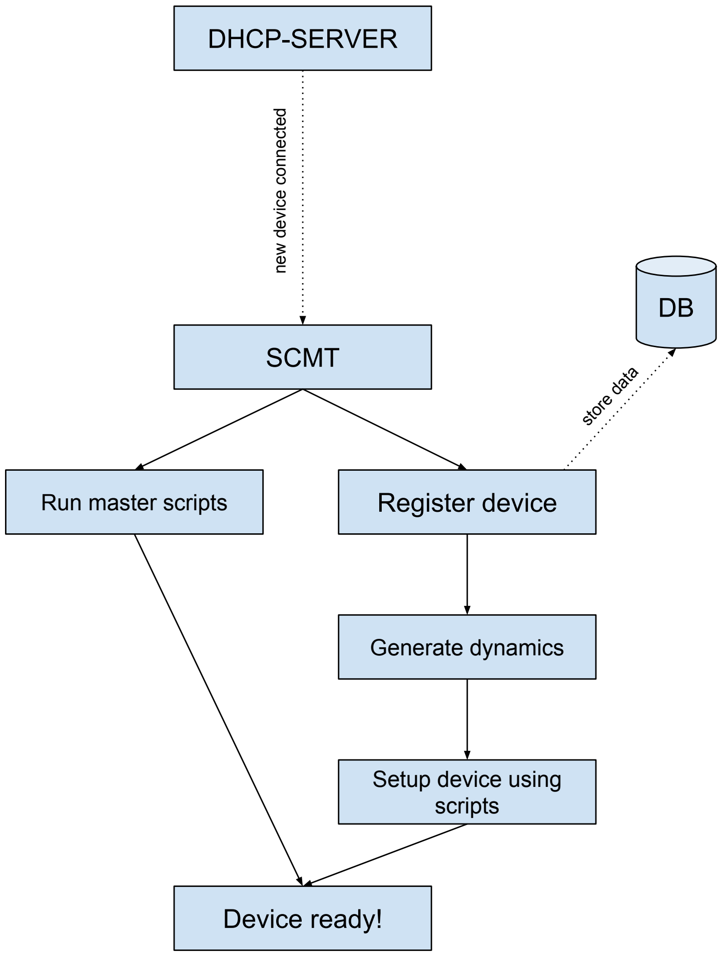


Figure 5.1: Device detection process in SCMT. A new lease from the DHCP-server invokes SCMT. The device is then registered, added to the database and configured. The master node is then configured to handle the new device.

5.4 Registration process

During the registration process of a new node, the appropriate scripts are run both on the new node and on the master node. The node is then assigned a special ID-number. From that ID-number, the hostname is generated, as well as the network address of the node.

Even though DHCPD is used as a gateway, all of the nodes will have a persistent network address once the setup procedure is finished. The network address is generated using the node's ID-number.

Setting base-address such as 10.46.1.1, and then encoding this network-address into a 32-bit integer value, we can generate the next consecutive network-address using simple addition, and then reverse the previous process to get the network address, as seen in formulas 5.1 and 5.2.

How address calculation is done (using C-style syntax, | for bitwise or, << for left shift):

$$base = (10 << 24)|(46 << 16)|(1 << 8)|1 \tag{5.1}$$

$$ip_{device} = base + 1 \tag{5.2}$$

The initial DHCPD pool of network addresses (typically a /24 subnet) is only intended for new devices connected to the network before they are assigned a static IP-address.

5.5 Command execution on nodes

One of the more difficult aspects of plug-and-play cluster management is how to execute commands on the nodes. The commands could be e.g. to install a piece of software, to edit a configuration file, or to change the node's hostname, etc.

As mentioned in section 2.2, one way to solve this problem would be to have a service running on all nodes that awaits commands from the master, then executes them. However, this would require manual installation of said service on every node. This is undesirable, as a goal of the project is to reduce manual setup to as large degree as possible.

One method which would not require user intervention is using network boot. Nodes could boot an image which already contains the desired operating system and a cluster management service over the network. However, this seems impossible, as the targeted single-board computers (e.g. *Odroid XU4*, *Raspberry Pi*) lack support for network boot.

Instead, SCMT assumes, as mentioned in the previous section, that Ubuntu 15.04 is installed on all nodes. Then, it uses Secure Shell (SSH) to execute commands on other nodes. This requires that SCMT is configured to connect to the correct user with the right password. Usernames and passwords for nodes are stored in a configuration file. SCMT attempts to connect with the first username-password pair in the configuration file, and, if that fails, moves on to the next username-password pair. Thus, SCMT can function even if nodes have different usernames and passwords, as long as they are all present in the configuration file.

5.6 Event handling implementation

There are certain events at which the cluster management software must invoke some action - e.g. when a new device is connected or disconnected. At that point, the new device must be registered, as is detailed in section 5.3-5.4; and set-up for use in the cluster. In the same way, a disconnected device should be unregistered from the system, as it no longer is part of the cluster. In order to implement this event-handling in a modular manner, we use Bash-scripts to achieve this.

The event at which a script is to be executed is determined by its placement in *event directories*. Within such a directory, execution order is determined by the lexical ordering of the script filenames, e.g.:

```
00-base.sh
10-set-hostname.sh
20-expand-filesystem.sh
...
```

SCMT manages input parameters into the scripts via a pre-defined set of environment variables, which then act as an API for SCMT scripting.

As mentioned above, scripts are associated with *events* by being placed in certain event directories. These directories are:

- `master.init.d`: scripts will run on the master node when the system is initialised for the first time.
- `device.init.d`: scripts will run on each compute node when the system is first initialised and on newly connected compute nodes.
- `master.newnode.d`: scripts will run on the master node whenever a new compute node is detected.

- `master.removenode.d`: scripts will run on the master node whenever a compute node is removed from the cluster.

5.7 Software Dependencies

The computer that SCMT is invoked on becomes the master node in the cluster. This master node will need to update or install services needed by SCMT itself:

- **MySQL:** SCMT will store all MAC addresses to any arbitrary connected device in the cluster in a MySQL database.
- **Approx:** Nodes are not directly connected to the Internet, all Internet traffic would have to be routed through the master node. However, nodes do need access to software packages. Approx allows the master node to act as a cache for software packages. This avoids network congestion when all nodes request the same packages.
- **Realpath:** Determines the absolute canonical path to files in the file system.
- **NFS:** Sharing a filesystem over a network.
- **DHCPD:** Detecting newly connected devices.

All these services are needed by SCMT in order to configure the cluster and be able to update nodes and install plugins on computer nodes within the cluster. The installation is done automatically at the *master init* event (see section 5.6).

5.8 Plugins

In order to make effective use of a cluster, a certain set of software has to be installed on it. Which software depends on what the purpose of the cluster is. In order to achieve modularity and flexibility for different use-cases when instaling extra features, a plugin system is used.

SCMT bundles a certain set of such plugins for managing software, such as *Ganglia* for monitoring or *OpenMPI* for running message-passing programs. Plugins may be enabled or disabled at any time, and SCMT will automatically configure all nodes in the cluster accordingly.

The plugin system is designed to allow creation of new plugins without needing to rebuild SCMT itself, all that is required is a set of executable scripts which will run on the system at certain events. The events, and the directory structure, is identical to the events of the core system, which is described in section 5.6. As such, the plugin system mirrors how the core cluster management works, but only for a single, self-contained segment of the cluster management (often software installation).

Chapter 6

Area 2: Monitoring

The second area which SCMT should have support for is monitoring. We have looked at existing software solutions and how to integrate them into SCMT. This chapter presents different monitoring software solutions and describes how we evaluated them with respect to the kind of cluster on which SCMT is intended to run. The chapter later continues to, in more detail, analyse the most appropriate software solutions, finishing with a description of how we integrated monitoring into SCMT.

6.1 Software properties to consider

When searching for a suitable monitoring software for this project a number of requirements were taken into consideration, both regarding how the software works and which features are available. Apart from the requirements mentioned below, we were looking for open-source software, so as to be able to extend and customise the software if necessary. For obvious reasons, the software must be compatible with ARM processors.

6.1.1 Available features

A monitoring solution should present relevant metrics to the user. These metrics could be e.g. CPU load, the number of processes and their state, and the number of connected nodes. Allowing users to customise which metrics to monitor is important for usability. The ability for the monitoring software to send notifications to the user if a metric pass a certain threshold is desirable, so that the user may be informed of alarming events. Getting statistics on how much computing power exists in the cluster, and getting feedback on how it grows as more nodes are connected, is something we looked for: some kind of performance test integrated into the monitoring software.

Moreover, these features should be available in a manner that is approachable for users without much technical expertise.

6.1.2 Resource efficiency

Since a computer cluster is used for tasks which may require large amounts of processing power, it is undesirable to dedicate undue amounts of resources on a monitoring tool. It is therefore crucial that the chosen monitoring tool consumes little computing resources. The usage of both physical memory and CPU time was taken into account.

6.1.3 Scalability capabilities

Depending on the size of a cluster, scalability of a monitoring software is a more or less relevant concern. If, as with this project, the size is arbitrary and the goal is to without difficulty expand the size of the cluster, scalability becomes a highly relevant factor in choosing what monitoring software to work with. As such, we desired a monitoring solution with good scalability. That is, for each node added to the system, the load on the master node increases only slightly.

The monitoring solution *Ganglia*, for instance, is designed with a focus on using effective algorithms to provide support for large computer clusters, while some other monitoring solutions are not designed to handle large computer clusters. *Munin*, which is a lightweight monitoring tool, provides a very simple plugin architecture [19, p.4].

6.1.4 Pushing vs Polling

Polling means the the master node requests status data from the compute nodes at a predetermined time interval. Choosing this interval can be difficult, as it should be short enough to detect potentially alarming events in a reasonable amount time while still being long enough so as to not reduce performance by requesting data that has not changed much since the latest poll. What time interval is optimal depends on what kind of statistics are being handled. Critical data, or data that has a large change rate, are examples of data for which frequent polling is desirable, while infrequent polling suffices for data of the opposite kind: data with a low change rate, or non-critical data.

While fetching data, there is a risk of a network bottleneck where all compute nodes are trying to send their data simultaneously.

In contrast with a polling-based solution, there is pushing: each compute node sends data to the master node when needed. This implies that the master node needs to always be ready to receive data. The advantages of using pushing is that it is customisable: the nodes decide when it is time to send metrics to the master node. This way, bottlenecks can be avoided to a large degree by making sure all nodes do not send data at the same time. On the other hand, setting up this system requires more work, since every node needs to be configured.

6.1.5 List of desirable features

To summarise, we looked for software that:

- Is ARM compatible
- Is resource efficient
- Is able to send alerts
- Has a graphical interface
- Provides workload statistics for each node
- Provides summarised workload statistics for the cluster overall
- Has performance tests
- Provides information of which nodes are connected
- Is open-source

6.2 Comparing the software

With the list of features 6.1.5 in mind we found seven software solutions which we chose to evaluate. These seven were: Munin, Nagios Core, Ganglia, Bright Cluster Management, Scyld ClusterWare, supermon, and eZ Server Monitor.

Ganglia, Munin, supermon and eZ Server Monitor are all open-source, while the software from Bright Computing, Bright Cluster Manager [20]; and from Penguin Computing, Scyld ClusterWare [21], are not. Nagios Core is only partially open-source. eZ Server Monitor only provides a client architecture with no central server [22], which makes it unsuitable for cluster monitoring. Nagios Core is an event-detection monitoring solution without a focus on performance trends [23], such as what Munin and Ganglia provides. Supermon was last updated in 2008 and is a programming interface for monitoring [24], which is not relevant for our purposes.

Table 6.1 summarises the features of each software that we compared.

	Monitoring Software						
Requirements	Munin	Nagios Core	Ganglia	Bright Cluster Manager	Scyld ClusterWare	supermon	eZ Server Monitor
ARM compatible	✓	✓	✓	?	?	?	✓
Resource efficient	✓	X	✓	?	?	✓	✓
Custom event notification	✓	✓	X	?	?	X	X
Graphical interface	✓	✓	✓	✓	✓	X	✓
Node workload stats	✓	✓	✓	✓	✓	?	✓
Cluster workload stats	X	X	✓	✓	✓	X	X
Performance tests	X	X	X	X	X	X	X
Connected nodes	✓	✓	✓	✓	✓	✓	?
Open source	✓	Partly	✓	X	X	✓	✓

Table 6.1: Different monitoring software's features. Munin and Ganglia fulfil the most requirements. We can also conclude that performance tests are not a typical feature for monitoring software solutions.

We chose to further evaluate the products, and automate the installation process of Ganglia and Munin, which both are monitoring tools that make use of the software *RRDtool* for logging and graphing large time series with good performance [25]. RRDtool is a proven software, which is industry standard according to its creator [25]. Furthermore, Ganglia provides linear scalability with respect to the number of nodes [26], which is desirable.

Because of the fact that both Ganglia and Munin are actively developed in combination with having a large user-base, our focus could be on evaluating and automating the software without spending much time researching how to use them and work around potential issues.

6.3 Munin

Munin is a lightweight monitoring tool that focuses on plug-and-play functionality and showing results in graphs for analysis of performance trends of nodes separately [27]. The Munin architecture is as shown in figure 6.1 and illustrates the master node (Munin-Master) and its data collection process. The master collects data in the form of logs from each connected node (Munin-Node) at regular five-minute intervals, as such, Munin is a polling software. The data is then stored in an RRDTool database, and graphs are generated based on the data. Users can also define alarms to warn users when measurements reach certain values; warnings appear in a web panel and in e-mail notifications. Munin is designed to be easy to extend with plugins for adding new statistical data [19, p.4].

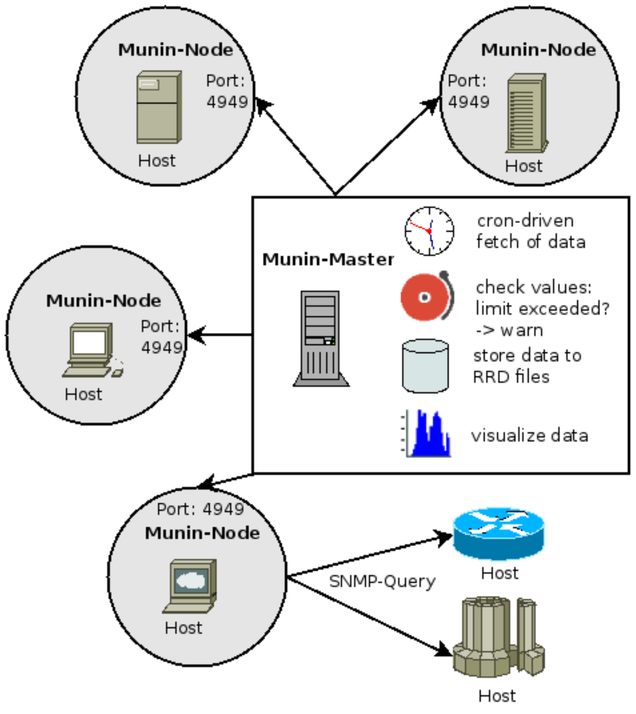
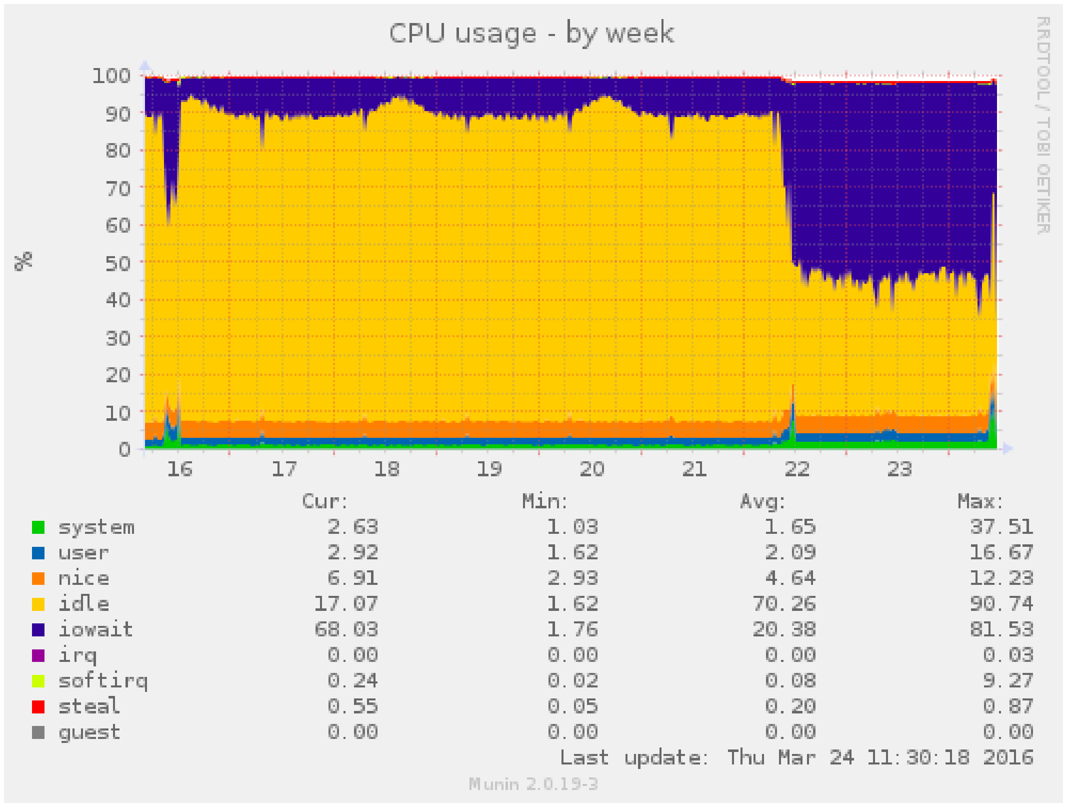


Figure 6.1: Munin's architecture. Illustrates the master node process of data collection from each connected node, limit value checking, data storage and drawing graphs. Used with permission from Author and Copyright owner: Gabriele Pohl [28].

An example graph can be seen in figure 6.2 which shows CPU usage by week for the master node at that time. The graph illustrates the benefits of using a monitoring tool, as a clear increase in io-wait CPU time is observed from the 22nd and onwards, enabling users to easily find the root cause of possible performance problems.



	Cur:	Min:	Avg:	Max:
🟩 system	2.63	1.03	1.65	37.51
🟦 user	2.92	1.62	2.09	16.67
🟧 nice	6.91	2.93	4.64	12.23
🟨 idle	17.07	1.62	70.26	90.74
🟦 iowait	68.03	1.76	20.38	81.53
🟪 irq	0.00	0.00	0.00	0.03
🟩 softirq	0.24	0.02	0.08	9.27
🟥 steal	0.55	0.05	0.20	0.87
⬛ guest	0.00	0.00	0.00	0.00

Last update: Thu Mar 24 11:30:18 2016

Figure 6.2: Example of Munin graph displaying CPU usage by week. There is an increase in I/O wait time from the 22nd.

6.4 Ganglia

Ganglia is a highly scalable monitoring solution which is designed for computer clusters [29]. As with Munin, Ganglia focuses on enabling statistical analysis with graphs (see section 6.3 for Munin). Figure 6.3 shows an example graph from Ganglia where the cluster load is displayed. The graph shows the number of nodes, the number of user processes and the number of CPUs. It also shows the 1-min load of the cluster, that is, how large the utilisation of the cluster is. In this case, the utilisation average is 4.4 out of 29 available CPU cores.

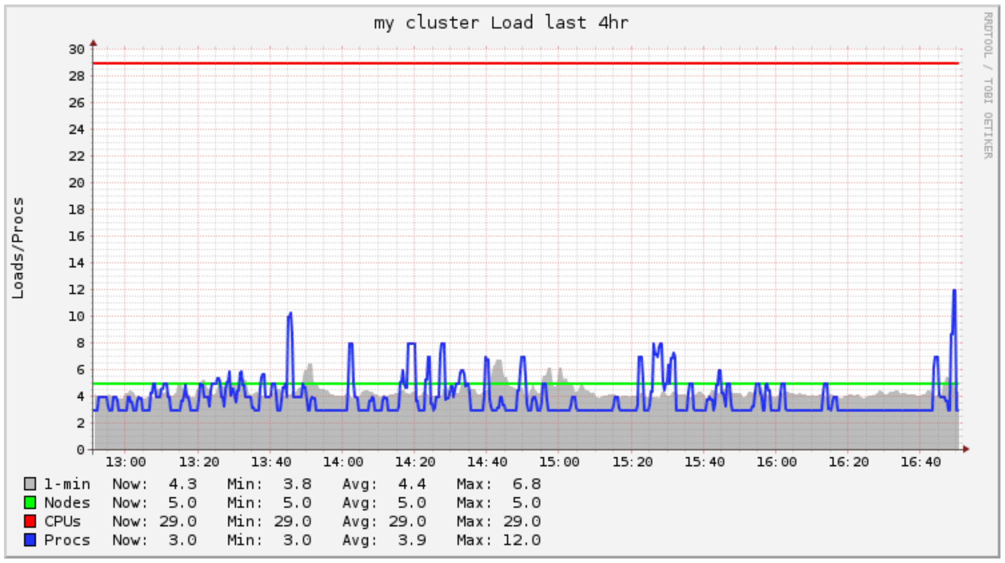


Figure 6.3: Graph generated by Ganglia, displaying the load on the system, and how many CPU cores exist, in this case 29.

Furthermore, Ganglia supports two methods for node communication. Unicast, which is a one-to-one connection between the master node and each compute node; and multicast, which is a many-to-many connection between all compute nodes. A combination of the two is also a possible setup [30, p.20-22]. The use of unicast means that each compute node does not store any other data than its own, which means less load per compute node. It requires one node, in our case the master node, which will listen and collect data from all other nodes. This setup is not very fault tolerant, since if the master node fails, there is no way to report data. The multicast setup results in all compute nodes broadcasting their metric data for any other node to listen to. This can lead to all nodes storing data of every node in the cluster, which puts a higher load on each compute node than with the unicast setup. The multicast approach is not as suitable for large clusters, but it provides a high degree of fault tolerance, since any node can be a backup for the master node [30, p.20-22].

Since SCMT currently relies on the master node to always be in working order regardless of monitoring considerations, we chose to work with the unicast setup. Should the project be extended, the possibility of a hybrid solution could be considered.

With Ganglia (as opposed to with Munin) the nodes push their data to the master. Each node is configured with a time interval to send data (default is 30 seconds). In addition, the nodes store their own data at a configurable time interval (default is 10 seconds). As such, the data represented in Ganglia-produced graphs is more fine-grained compared with graph produced by Munin, which only presents a snapshot of each node in the system at a five minute interval.

Ganglia has no built-in event notification system, but there are ways of setting up a notification system, for example, the open source solution *ganglia-alerts* [31].

6.5 Running and testing Munin and Ganglia in the cluster

To further evaluate the suitability of using either Munin or Ganglia, both programs were installed on our test cluster. Performance tests were then done, considering CPU usage and usage of physical memory. The tests were performed on clusters of different sizes. First, with only the master node, then with one, two, three and four additional compute nodes connected, to gain insight on the scalability of the software solutions.

6.5.1 How the tests were performed

For testing the CPU usage and usage of physical memory of both Munin and Ganglia, pidstat was used. Pidstat is a program used for reporting statistics of Linux processes [32]. By collecting statistics about each process that Munin and Ganglia run every second for two hours, we summarised the average resource usage on the master node for both software solutions. Since the compute nodes only communicate with the master and not with each other, we assume that the load of running Munin and Ganglia are roughly equivalent on each compute node, regardless of how many nodes are connected in the cluster. Therefore, a corresponding test to that of the master node was performed on one of the compute nodes to gather results on CPU usage and usage of physical memory.

6.5.2 Performance test results

As mentioned above, two different measurements for performance were taken. Figure 6.4 shows the CPU usage of the master when running Ganglia or Munin with different number of nodes in the cluster, while figure 6.5 shows the memory usage. Furthermore, table 6.2 shows the CPU usage and physical memory usage on a compute node, which in this case was an *Odroid XU4*.

The diagram in figure 6.4 shows that Munin has a linear increase in CPU usage that has a growth rate that is significantly larger than that of Ganglia, and figure 6.5 shows a somewhat similar memory usage for the number of nodes that were tested, but Munin has larger growth rate. The CPU and physical memory usage graph in figure 6.2 shows that Ganglia uses more CPU time than Munin, but significantly less physical memory on compute nodes.

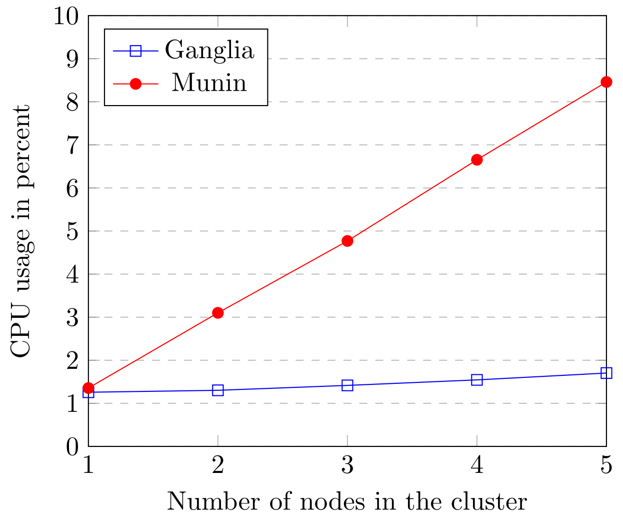


Figure 6.4: CPU usage statistics on master node for Ganglia and Munin. CPU usage increases with a larger difference per additional node for Munin, compared with Ganglia.

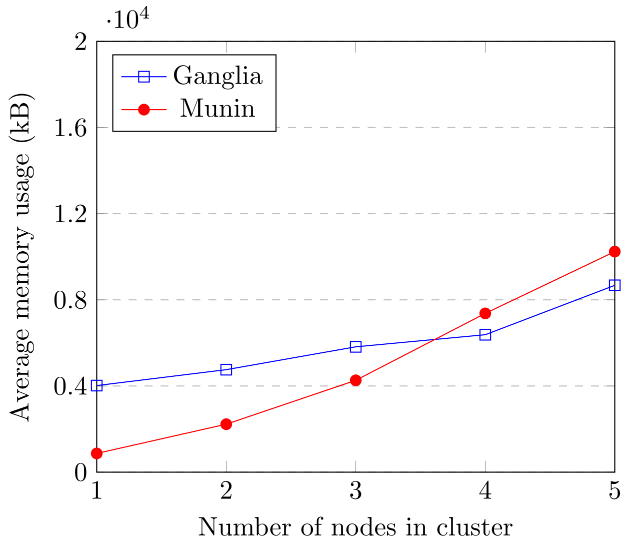


Figure 6.5: Memory usage statistics on the master node for Ganglia and Munin. Both software solutions require little memory.

	CPU usage(%)	Physical memory usage(kB)
Munin	0.016667	275
Ganglia	0.063333	0.54

Table 6.2: The CPU and memory usage of Munin and Ganglia when running on a computational node

6.6 Integrating status monitoring into SCMT

If a small cluster with maximum of four nodes is to be set up, the difference in the effect of using Ganglia or Munin, considering resource utilisation, is not large. For this reason, we chose to enable automatic management of both Ganglia and Munin in SCMT.

Both tools are integrated as SCMT plugins. This means that scripts for initialising the master node, as well as scripts necessary for adding and removing a device from the cluster, are implemented. The scripts are written in Bash-script and Python. Initialisation scripts for the master node downloads and installs required packages using Advanced Packaging Tool (APT). When a new node connects, the required packages are installed and configured on the new node and the relevant configuration files are automatically set up on the master node. A node disconnecting results in execution of scripts that modifies configuration files on the master node.

The choice of which monitoring tool to install (if any) can be made during the installation of SCMT. It can also be done via the command:

```
scmt install-plugin munin|ganglia
```

Chapter 7

Area 3: Cluster applications

A computer cluster is maximally useful if it can run distributed applications. Having support for different use-cases in SCMT is one project's purposes. As such, it is relevant to automatically set up a cluster to do common tasks, in particular, the run-time environments that are commonly used for distributed programs for use in clusters for various applications - e.g. numeric calculations and database handling.

7.1 Finding common cluster applications

SuperK Cluster Management Toolkit simplifies cluster setup by including necessary software to support the most common cluster use-cases. In order to provide such support, which these use-cases were had to be determined. As with the monitoring software solutions, the application support software obviously also needs to be compatible with ARM-processors.

Research on cluster applications led to the following applications:

7.2 MPI

Message Passing Interface (MPI) is a standardised library interface for programming distributed-memory message passing applications. It is widely used in the development of applications intended to run on computer clusters [33]. Thus, it is highly relevant that our project creates a simple method for setting up MPI run-times and compilers in a SCMT cluster.

7.2.1 OpenMPI & MPICH

OpenMPI is one of the most commonly used implementations of MPI. It was created as a merger of three older MPI implementations, all coming from various universities around the world. It is used in some of the fastest supercomputers in the *TOP500* supercomputers list [34].

MPICH is another widely used implementation of MPI. MPICH is public domain software, which has led to it being used as the base for many other (often proprietary) MPI implementations [35], [36].

As indicated above, there are many other MPI implementations. However, most of these could not be used in the project, as they are commercial, proprietary software.

7.2.2 MPI implementation selection

OpenMPI and MPICH are both widely used - the choice between the two seems to be a matter of trade-offs and personal preference. In order to provide for such preferences, and for possible incompatibilities in existing MPI software, we elected to support both. In addition, we found that the installation process for these were very similar, so not much time had to be spent in automating the installation of MPICH once it had been done for OpenMPI.

7.3 Hadoop

Hadoop is a open-source framework developed by Apache. It provides the *Hadoop distributed file system* (HDFS) and computation close to the data in a cluster. Hadoop is designed for scalability, and, as an example, *Yahoo!* has the (as of writing) largest cluster running Hadoop with over 4000 nodes [37].

Hadoop was developed to store large amounts of data distributed over the cluster's nodes in a fault-tolerant manner, where the loss of a single node does not mean that any data has been lost, as files are replicated within the HDFS.

Computations that are performed on data in the Hadoop system are distributed within the Hadoop cluster. This means that the master node will organise and distribute the workload, choosing nodes as close as possible in regards to where the data is being stored within the cluster [38].

7.4 Automated installation of applications

SCMT allows users to trivially set up clusters for the applications we elected to support. This is done via SCMT's plugin installation feature. The selected applications were given their own plugin:

- OpenMPI
- MPICH
- Hadoop

7.4.1 Installation of OpenMPI & MPICH

The installation procedures for OpenMPI and MPICH largely resemble each other. This is to be expected, since they are intended to achieve the same purpose (i.e. compile and run MPI-programs). As such, it would have been possible for the OpenMPI and MPICH plugins to share most code, however, in the interest of modularity, they were each implemented as self-contained, complete plugins.

What follows is a description of the steps taken by SCMT's automated set-up procedure for the OpenMPI and MPICH plugins.

One environment that both OpenMPI and MPICH can use to execute programs is SSH [39], [40]. [1] In order for MPI-programs to run using either OpenMPI or MPICH, the cluster needs to be set up for SSH between the compute nodes, ideally without requiring a password. [40].[2] It is also much simpler (requires less configuration) if all compute nodes have the same username, password, and *UID* (user identifier). For this reason, and to limit privileges granted to other nodes, the MPI-plugins first create a dedicated user, *mpiuser*. This user's home-directory is set-up to be a NFS-shared directory, hosted on the master node. This allows the compute-nodes to immediately access the executable files and required resources to run MPI-programs, while still all using the exact same file-paths.

Setting up passwordless SSH is now trivial: `ssh-copy-id localhost`
Running this command on the master node is enough. Due to the home-directory being NFS-shared, SSH gets configured on all nodes.

The final step is configuring and running MPI-programs. Since this is something that the user must do for each program and execution thereof, it is not possible to automate this within the plugin installation. However, some utilities are offered to simplify doing so. Both OpenMPI and MPICH require information on over which nodes to distribute each job, in the form of hostnames or IP-addresses.

[1] In MPICH terminology, SSH acts as a *Bootstrap Server* [39]

[2] That is, between the non-privileged *mpiuser*s, as opposed to the privileged *sudo* SSH which SCMT requires to invoke commands on compute nodes.

To allow users to not have to specify a (possibly quite large) list of nodes before each MPI-program invocation, the MPI-implementations allow the information to be passed as a configuration file. This configuration file is called a *machinefile* or *hostfile*[3]. However, how the machinefile is specified differs between the two MPI implementations, since they use different task schedulers.

Within the installation scripts, appropriate machinefiles are generated so as to instruct the task schedulers to distribute jobs over all nodes in the cluster.[4] The installation scripts also create two small utility scripts in the mpiuser home-directory for compiling and for running programs on the whole cluster, using the MPI-implementation that was installed. If both MPI-implemenation plugins are installed, the user will thus have two sets of compile-and-run scripts to choose from, abstracting away the differences between using MPICH and OpenMPI.

This setup does not require any user configuration (with one small exception mentioned below). As such, from a user perspective, installing OpenMPI or MPICH is as simple as running:

`scmt install-plugin openmpi/mpich`.

There is one case in which a user may need to manually edit a configuration file before installing the plugins: if one or more of the nodes already have a user with the default UID, then it obviously cannot be used to create mpiuser. As such, the plugin comes with a configuration file where the user can specify another UID for the mpiuser.

7.4.2 Installation of Hadoop

The installation procedure for Hadoop is somewhat similar to that of OpenMPI and MPICH. Hadoop requires a Java runtime, version 1.5 or higher, and thus this needs to be installed before the installation of Hadoop. As with OpenMPI and MPICH, Hadoop also requires passwordless SSH. The group *hadoop* is created and the user *hduser* is added to that group on the master node. An RSA key is then generated and copied onto the slave nodes in order to achieve passwordless SSH.

In the configuration file *hdfs-site.xml*, the value of `dfs.replication` determines how many redundant copies there should be of any file within the Hadoop Distributed File System (HDFS). These copies will exist on other nodes in the cluster, thus preventing data loss within the HDFS.

Hadoop also requires the master node to have a list*(/hadoop/etc/hadoop/slave)* with all the slave nodes listed in order to distribute the files stored within the cluster and the different jobs between nodes in the cluster. This node-list is generated within the installation scripts, if a node is removed from the cluster the same node is also removed from the slave list.

There are some differences between the master node and slave nodes within a Hadoop cluster, the master node runs Namenode, SecondaryNameNode and DataNode processes and the slave nodes only run DataNode processes. The NameNode process keeps the metadata for the cluster.

The SecondaryNameNode connects to the NameNode (once every hour, by default) and copies the cluster metadata directly from memory and also as files containing stored metadata, it then combines these two in order to form an updated version of the entire metadata of the cluster and returns this to the NameNode. This ensures that if NameNode process would stop then there still exist a copy of the metadata of the cluster on the SecondaryNameNode [41].

SecondaryNameNode could be configured to do this operation more frequently than the default, the user can change the configuration file *hdfs-site.xml* under the value of *dfs.namenode.accesstime.precision* [41].

[3]Not to be confused with the operating system hosts-file, which is a map from hostnames to IP-addresses.

[4]Naturally, SCMT automatically updates the machinefiles if nodes are added or removed.

Chapter 8

Benchmarks

One of the main benefits of using a cluster to achieve performant execution of applications is the relatively lower economic cost, in comparison with a single comparably performant computer. As such, it is important to show that clusters consisting of single-board computers may significantly outperform a traditional PC or server of comparable cost.

8.1 NAS Parallel Benchmarks

NAS Parallel Benchmarks (NPB), is a suite of benchmark applications for determining the performance of a computer system in parallel computation [42]. It is particularly suited for testing clusters and supercomputers. NPB includes benchmarks using, among others, MPI for message-passing parallelism, OpenMP [1] for shared-memory parallelism, and benchmarks using OpenMP and MPI together for both shared-memory and message-passing parallelism [42]. This was suitable for our purposes: we had already elected to include support for multiple MPI implementations (OpenMPI and MPICH) and had installed both on our test cluster (see section 7.4.1). This test cluster, and indeed the single-board computers that SCMT targets, have multiple processor cores per node, so in order to get the most performance out of the cluster it was appropriate to use a test that makes use of all cores. Therefore, we used an MPI-based benchmark (BT-MZ MPI), running four processes per compute node to make use of the available hardware-resources for parallelism.

The tests ran were as follows:

- Test type: BT-MZ
- Problem classes: A, B, and C
- Number of nodes: 1, 2, 3, and 4
- 4 processes per node

Additionally, as a comparison, we performed the same benchmark on a single x86-based PC with the following specifications:

- Intel i7-4500
- 8 GB RAM

[1] *Open Multi-Processing*, not to be confused with OpenMPI

8.2 Benchmark results

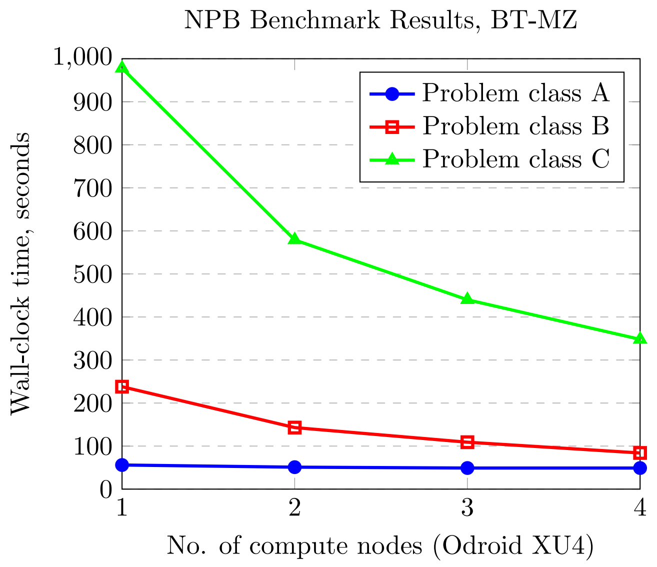


Figure 8.1: Graph displaying NPB test results in wall-clock time for problems of class A, B and C for 1 - 4 connected compute nodes. Notice especially that class A problems does not show any appreciable decrease in wall-clock time as the cluster grows.

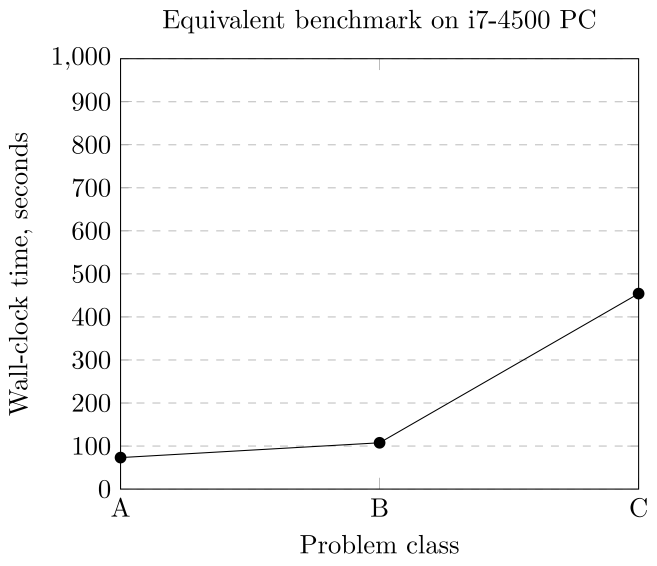


Figure 8.2: Graph displaying NPB tet results in wall-clock time for problems of class A, B and C for a i7-4500 PC.

The benchmarks were run with different problem classes, this simply corresponds to the size of the problem in terms of number of equivalently sized sub-problems. As we can see in the diagram, a larger problem class results in an exponential increase in wall-clock time.

While not much difference is apparent at all over different numbers of compute nodes for the A-class benchmark, there are significant reductions in time for B-class and C-class benchmarks - approximately 40% reduction in time when doubling the number of nodes.

Additionally, for all problem classes, at four compute nodes the Odroid-cluster performs faster than the PC. Interestingly, the cluster outperforms the PC for the A-class benchmark regardless of the number of nodes.

Chapter 9

Discussion & Conclusion

In this chapter we discuss the results of the project, the challenges faced, alternative solutions, and possibilities for future development of SCMT. Finally, we present our conclusions.

9.1 Alternative cluster architectures

SCMT is constructed in such a way that one node is always responsible for all cluster management, including keeping a database of nodes; running the DHCP server; running the file system server, etc. However, while this is a solution which has a major benefit in its simplicity, it is not suited for arbitrarily large clusters, as the workload on the master grows with the size of the cluster and eventually reaches saturation. Additionally, it creates a single point of failure - should the master node stop working, the whole cluster becomes unusable.

When constructing large clusters, it is thus desirable to distribute properties of the cluster management. For example, as discussed in section 5.1, Lustre is a distributed file-system intended for large clusters. Using such a solution for file sharing is one way to decrease the dependence on a single master node.

Distribution could also be done per service; one node could be responsible for the file-system, another could be run the DHCP server, etc. This also reduces the scalability limitation incurred by overloaded master nodes - however, each of the nodes running a vital service would still be a single points of failure.

One way to improve reliability and removing the single point of failure would be to have nodes backing up the master. For example, if one node is responsible for the file-system, another file-system backup node could store a copy of the shared file-system. Should the node responsible for the file-system fail, the file-system backup node could then seamlessly take over its responsibilities.

9.2 Node operating system

For our prototype, we chose to primarily support *Ubuntu* (15.04), since that is what new Odroid-devices are supplied with. While this is a workable solution, it might not be the best fit for a cluster - Ubuntu is designed for a user friendly desktop experience, not a high-performance computing environment.

Furthermore, a few years from now, there might be a different operating system supplied with Odroids (or, indeed, any other single-board computer to be used with SCMT), at that point, SCMT must either be updated to match the new situation, or users will have to flash the version of Ubuntu that SCMT expects on each node. This will be a hassle for the users and the developers and it would be desirable for SCMT to an as large extent as possible be independent of the Linux distribution in a reasonable extent.

9.3 Command invocation on nodes

SCMT executes commands on nodes via SSH. This was non-trivial, SSH is not intended for automated use, and there were some difficulties in implementing this functionality.

With this experience, and the possibility that users in the future will have to flash each node either way (as discussed in the previous section), an alternative solution might seem beneficial. Instead of relying on pre-installed software, a cluster management solution could supply its own image for installation on nodes, complete with operating system and a cluster management service. This removes the risk of errors caused by unexpected differences between nodes' pre-installed software. In addition, it provides a more robust solution for command invocation than the current SSH-based solution. However, it has the major drawback in requiring (non-trivial) manual setup of each node.

If, in the future, most single-board computers will have support for network boot, a more optimal solution could be achieved. Nodes which are connected to the cluster could boot the custom image over the network, which has all the benefits of flashing a custom image, combined with the complete plug-and-play functionality which so far is only possible by relying on pre-installed software. In addition, if network booting, nodes need not necessarily have their own storage, which may be desirable for some clusters (if nothing else, to reduce costs).

9.4 Task scheduling & workload management

We did not actively pursue any development in task scheduling, this is considered out of the scope of SCMT itself, and is in the domain of specific applications installed through the plugin system.

If SCMT is to be used for MPI jobs or as a Hadoop cluster, task scheduling will be managed by the runtimes distributed with those applications.

However, a cluster to be used by multiple users (or used for several different purposes) at once would benefit from some form workload management - a way of distributing the cluster's resources (that is, CPU time and memory, or whole subsets of the cluster) to the users in a fair manner.

SCMT does not support any form of automated workload management, but it could be implemented as a plugin providing an interface launching for jobs in a managed manner.

9.5 Munin or Ganglia

Both Ganglia and Munin provide a working monitoring solution but their internal architectures are different. While Munin is simple to setup and use, it does not handle large clusters as well as Ganglia does. Ganglia, on the other hand, is more complicated and requires more work to understand and set up. Munin's poll-based design is inflexible in the sense that it is not designed for an adjustable polling rate, whereas Ganglia's is configurable. This, combined with the fact that Munin's poll rate is only once every five minutes, forces the user to use data that is less accurate than what Ganglia offers.

We observed that Ganglia has significantly lower CPU usage on the master than Munin. The physical memory usage had a steeper incrementation in size on Munin in comparison with Ganglia. Which one to use depends on the requirements, but for large clusters, Ganglia will provide a suitable monitoring solution when considering performance.

9.5.1 Test results

The fact that the performance measurements were done only once every second meant that processes which completed in less than one second had the risk of not being detected and recorded. This in combination with the high input/output load on the master may have contributed to inaccurate results.

9.6 Benchmark results

The NAS Parallel Benchmarks showed promising results (see section 8.2). With as few as four Odroid XU4 devices, a reasonably performant x86-based PC could be outperformed. Four such devices are generally cheaper than any single PC, as such, the benchmark results show potential in using single-board computers for high-performance computing from an economical perspective.

An interesting aspect of the benchmark results is that there was no appreciable speed-up from running A-class benchmarks on more than one node. We speculate that A-class benchmarks are small enough that constant factors (e.g. nodes accessing the executable file over NFS) are significantly larger than the actual benchmarking job; that the job is small enough to achieve most of the potential gains from parallelism on a single node, or both.

Another observation on the results is that a single XU4 outperforms the PC on the A-class benchmark. For this, we can only speculate that something in particular about the test made it better suited for the XU4 than the PC in question, since we would otherwise expect the PC to perform better. To gain clarity in this, it would be appropriate to run the benchmarks on a variety of PCs.

9.7 Societal impact

As mentioned in the introduction of this report, there are several benefits to using single-board computers to construct high-performance computer clusters. Due to the lower economic cost compared with clusters of traditional servers, it furthers a democratisation of computer cluster usage, putting arbitrarily scalable computational power in the hands of the mainstream.

However, alternative solutions to this exist in the form of cloud services offering computational resources as a service. Still, hosting an own cluster is not irrelevant, as one may prefer to keep data within an organisation, and not rely on external providers and network connections.

Additionally, due to their lower power consumption, clustering single-board computers can provide large-scale computational power in a more environmentally friendly manner. With lower power consumption, it is reasonable to also assume that the single-board computers also generate less heat (although it would be relevant to do research on this subject), which means these clusters could be used in data centres without needing to spend as much money and electric power on cooling. This, of course, further helps reducing the environmental impact of high-performance computing.

9.8 Future development

There are many ways in which SCMT could be improved and extended, things we did not pursue due to lack of time, or because they fell outside the scope and limitations we set up for the projects. In this section we mention a few ideas for extensions and improvements.

- Improve the fault tolerance in plugin installation.
- Improve performance and scalability by parallelising as much as possible in order to reduce the setup time for each node, this would be especially usable in the setup of large clusters.
- Adding support for distributing the responsibilities of the master node over the cluster to improve scalability and to remove the single point of failure.
- Make SCMT work independently of which Linux or Unix version that is running on the master and the nodes.
- Further events to invoke scripts on, e.g. plugin uninstallation.

9.9 Conclusion

The purpose of our project was to explore methods for easily setting up and maintaining a cluster of single board computers. During the implementation of SCMT, we recieved first-hand experience of the need for a software like the one we created. When testing SCMT in the cluster we have had to re-flash devices several times and this has been time consuming.

The members of the project group had little experience in both writing Bash-scripts and programming in Go, which we all take with us as a learning experience.

Finally, and most importantly, we finished the SCMT prototype - and it works as intended, all the core functionality is present, even if it could be improved with more features and robustness. Thus, we can consider the project to be a success.

Bibliography

[1] D. A. Bader and R. Pennington, *Cluster computing: applications*, 2001. [Online]. Available: `http://www.cc.gatech.edu/~bader/papers/ijhpca.html`.

[2] P. Bougeault, "High performance computing and the progress of weather and climate forecasting," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5336 LNCS, p. 349, 2008, ISSN: 03029743. DOI: `10.1007/978-3-540-92859-1_31`.

[3] G. L. Valentini, W. Lassonde, S. U. Khan, *et al.*, "An overview of energy efficiency techniques in cluster computing systems," *Cluster Computing*, vol. 16, no. 1, pp. 3–15, 2013, ISSN: 13867857. DOI: `10.1007/s10586-011-0171-x`.

[4] Hardkernel co, *Odroid — hardkernel.* [Online]. Available: `http://www.hardkernel.com/main/products/prdt%7B%5C_%7Dinfo.php?g%7B%5C_%7Dcode=G143452239825` (visited on 05/11/2016).

[5] Raspberry Pi Foundation, *Raspberry pi 3 model b - raspberry pi.* [Online]. Available: `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/` (visited on 05/11/2016).

[6] J. J. Cusick, W. Miller, N. Laurita, and T. Pitt, "Design, construction, and use of a single board computer beowulf cluster: application of the small-footprint, low-cost, insignal 5420 octa board," Dec. 2014. arXiv: `1501.00039`. [Online]. Available: `http://arxiv.org/abs/1501.00039`.

[7] D. Abdurachmanov, P. Elmer, G. Eulisse, and S. Muzaffar, "Initial explorations of arm processors for scientific computing," *Journal of Physics: Conference Series*, vol. 523, no. 1, p. 12 009, 2014, ISSN: 17426596. DOI: `10.1088/1742-6596/523/1/012009`. arXiv: `arXiv:1311.0269v1`. [Online]. Available: `http://stacks.iop.org/1742-6596/523/i=1/a=012009`.

[8] B. J. Walker, "Introduction to single system image clustering," pp. 1–16, 2001.

[9] P. Abrahamsson, S. Helmer, N. Phaphoom, *et al.*, "Affordable and energy-efficient cloud computing clusters: the bolzano raspberry pi cloud cluster experiment," *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, vol. 2, pp. 170–175, 2013, ISSN: 23302186. DOI: `10.1109/CloudCom.2013.121`.

[10] P. M. Papadopoulos, M. J. Katz, and G. Bruno, "Npaci rocks: tools and techniques for easily deploying manageable linux clusters," in *Cluster Computing, 2001. Proceedings. 2001 IEEE International Conference on*, Newport Beach, CA, USA: IEEE, 2001, pp. 258–267, ISBN: 0769511163. DOI: `10.1109/CLUSTR.2001.959986`.

[11] T. G. Mattson, "High performance computing at intel: the oscar software solution stack for cluster computing," in *Proceedings - 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGrid 2001*, 2001, ISBN: 0769510108. DOI: `10.1109/CCGRID.2001.923170`.

[12] G. F. Pfister, "The varieties of single system image," in *Advances in Parallel and Distributed Systems, 1993., Proceedings of the IEEE Workshop on*, Princeton, NJ: IEEE, 1993, pp. 59–63, ISBN: 0818652500. DOI: `10.1109/APADS.1993.588768`.

[13] J. Harr and G. Denault, “Issues concerning linux clustering: cluster management and application porting,” in *Proceedings 16th International Parallel and Distributed Processing Symposium*, Ft. Lauderdale, FL: IEEE, 2002, 6 pp, ISBN: 0-7695-1573-8. DOI: 10.1109/IPDPS.2002.1015539. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1015539.

[14] V. A. Nguyen and S. Pierre, “Scalability of computer clusters,” *Electrical and Computer Engineering, 2001. Canadian conference on*, vol. 1, 405–409, vol. 1, 2001.

[15] B. Nowicki, Network Working Group, *nfs: network file system protocol specification*, 1989. [Online]. Available: https://tools.ietf.org/html/rfc1094 (visited on 05/12/2016).

[16] P. Scwhan, “Lustre: Building a file system for 1,000-node clusters,” in *Linux Symposium, Proceedings of the*, Ottawa, Ontario, Canada, 2003. [Online]. Available: https://www.kernel.org/doc/ols/2003/ols2003-pages-380-386.pdf.

[17] Internet Engineering Task Force (IETF), *network file system (nfs) version 4 minor version 1 protocol*, 2010. [Online]. Available: https://tools.ietf.org/html/rfc5661 (visited on 05/12/2016).

[18] S. Bigger, *Why use lustre - hpdd community space - hpdd community wiki*, 2011. [Online]. Available: https://wiki.hpdd.intel.com/display/PUB/Why+Use+Lustre (visited on 05/12/2016).

[19] B. ten Brinke, *Instant Munin Plugin Starter*. 2013, ISBN: 9781849696746.

[20] Bright Computing, *Monitoring*. [Online]. Available: http://www.brightcomputing.com/linux-cluster-monitoring (visited on 03/28/2016).

[21] Penguin Computing, *Scyld clusterware management software from penguin computing*. [Online]. Available: http://www.penguincomputing.com/products/software/cluster-management-scyld-clusterware/ (visited on 03/28/2016).

[22] eZ Server Monitor, *A lightweight and simple dashboard monitor for linux - ez server monitor*. [Online]. Available: http://ezservermonitor.com/ (visited on 03/28/2016).

[23] Nagios, *Nagios core - nagios*. [Online]. Available: https://www.nagios.com/products/nagios-core/ (visited on 03/28/2016).

[24] Supermon, *Supermon*. [Online]. Available: http://supermon.sourceforge.net/index.html (visited on 03/28/2016).

[25] T. Oetiker, *Rrdtool - about rrdtool*, 2014. [Online]. Available: http://oss.oetiker.ch/rrdtool/index.en.html (visited on 03/24/2016).

[26] M. L. Massie, B. N. Chun, and D. E. Culler, “The ganglia distributed monitoring system: design, implementation, and experience,” *Parallel Computing*, vol. 30, no. 7, p. 838, Jul. 2004, ISSN: 01678191. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819104000535.

[27] Munin-monitoring, *Munin*. [Online]. Available: http://munin-monitoring.org/.

[28] ——, *Munin's architecture*. [Online]. Available: http://guide.munin-monitoring.org/en/latest/architecture/ (visited on 02/22/2016).

[29] Ganglia, *Ganglia monitoring system*. [Online]. Available: http://ganglia.sourceforge.net/ (visited on 02/23/2016).

[30] M. Massie, B. Li, B. Nicholes, and V. Vuksam, *Monitoring with Ganglia*. O'Reilly Media, 2012, pp. 1–2, 2–4, ISBN: 9781449329709.

[31] Ganglia contribution repository (ganglia-contrib), *Ganglia-alerts*, 2012. [Online]. Available: https://github.com/ganglia/ganglia_contrib/blob/master/ganglia-alert/ganglia-alert (visited on 05/16/2016).

[32] S. Godard, *Sysstat*, 2016. [Online]. Available: `http://sebastien.godard.pagesperso-orange.fr/man_pidstat.html` (visited on 03/22/2016).

[33] D. Dauger and V. Decyk, "Plug-and-play cluster computing: high performance computing for the mainstream," *Computing in science & engineering*, vol. 7, no. 2, pp. 27–33, 2005. DOI: `10.1109/MCSE.2005.37`. [Online]. Available: `http://scitation.aip.org/content/aip/journal/cise/7/2/10.1109/MCSE.2005.37`.

[34] J. Squyres, *Open mpi powers 8 petaflops*, 2011. [Online]. Available: `http://blogs.cisco.com/performance/open-mpi-powers-8-petaflops` (visited on 05/13/2016).

[35] Argonne National Laboratory, *Mpich2: Performance and portability*. [Online]. Available: `http://www.cels.anl.gov/events/conferences/SC07/presentations/mpich2-flyer.pdf` (visited on 05/14/2016).

[36] MPICH project, *Mpich license*. [Online]. Available: `http://git.mpich.org/mpich.git/blob_plain/db40bb8aa2168ffad8f9a681723dff9f8d6c426c:/COPYRIGHT` (visited on 05/14/2016).

[37] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, May 2010, pp. 1–10. DOI: `10.1109/MSST.2010.5496972`.

[38] E. Sivaraman and R. Manickachezian, "High performance and fault tolerant distributed file system for big data storage and processing using hadoop," in *Intelligent Computing Applications (ICICA), 2014 International Conference on*, Mar. 2014, pp. 32–36. DOI: `10.1109/ICICA.2014.16`.

[39] MPICH wiki, *Hydra process management framework*, 2014. (visited on 05/30/2016).

[40] OpenMPI project, *Faq: Running jobs under rsh/ssh*, 2015. [Online]. Available: `https://www.open-mpi.org/faq/?category=rsh` (visited on 05/30/2016).

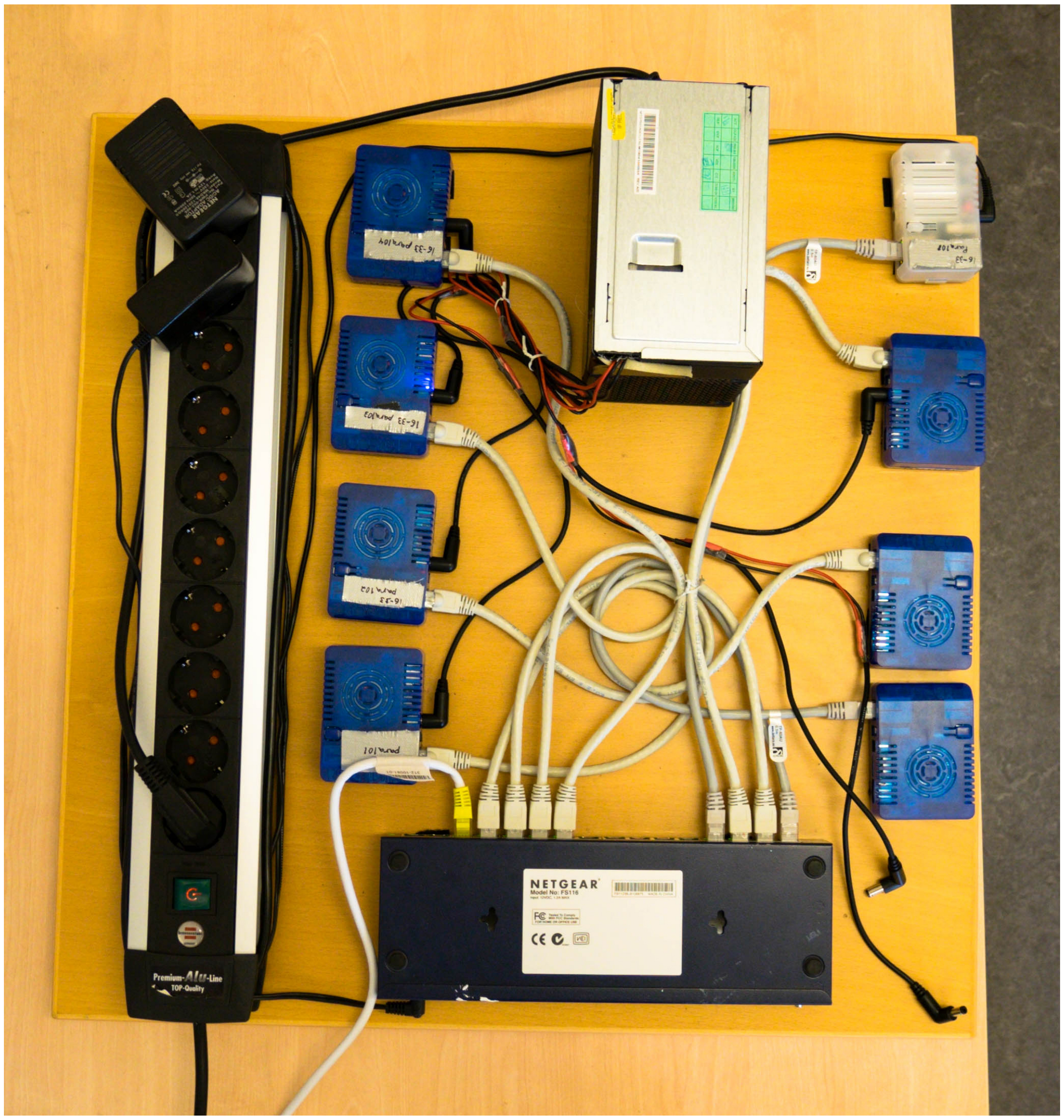
[41] D. Zburivsky and E. (.-b. collection), *Hadoop cluster deployment*, English, 1st ed. Birmingham: Packt Publishing, 2013, ISBN: 9781783281718.

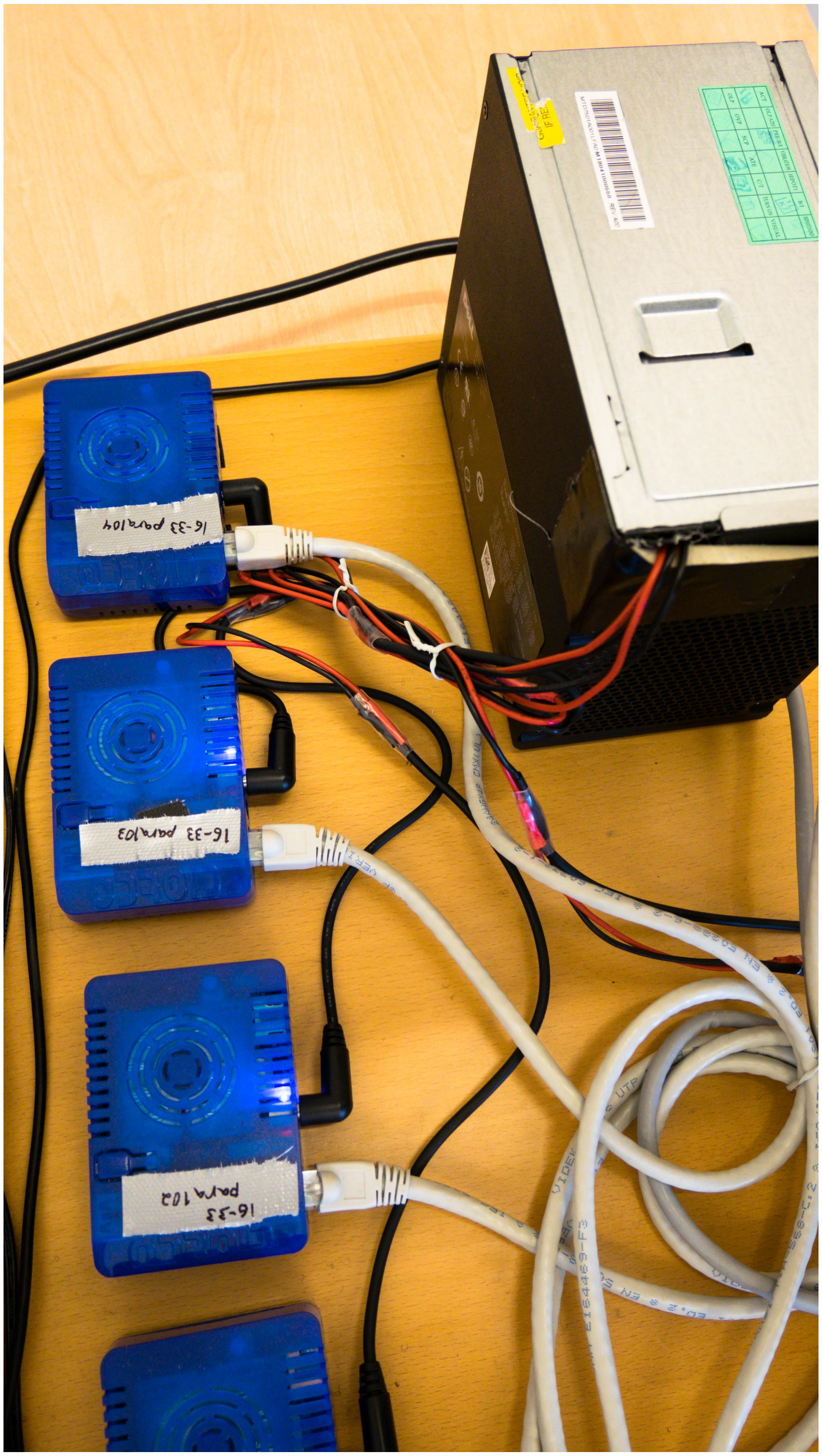
[42] H. Jin and R. F. Van Der Wijngaart, "Performance characteristics of the multi-zone nas parallel benchmarks," *Journal of Parallel and Distributed Computing*, vol. 66, no. 5, pp. 674–685, 2006, ISSN: 07437315. DOI: `10.1016/j.jpdc.2005.06.016`.

Appendices

Appendix A

Cluster pictures

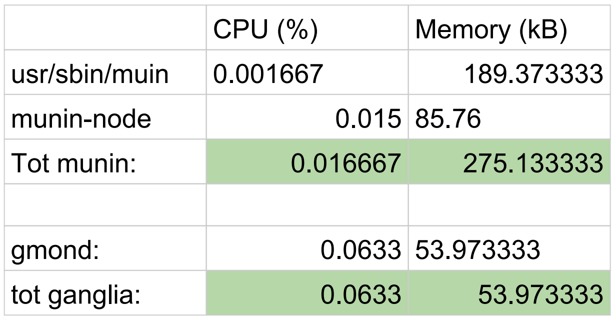



16-33 para104
16-33 para103
16-33 para102

NETGEAR

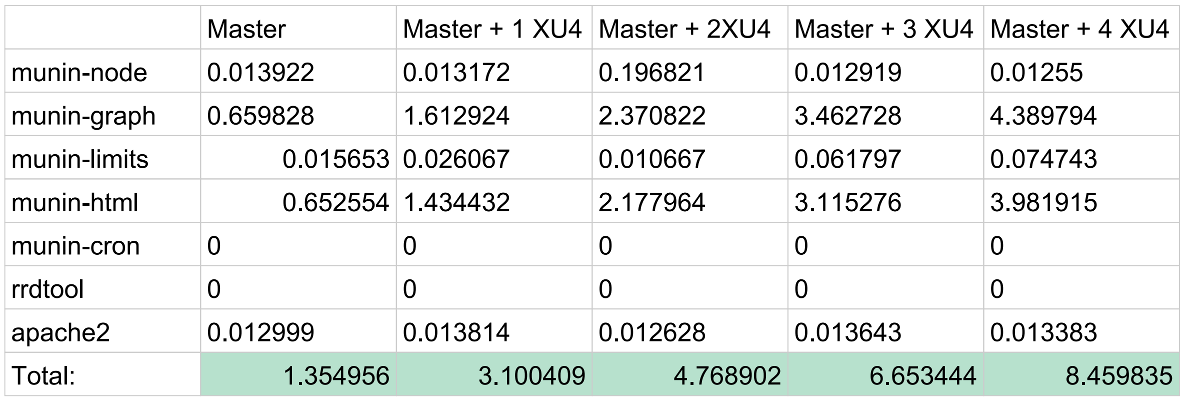
Appendix B

Raw test data, monitoring tests



	CPU (%)	Memory (kB)
usr/sbin/muin	0.001667	189.373333
munin-node	0.015	85.76
Tot munin:	0.016667	275.133333
gmond:	0.0633	53.973333
tot ganglia:	0.0633	53.973333

Figure B.1: Average usage of physical memory (in kB) and CPU (%) on an Odroid XU4 (compute node) for Munin and Ganglia.



	Master	Master + 1 XU4	Master + 2XU4	Master + 3 XU4	Master + 4 XU4
munin-node	0.013922	0.013172	0.196821	0.012919	0.01255
munin-graph	0.659828	1.612924	2.370822	3.462728	4.389794
munin-limits	0.015653	0.026067	0.010667	0.061797	0.074743
munin-html	0.652554	1.434432	2.177964	3.115276	3.981915
munin-cron	0	0	0	0	0
rrdtool	0	0	0	0	0
apache2	0.012999	0.013814	0.012628	0.013643	0.013383
Total:	1.354956	3.100409	4.768902	6.653444	8.459835

Figure B.2: Average CPU usage in percent on the master node for all Munin processes and different sized clusters.

	Master	Master + 1 XU4	Master + 2 XU4	Master + 3 XU4	Master + 4 XU4
gmond	0.325847	0.310342	0.341233	0.354499	0.475075
gmetad	0.9199	0.979753	1.060586	1.175832	1.213201
rrdtool	0	0	0	0	0
apache2	0.011996	0.012496	0.014743	0.014232	0.01499
Total:	1.257743	1.302591	1.416562	1.544563	1.703266

Figure B.3: Average CPU usage in percent on the master node for all Ganglia processes and different sized clusters.

	Master	Master + 1 XU4	Master + 2XU4	Master + 3 XU4	Master + 4 XU4
munin-node	124.525	125.125	125.083333	133.422222	102.612222
munin-graph	330.713333	779.159444	1311.666111	2876.461111	3217.210556
munin-limits	14.31	12.664444	10.351111	34.335556	36.017222
munin-html	398.786111	1309.192222	2711.485	4459.687222	6976.736667
munin-cron	0	0	0	0	0
rrdtool	0	0	0	0	0
apache2	0	0	104.8	0	9.069444
Total:	868.334444	2226.14111	4263.385555	7370.483889	10239.03389

Figure B.4: Average physical memory usage in kB on the master node for all Munin processes and different sized clusters.

	Master	Master + 1 XU4	Master + 2 XU4	Master + 3 XU4	Master + 4 XU4
gmond	2221.049444	2877.766111	3949.233333	4426.495	6527.565
gmetad	1798.386111	1880.379444	1869.610556	1953.861667	2145.573333
rrdtool	0	0	0	0	0
apache2	0.011996	0.012496	0.014743	0.014232	0.01499
Total:	4019.447551	4758.158051	5818.858632	6380.370899	8673.153323

Figure B.5: Average physical memory usage in kB on the master node for all Ganglia processes and different sized clusters.

Appendix C

Raw test data, NPB

NPB benchmark results, BT-MZ, MPI
On cluster, 4 processes per node.
Test 1:
Class A, 1 nodes: Time in seconds = 56.99
Class A, 2 nodes: Time in seconds = 52.17
Class A, 3 nodes: Time in seconds = 49.99
Class A, 4 nodes: Time in seconds = 49.75
Class B, 1 nodes: Time in seconds = 241.43
Class B, 2 nodes: Time in seconds = 146.13
Class B, 3 nodes: Time in seconds = 108.98
Class B, 4 nodes: Time in seconds = 84.89
Class C, 1 nodes: Time in seconds = 976.29
Class C, 2 nodes: Time in seconds = 581.50
Class C, 3 nodes: Time in seconds = 446.22
Class C, 4 nodes: Time in seconds = 350.51

Test 2:
Class A, 1 nodes: Time in seconds = 56.41
Class A, 2 nodes: Time in seconds = 51.26
Class A, 3 nodes: Time in seconds = 48.96
Class A, 4 nodes: Time in seconds = 48.95
Class B, 1 nodes: Time in seconds = 237.59
Class B, 2 nodes: Time in seconds = 143.29
Class B, 3 nodes: Time in seconds = 109.10
Class B, 4 nodes: Time in seconds = 84.22
Class C, 1 nodes: Time in seconds = 977.78
Class C, 2 nodes: Time in seconds = 578.99
Class C, 3 nodes: Time in seconds = 439.71
Class C, 4 nodes: Time in seconds = 348.37

NPB benchmark results, BT-MZ, MPI
On laptop i7-4500, 4 x Core, 2700MHz

Class A: Time in seconds = 73.20
Class B: Time in seconds = 107.59
Class C: Time in seconds = 454.22