GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Set Up a Test Cluster with Slurm

David Nelles

MatrNr: 24609157

Supervisors:

Vanessa End & Timon Vogt

Georg-August-Universität Göttingen
Institute of Computer Science

March 31, 2023

Abstract

When maintaining a cluster, already short stalling periods can cause huge costs. Nonetheless, the cluster needs to be updated sometimes. To keep the stalling time short, it is important to prevent failures during the update process. Therefore, it is sensible to simulate the update process, before updating the cluster. In order to do that, a test cluster has to be designed. A test cluster is also useful to simulate other operations like infrastructure changes and new features. It can also be used to reproduce and debug errors that occur on the original cluster. The test cluster needs to fit special requirements. For example, it must be energy efficient and do not need much compute power. When setting up the cluster, the hardware needs to be set up first. In order to fulfill the requirements, the test cluster uses virtual machines, that emulates the hardware. After setting up the hardware, software such as SLURM and Munge, which manage the cluster, must be configured. To enable accounting, additional software like a SQL-Database needs to be set up.

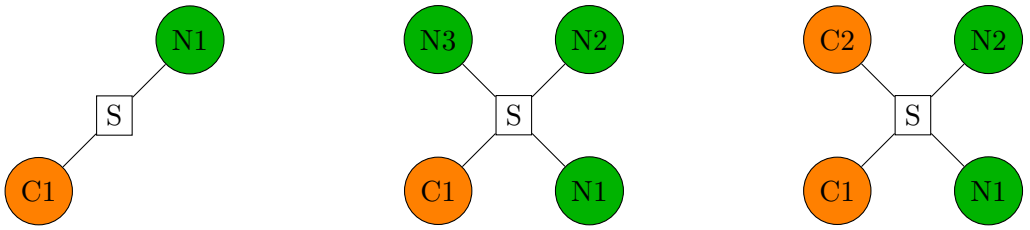
Acronyms

CPU	Central Processing Unit
DBMS	Database Management System
DNF	*Dandified YUM*
GWDG	*Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen*
NFS	Network File System
SLURM	*Simple Linux Utility for Resource Management*
SQL	Structured Query Language

1. Introduction

To reduce the risk of failures while updating a cluster, the update process should be simulated. Therefor, a test cluster on which the simulation is executed must be developed. A cluster is a collection of multiple computers connected by a network that appear as a single system. Each computer of the cluster is called a node. There are two different cluster types. High availability clusters increase the reliability of certain applications, while high performance clusters increase the computational power of the computing system. This report deals with the implementation of *Simple Linux Utility for Resource Management* [20] (*SLURM*) on the test cluster. *SLURM* is a workload management tool used on high performance cluster to split and distribute the workload among different compute nodes. Therefor, *SLURM* performs the querying of multiple tasks, as well as the distribution of subtasks between multiple nodes.

SLURM distinguishes between two different node types, which are connected to each other as shown in Fig. 1. Control nodes (orange) are responsible for receiving the tasks and distributing them to the compute nodes (green). Those are responsible for the execution of the subtasks that are assigned to them. *SLURM* provides different services for control and compute nodes. A service called *slurmctld* is executed on the control nodes, while each compute node executes the *slurmd* service. Clusters managed with *SLURM* must have at least one and at most two control nodes. Additionally, there must be at least one compute node, leading to a minimal setup as



(a) Minimal setup (b) Single control node (c) Double control node

Figure 1: SLURM topologies

shown in Fig. 1a. However, since this setup does not provide any benefits toward operating the compute node without a control node, it is more reasonable to connect multiple compute nodes with a single control node as shown in Fig. 1b. This setup is fully functional and enables reasonable workload distributions. A setup with two control nodes as depicted in Fig. 1c does not provide any performance benefits, but reliability. In those setups, the second control node acts as backup, in case the first control node crashes.

Since operating a cluster is very expensive, most clusters are used by different institutions or at least for different projects. For accounting purposes, it is important to record who used the cluster when, for how long, and for what. The accounting data is gathered by the control node and stored into a database, which is connected to *SLURM* through a third service named *slrumdbd*. Since accounting is crucial to the operation of the cluster and can be easily affected by changes to the cluster, the test cluster needs to model the accounting mechanism as well as the workload management system.

2. Prerequisites

The original cluster uses multiple nodes on which Rocky Linux [9] is installed. On the original cluster, *SLURM* is used as workload manager. The authentication is controlled by the authentication service *munge* [3]. Since *SLURM* is used as workload manager, the *slurmdbd* service is used to populate the accounting data, to an Structured Query Language (SQL) database, where it is stored persistently.

The more similarities there are between the original cluster and the test cluster, the more accurate the simulation. Because of that, the test cluster should have the same specifications. On the other hand, maintaining the test cluster must be much less expensive than maintaining the original cluster. To achieve this, the test cluster cannot be identical to the original cluster. For this use case, the test cluster does not need to consist of as many nodes as the original cluster. Instead, the test cluster uses one control node and two compute nodes. Since the test cluster is not used to process high performance applications, the compute power of the test cluster is not important. Additionally, the test cluster can be turned off most of the time. It only needs to run in case somebody wants to test something.

Because of these requirements, multiple virtual machines were used as nodes. This not only has the advantage that they can be easily switched on and off, it is also easy to back up and restore virtual machines. This allows us to easily restore the test cluster if it is completely destroyed during the update process. In order to host the virtual machines, the *OpenStack* [12] service provided by the *Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen* (*GWDG*) [6] is used.

3. System setup

In order to set up the test cluster, the virtual machines had to be set up first. After that, the dependencies of *SLURM* could be installed. In the end, *SLURM* and the accounting database were set up.

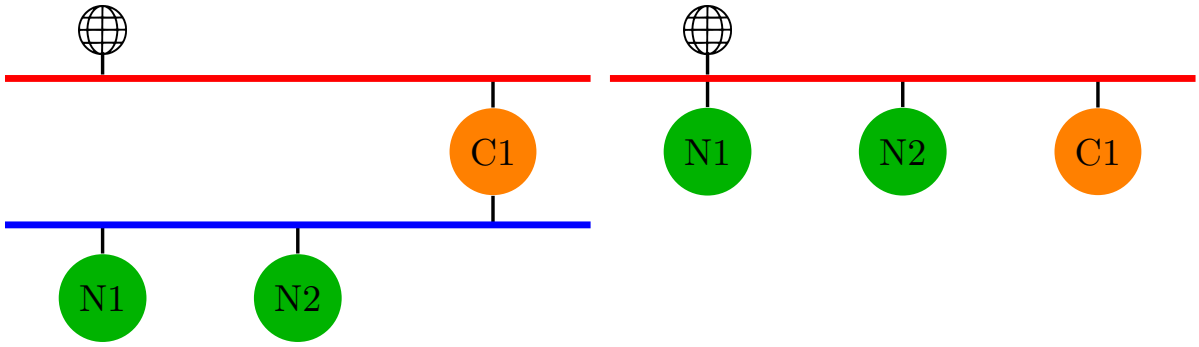
3.1. Virtual machines

First, three different virtual machines had been generated. For this purpose, three new instances have been created after logging into the *OpenStack* web interface. As already mentioned in Section 2, the compute power of the nodes is not important. Therefore, the least powerful configuration was chosen for each instance. All instances were configured as shown in Table 1.

RAM	2GB
VCPUs	2
Disk space	20GB
OS	Rocky Linux 8.7

Table 1: Virtual machine specifications

Since the nodes need to communicate with each other, it is important to set up a network between the instances. As shown in Fig. 2a, normally only the control nodes are connected to the internet (red), while the compute nodes are on a separate network (blue) to which only the control node has access. This topology is used for security reasons and guarantees that no one logs on to the compute nodes. Since the test cluster is only used by people who know what they are doing and attacks on the test cluster are not critical, the test cluster does not need to protect the compute nodes. The cluster setup is much easier if all nodes are connected to the internet. Because of that, the network topology depicted in Fig. 2b was configured for the test cluster.



(a) Regular network topology of clusters (b) Network topology of the test cluster

Figure 2: Network topologies regular clusters and the test cluster

3.2. Munge

The first software that needs to be set up is *munge*. *Munge* is an authentication service that is used to securely forward authentication data to remote systems. Users should be able to run processes on the compute nodes by logging in to the control node. Nevertheless, the user ID and the group ID of the processes running on the compute nodes should correspond to the user who started the execution on the control node. *Munge* is used to securely forward the login information to the compute nodes. Because of this, the users do not have to manually log in to each compute node. [3]

Munge needs to be installed on each node. The installation can be easily performed with a package manager. To communicate between the nodes, all nodes need the same `mungekey`. This key can be generated via the *create-munge-key* shell script, which is part of the package. The shell script generates a key file at `/etc/munge/`. After copying the key file to each node, the *munge* service is enabled as described in the documentation [8].

3.3. Network file system

In order to run *SLURM*, it is important that the configuration of all nodes fit to each other. To guarantee that the configuration is the same for each node, the configuration is stored on a Network File System (NFS). By mounting the NFS on all nodes, the identical configuration file can be used for all nodes. Additionally, the *SLURM* binary files can be stored on the NFS. This simplifies updating *SLURM* and guarantees that the same version is used for every node.

The NFS of the test cluster should be hosted on the control node. Thus, the next step was to set up NFS. Therefor, *nfs-utils* [1] was installed on each node. First, the NFS server was set up. For this purpose, the directories where the files will be stored have been created. The directory tree depicted in Fig. 3 consists of at least three directories and one soft link. The `etc` directory contains the configuration files which are used to configure *SLURM*. In order to debug and protocol failures, *SLURM* creates some log files. Those files are created inside the `log` directory. Additionally, the NFS contains one directory for each *SLURM* version that has been installed on the cluster. This allows reverting the *SLURM* version in case something goes wrong. A soft link named `current` points to the version directory, that contains the currently used *SLURM* version. Each version directory contains a `source` directory and an `install` directory. The `source` directory contains the source files of the specific version. While the `install` directory contains the binary files that are used to execute *SLURM*.

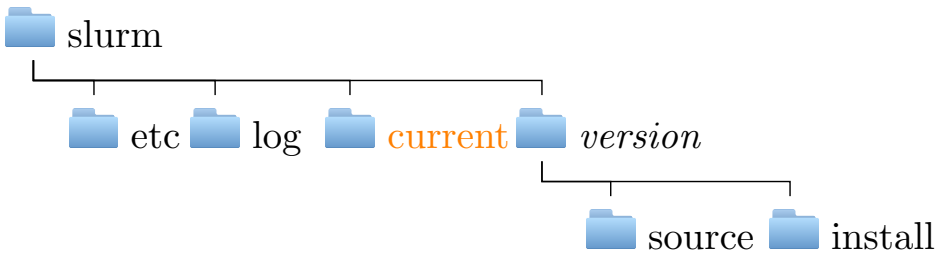


Figure 3: NFS directory structure

NFS is configured via a file located at `/etc/exports`. To share a directory, a new line, starting with the directory path, must be inserted into the configuration file. After the directory path, the IP address of the client to be granted access must be specified. Additionally, the permissions

of the clients are written behind each client IP. The configuration used for the test cluster is shown in Listing 1.

```
# direcotry                client-ip  config   client-ip  config
  /media/nfs               10.254.1.5(rw,sync) 10.254.1.11(rw,sync)
```

Listing 1: Example configuration of NFS

Since the clients of the test cluster shall write the log files to the NFS, each client had to be granted read and write access. The NFS server was activated by starting the corresponding `nfs-server` service via *systemd*. Since the NFS server shall start automatically when rebooting the control node, the service was also `enabled`.

The clients just have to mount the NFS. In order to mount the shared directories on each startup automatically, the line presented in Listing 2 was added to the *fstab* file of the compute nodes. For that, it is important that *nfs-utils* is installed on each client.

```
10.254.1.6:/media/nfs    /mnt/nfs        nfs4    defaults,user,exec ↩
        0 0
```

Listing 2: Content of the fstab file that enables mounting the NFS

3.4. SLURM

After setting up all dependencies, *SLURM* can be set up. To guarantee the best performance, *SLURM* is compiled on the system where it is executed later on.

3.4.1. Compiling

To compile *SLURM* an *C-compiler*, *python3* and *perl* need to be installed first. After that, the source code [16] is downloaded and unpacked. The *SLURM* compilation is done using the build management tool *GNU Make* [5]. In order to use *GNU Make*, a *Makefile* is needed. Unlike other applications, the *SLURM* source code does not contain a *Makefile*. The *Makefile* need to be generated individually on each system. That is because some settings influencing the compilation need to be specified beforehand. A configuration script helps to configure the build environment and generates the *Makefile*. This script is part of the source code and needs to be executed as superuser. To generate a functional *SLURM* application, the `prefix` and the `sysconfdir` settings must be specified. When compiling *SLURM* on the test cluster, the following settings were used:

prefix The path where the *SLURM* installation should be placed.
`/mnt/nfs/slurm/22.05/install`

sysconfdir The path where the *SLURM* configuration files should be placed afterward.
`/mnt/nfs/slurm/etc`

After generating the make file, *SLURM* can be compiled by executing `make`. Subsequently, *SLURM* is installed into the `sysconfdir` by running `make install`.

3.4.2. Configuration

Before executing *SLURM*, the installation needs to be configured. As suggested in the installation guide [17] the online configuration tool [13] was used to configure the *SLURM* on the test cluster. For many of the settings, the presets could be used. Table 2 shows the settings which had to be changed in order to run *SLURM*.

Setting	Description	Value
SlurmctldHost	Host name of the system, used as control node	n1
BackupController	Host name of the system, used as backup control node	<empty>
NodeName	Host names of the systems, used as compute nodes	n2-[1-2]
CPUs	Number of hardware threads of each system	2

Table 2: SLURM settings

For debugging purposes, it is useful to additionally configure the location of the log files. Otherwise, no log will be created. Therefor, the `SlurmdLogFile` and `SlurmctldLogFile` settings must be specified. The test cluster, was configured in a way that all log files are created inside the `nfs/slurm/log` directory. While the control node writes the log to the `slurmctld.log` file, the compute nodes write teir logs to the `slurmd.log` file. The log level can be set separately for the compute nodes (`SlurmdDebug`) and the control nodes (`SlurmctldDebug`). It turned out that log level `info` is sufficient for most debugging. Because of that, the `info` level was used for both, compute and control nodes.

The online configuration tool generates a configuration file. This file needs to be copied into the `sysconfdir` specified while configuring the build environment (`/mnt/nfs/slurm/etc`). After successfully configuring *SLURM*, the basic setup is finished.

In order to execute *SLURM*, a service must be executed on each node. The control node, must execute the *slurmctld* service while the compute nodes have to run the *slurmd* service. As already briefly mentioned in Section 3.3, the test cluster uses *systemd* [19] to control services. *Systemd* uses service files which define how to execute a service. It expects those service files to be located in `/usr/lib/systemd/system`. The *SLURM* source code contains the service files for each of the *SLURM* services. But when installing *SLURM*, those service files are not moved. There are multiple ways to place the *SLURM* service files in the correct directory. On the test cluster soft links referencing to the service files located in the source code directory were created. A positive side effect of this solution is that the service files are also updated when *SLURM* is updated. To achieve this effect, it is important to refer to the service files, using the soft link referring to the latest *SLURM* installation (`/mnt/nfs/slurm/current`).

3.5. Accounting

To enable accounting, some additional tools and configurations are needed. In order to store the accounting data, an SQL database must be set up. Since *SLURM* uses the InnoDB storage engine to enable rollbacks, it is important that the Database Management System (DBMS) supports *InnoDB* [14]. The DBMS can be installed via the default package manager. On the test cluster, `MariaDB` [4] was installed.

The connection between *SLURM* and the DBMS is handled by a service called *slrumdbd*. *Slurmdbd* is automatically generated while compiling *SLURM*. However, it is important that the development libraries of the DBMS are installed before the build environment is configured.

After successfully compiling *slurmdbd*, the accounting must be configured. This is done via a separate configuration file. As with the *SLURM* configuration file, the accounting configuration file must be located in the `sysconfdir` (`/mnt/nfs/slurm/etc`) specified while configuring the build environment. Table 3 shows the settings, their meanings and the values used by the test cluster [18]. Additionally, some accounting settings have to be specified inside the *SLURM* configuration file. Those settings and their purpose are shown in Table 4. The last column again shows the values used by the test cluster. The accounting mechanism is activated by selecting any other option than "none" for the `JobAcctGatherType`.

Setting	Description	Value
DdbHost	Host name of the database host	n1
StorageType	Defines the accounting storage mechanism type	accounting_storagemysql
StorageLoc	Database name where to store the accounting data	slurm_acct_db
StorageUser	User to log in to the database	password
StoragePass	Password fitting to the StorageUser	slurm
StoragePort	Port which is used to communicate between *slrumdbd* and the database	3306
LogFile	File in which the *slrumdbd* log should be written	<log dir>/slurmdbd.log

Table 3: Accounting settings

Setting	Description	Value
JobAcctGatherType	Defines what information should be gathered	linux
AccountingStorageType	The software that is used to store the information	SlurmDBD
AccountingStorageHost	Host name of the database host	n1
AccountingStoragePort	Port used to communicate with the database	3306

Table 4: Accounting settings inside the SLURM configuration

4. Validation

For the test cluster, it is important that the *SLURM* setup works as expected and the database connection is valid. The performance of the test cluster, on the other hand, is irrelevant. For this reason, I investigated whether all nodes are interconnected and parallel execution of programs on multiple nodes is successful. But computationally intensive programs were not executed.

First, I checked whether the controller detects all compute nodes and if they were running. A small *SLURM* tool called *sinfo* displays this information. The output presented in Listing 3 shows that both compute nodes `n2-1` and `n2-2` were idling. This means, that they did not execute something but were available.

```
$ sinfo

PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug*       up    2:00:00      2   idle n2-[1-2]
```

Listing 3: Result of the *sinfo* execution

To verify that *sinfo* would indicate that a node is unavailable, I disabled a node and rechecked the output. The output presented in Listing 4 shows that *sinfo* indeed indicates if the compute nodes are available or not. If a node is not available, the `STATE` column indicates that by displaying the value `down`.

```
$ sinfo

PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug*       up    2:00:00      1  down* n2-2
debug*       up    2:00:00      1   idle n2-1
```

Listing 4: Result of the *sinfo* execution with a deactivated compute node

To check if the control node distributes the workload between all compute nodes, I wrote a shell script executing *hostname* a few times. The advantage of executing *hostname* is that it returns the name of the host where it is executed. Because of that, it is possible to see what nodes executed the bash script by analyzing the generated output. When writing the shell script, it is important to indicate that two nodes can be used to execute the different tasks. This is done by adding the line `#SBATCH -N 2` to the start of the script. Each task, that should be executed in parallel, must be executed via `srun`. Additionally, it is important to mark that each task is only executed on a single node. The shell script can be found in Appendix A.1. To execute it via *SLURM*, the `sbash` command is used. The output of the execution that is presented in Appendix A.2 shows that the first two executions were transmitted to both compute nodes first. The four remaining tasks could not be transmitted to any compute node since both nodes were busy. After completing the first two tasks, the next two tasks were transmitted and so on. However, since the output of the executions contains both host names (`n2-1.novalocal` and `n2-2.novalocal`) it is certain that *SLURM* distributed the workload between all compute nodes.

After executing the shell script (Appendix A.1), the database should have recorded the execution. To check if the accounting mechanism works, I started an SQL client and connected to the database which was created in Section 3.5 (`slurm_acct_db`). I then reviewed the entries stored in the table named `cluster_job_table`. As expected, the job events were stored in this table. Appendix B shows a section of the table. Further analysis, showed that the `account` column of the table contained `NULL` for each job-event. This would be an issue in case the computing time is charged. That was because, the jobs were submitted via the default account. However, adding further certain account to the cluster, would solve this issue, but was not part of this project.

One goal of using virtual machines is the possibility to turn the test cluster off if it is not used and to turn it on when it is needed. Turning the test cluster off is no problem. Much more important is to find out if the cluster can be booted with small effort. To check this, I shut down all nodes of the test cluster and rebooted them after a few minutes. After that, the cluster did not work

as expected. The compute nodes did not start the *slurmd* service. It turned out that the reason for that was, that the `slurmd.service` file was missing while the *systemd*-daemon was started. Probably the reason for that was, that the *systemd* daemon was started before the NFS was mounted. To start the *slurmd* service afterward, the *systemd* daemon needs to be reloaded (`sudo systemctl daemon-reload`). After that, the *slurmd* service can be started manually. Due to the delayed start of the compute nodes, the controller did not detect that the compute nodes were running and marked them as `down`. In order to reset their state, the *SLURM* user needs to override the states manually by executing `scontrol update nodename=<note-name> state=idle`.

5. Challenges

While setting up the test cluster, some issues came up. The first issue was that *SLURM* could not access the `/var/spool/slurmd` directory. Because of that, the *SLURM* services did not even start. The reason was that the user `slurm` was not allowed to access this directory. I was able to solve this issue by making the `slurm`-user owner of this directory.

Another issue preventing the *SLURM* controller service (`slurmctld`) from starting was a missing `MailProg`. This is because there is no standard mail program on Rocky Linux, which is supported by *SLURM*. I could solve this issue by installing the mail program *mailx* [7].

The last issue preventing the start of the controller service was, that *SLURM* was not able to find a plugin called `cred/munge`. It turned out that this plugin is normally compiled together with *SLURM*. But that was not the case when I compiled *SLURM* first. The reason was that some *munge* libraries did not exist on the system, while compiling *SLURM*. These missing libraries are part of the `munge-libs` and `munge-devel` packages. Installing the `munge-libs` package was easily done via the package manager *Dandified YUM* (*DNF*). But installing `munge-devel` was a bit more complicated. By analyzing the "Automatic SLURM Build Script" published by the NI SP GmbH [11], I found that `munge-devel` is part of the `PowerTools` repository. This repository is not used by default and needs to be enabled separately. The installation is done with the command `dnf --enablerepo=powertools install munge-devel`.

Although the control service started after solving the previous issues *SLURM* did not work. The compute services (`slurmd`) did not start. One reason was that the `prefix` that was stated while configuring the build environment was incorrect. Since I built *SLURM* on the system where the NFS was hosted, I specified the directory where the NFS directories were located (`/media/nfs/slurm/install`). But this directory did not exist on the other nodes. The other nodes mount the NFS at `/mtn/nfs`. To fix this, I used `/mnt/nfs/slurm/install` as `prefix` and created a soft link (`/mnt/nfs`) referring to `/media/nfs`.

After correcting the location, where to search for *slurmd*, the service still did not start successfully. Additionally, no log file was created. This was because the compute nodes did not have permissions to write to the NFS. This was a problem since the log files should be written to the `nfs/log` directory. The reason for that was not an issue of the *SLURM* configuration, but of how the NFS is mounted. When mounting the NFS via the `fstab` file, the mounting process is executed by the user `root`. But the NFS host prevents `root` from writing to the NFS directories [10]. I was able to solve this issue by adding `no_root_squash` to the NFS configuration.

Now that the log files were written, the reason why the *slurmd* service did not start could be analyzed. The log showed that the configuration of the compute nodes was wrong. In the configuration file, I stated that the compute nodes have one Central Processing Unit (CPU). I chose this value, since all the virtual machines the test cluster consists of only contain a single virtual CPU. But, the `VCPUs` setting does not care about the number of processors, but about the number of hardware threads. The issue was solved after stating the number of hardware thread (2).

Additionally, the start of *slurmd* failed because a certain *cgroups* namespace was not mounted. The reason for that was, that *cgroup v2* was not installed but *cgroup v1*. In order to automatically mount the namespace with *cgroup v1*, some further settings need to be specified. *Cgroup* related settings are configured in a separate file named `cgroup.conf`. This file must be located next to the `slurm.conf` configuration file (`nfs/slurm/etc`). The automatic mount is activated by setting `CgroupAutomount=yes`. [15]

While configuring the accounting database, some further issues came up. One of those issues was that *SLURM* could not find a certain plugin called `accounting_storage/mysql`. From the previous issue, where *SLURM* did not find the `cred/munge` plugin, I knew that those issues occur in case some development libraries were missing during the build process. I installed the `MariaDB-devel` package and recompiled *SLURM*, which actually solved the problem.

During execution, the *slurmdbd* service generated some errors regarding not recommended values of `innodb_buffer_pool_size` and `innodb_lock_wait_timeout`. Without knowing the reason for these errors, I was able to fix them by choosing identical values for the `AccountingStoragePort` setting inside the *SLURM* configuration file and the `DbdPort` setting inside the `slurmdbd` configuration file. In addition, I defined the `DbdHost` and `StorageHost` settings inside the `slurmdbd` configuration.

6. Conclusion and Outlook

During the setup, several issues like wrong configurations, missing dependencies and incorrect file locations came up. But after all, the log files provided enough information to solve all the issues, and the test cluster was successfully set up. After the setup, the test cluster it is able to execute small programs on multiple nodes, and the accounting mechanism captures all interactions. All steps needed to set up the test cluster were documented in a markdown file stored at the *HedgeDoc* [2] service hosted by the *GWDG*. In case the setup must be done again, the documentation may help to reproduce the setup. However, it will not cover the entire setup, as most of the applications used in the setup are still being developed and will change over time.

During the validation, I could show that the test cluster can be switched on and off. However, in order to make the cluster working, the boot process requires more manual actions than expected. I therefore suggest writing some bash scripts that care about executing the `slurmd` service after booting the nodes and reset the node's states automatically. Additionally, the backup and restore process was not implemented. Before using the test cluster for simulations, this process should be implemented. Additionally, the test cluster only uses one control node. Since the original cluster uses two control nodes, I recommend extending the test cluster by one further control node in the future. After simulating an update process with the test cluster, it needs to be tested whether the cluster still works as expected. Therefor, some validation steps described in Section 4 need to be done over and over again. It might be useful to automate this validation step in the future.

References

[1] *Center for Information Technology Integration*. URL: http://www.citi.umich.edu/projects/nfsv4/linux/ (visited on 01/18/2023).

[2] The HedgeDoc developers. *HedgeDoc - The best platform to write and share markdown.* en-us. URL: https://hedgedoc.org/ (visited on 03/31/2023).

[3] dun. *MUNGE*. URL: https://dun.github.io/munge/ (visited on 01/18/2023).

[4] MariaDB Foundation. *MariaDB Server: The open source relational database*. en-US. URL: https://mariadb.org/ (visited on 01/18/2023).

[5] Free Software Foundation, Inc. *Make - GNU Project - Free Software Foundation*. URL: https://www.gnu.org/software/make/ (visited on 03/29/2023).

[6] *GWDG Cloud Server - GWDG - IT in der Wissenschaft*. URL: https://www.gwdg.de/server-services/gwdg-cloud-server (visited on 01/18/2023).

[7] *Heirloom mailx*. URL: https://heirloom.sourceforge.net/mailx.html (visited on 03/28/2023).

[8] *Installation Guide · dun/munge Wiki*. en. URL: https://github.com/dun/munge (visited on 01/19/2023).

[9] *It all started with a blog comment. — Rocky Linux*. en. URL: https://rockylinux.org/about (visited on 01/18/2023).

[10] Shrikant Lavhate. *Access denied error in NFS for root account*. en-US. Nov. 2017. URL: https://35.173.37.49/troubleshooting/access-denied-error-in-nfs-for-root-account/ (visited on 03/28/2023).

[11] NI SP GmbH. *Automatic SLURM Build and Installation Script*. en-US. URL: https://www.ni-sp.com/slurm-build-script-and-container-commercial-support/ (visited on 03/28/2023).

[12] *OpenStack Open Source Cloud Computing Infrastructure*. URL: https://www.openstack.org/ (visited on 03/29/2023).

[13] *Slurm System Configuration Tool*. URL: https://slurm.schedmd.com/configurator.html (visited on 01/20/2023).

[14] *Slurm Workload Manager - Accounting and Resource Limits*. URL: https://slurm.schedmd.com/accounting.html (visited on 03/16/2023).

[15] *Slurm Workload Manager - cgroup.conf*. URL: https://slurm.schedmd.com/cgroup.conf.html (visited on 03/29/2023).

[16] *Slurm Workload Manager - Download Slurm*. URL: https://slurm.schedmd.com/download.html (visited on 01/19/2023).

[17] *Slurm Workload Manager - Quick Start Administrator Guide*. URL: https://slurm.schedmd.com/quickstart_admin.html (visited on 01/20/2023).

[18] *Slurm Workload Manager - slurmdbd.conf*. URL: https://slurm.schedmd.com/slurmdbd.conf.html (visited on 03/16/2023).

[19] *System and Service Manager*. URL: https://systemd.io/ (visited on 03/18/2023).

[20] Andy B. Yoo, Morris A. Jette, and Mark Grondona. "SLURM: Simple Linux Utility for Resource Management". en. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 44–60. ISBN: 978-3-540-39727-4. DOI: 10.1007/10968987_3.

A. The hostname shell script

A.1. Shell script

```
#!/bin/bash

#SBATCH -N 2

/mnt/nfs/slurm/current/install/bin/srun --ntasks=1 hostname &
/mnt/nfs/slurm/current/install/bin/srun --ntasks=1 hostname &
/mnt/nfs/slurm/current/install/bin/srun --ntasks=1 hostname &
/mnt/nfs/slurm/current/install/bin/srun --ntasks=1 hostname &
/mnt/nfs/slurm/current/install/bin/srun --ntasks=1 hostname &
/mnt/nfs/slurm/current/install/bin/srun --ntasks=1 hostname &

wait
```

A.2. Output

```
n2-1.novalocal
n2-2.novalocal
srun: Job 99 step creation temporarily disabled, retrying (↩
   Requested nodes are busy)
srun: Job 99 step creation temporarily disabled, retrying (↩
   Requested nodes are busy)
srun: Job 99 step creation temporarily disabled, retrying (↩
   Requested nodes are busy)
srun: Job 99 step creation temporarily disabled, retrying (↩
   Requested nodes are busy)
srun: Step created for job 99
srun: Step created for job 99
srun: Job 99 step creation still disabled, retrying (Requested ↩
   nodes are busy)
srun: Job 99 step creation still disabled, retrying (Requested ↩
   nodes are busy)
n2-1.novalocal
srun: Step created for job 99
n2-2.novalocal
srun: Job 99 step creation still disabled, retrying (Requested ↩
   nodes are busy)
srun: Step created for job 99
n2-1.novalocal
n2-2.novalocal
```

B. Database content

```
$ MariaDB [slurm_acct_db]> select job_name, account, nodelist, ↩
   time_submit, time_end from cluster_job_table;
+------------+---------+---------------+--------------+------------+
| job_name   | account | nodelist      | time_submit  | time_end   |
+------------+---------+---------------+--------------+------------+
|     :      |    :    |       :       |      :       |     :      |
| hostname   | NULL    | n2-1          |  1679332933  | 1679332933 |
| hname.sh   | NULL    | None assigned |  1679494555  | 1679494599 |
| hname.sh   | NULL    | None assigned |  1679494820  | 1679494860 |
| hname.sh   | NULL    | n2-1          |  1679494887  | 1679494888 |
| hname.sh   | NULL    | n2-[1-2]      |  1679495048  | 1679495050 |
| hname.sh   | NULL    | n2-1          |  1679495192  | 1679495193 |
| hname.sh   | NULL    | n2-[1-2]      |  1679495603  | 1679495604 |
| hname.sh   | NULL    | n2-[1-2]      |  1679495681  | 1679495682 |
| hname.sh   | NULL    | n2-[1-2]      |  1679495760  | 1679495761 |
| hname.sh   | NULL    | n2-[1-2]      |  1679591894  | 1679592147 |
| hname.sh   | NULL    | n2-[1-2]      |  1679593389  | 1679593390 |
+------------+---------+---------------+--------------+------------+
92 rows in set (0,001 sec)
```