NHR Data Transfer Report

Freja Nordsiek (GDWG),
Steffen Christgau (ZIB), Jan Frenzel (TUD),
Marcel Nellesen (RWTH), Hendrik Nolte (GWDG),
Fabian Dünzer (RWTH)

June 2023

1 Introduction

Researchers may use more than one compute center for any of a number of reasons including:

- Have compute tasks that have different requirements (data capacity, hardware, connectivity, etc.) which can't all be meet by the same compute center.
- Preferred compute center does not have the capacity, so they spread the load across more than one.
- Large collaborations where each institution primarily uses one compute center and collaborators need to use that center to work with that data.
- Reuse of data that was generated by researchers from the center but that is too big to be transferred but is of interest for users from other centers

Additionally, researchers sometimes change institutions and change compute centers when they do and want to move their data with them.

Every compute center has one or more ways for its users to upload and download their data. Most even provide instructions for these ways including a recommended way. But when someone wants to transfer data between two compute centers that they are a user at, things get more difficult or less convenient. There are two main transfer strategies:

1. Transfer by intermediary host (Compute Center 1 $\rightarrow$ 3rd host $\rightarrow$ Compute Center 2).
2. Point-to-point transfer directly between the two compute centers.

Strategy 1 where the intermediary host is the person's own machines is the most obvious method and always works. After all, both compute centers must provide a way to upload and download data for their users. But, it isn't the most efficient method, requiring:

- Data has to be transferred twice
- User to have enough storage to host the temporary copy when the tool can't handle two remote hosts simultaneously, or have to do the transfer in stages
- Often limited by the internet bandwidth of the user (usually less than that of the compute centers)

Point-to-point transfers, when possible, are generally optimal. Though, they can have many barriers including

- Limited access to internet from inside frontend/transfer nodes, whether total or partial (e.g. outgoing SSH is blocked)
- Requirement of using a VPN to connect to the data center
- Security policies with regards to SSH keys and agents

1.1 Goals

As an alliance of computing centers in Germany, we want to provide our users with a well thought out path for moving their data between the different NHR centers (but also connect to Tier-3 and Tier-1 centers). Whether it be because our users are using more than one center, have moved institutions and want to use the closest center, or their needs have changed and a different center best meets their needs. To do this, we analyze transferring data between pairs of NHR centers in each direction, determining

1. Which transfers can be done point-to-point, and which must rely on an intermediary host
2. What is the best way to do each transfer
3. What is the transfer bandwidth a user could expect
4. How to do the transfer safely from a security standpoint

In the end, we will be able to provide information to users on the best way/s to safely transfer data between different NHR centers. We also hope to identify changes that could be made to improve the transfer story and/or provide more convenient methods for users.

2 Data Transfer Strategies

There are various alternatives to perform data transfer in regards to:

- Direction: push and pull. In push transfers, one machine transfers data from itself to another machine. In pull transfers, one machine transfers data on another machine to itself.
- Transport protocol: TCP, UDP, storage devices via truck[1]
- Synchronicity: Asynchronous (user request transfer, waits until completion), synchronous - user runs program to start transfer and has to wait until complete
- Single file/object vs. several files/objects

Our requirements for data transfer are:

- Usability - Easy to use
- Security
- Efficiency
- Prioritization - to allow more important/interactive packets to prioritize vs. background large-data transfers

The transfer strategies we currently use in production are SSH based. Some NHR centers are working on providing object storage systems (e.g. S3-like, Ceph, etc.) that would be accessible both internally in the compute center and from the outside via HTTP REST APIs. If these come to fruition, they would provide additional transfer methods for users.

2.1 SSH/sftp

`sftp`[2] is a simple client program based upon the SSH protocol with a similar interface to FTP clients that copies files and directories and allows one to see what files and directories are there, and is capable of doing both push and pull transfers. It is meant for interactive use, though it can be scripted in batch mode with the `-b` option. This means that it does not try to identify changes and only transfer the changes other than to the extent that in interactive use a person can manually decide which file/directories to transfer for updating and which not to send, instead transferring everything and overwriting. While very simple, this has the downside that subsequent synchronizations take the full transfer time. However, for simplicity to users, we will only consider the batch script case of transferring whole and not interactively deciding item by item. The simplest push sftp batch script is:

[1] https://iopscience.iop.org/article/10.1088/1748-0221/15/09/T09005/pdf https://www.researchgate.net/publication/220951078_Performance_adaptive_UDP_for_high-speed_bulk_data_transfer_over_dedicated_links

[2] https://man.openbsd.org/sftp

```
put -R -p LOCAL_PATH REMOTE_PATH
bye
```

and the corresponding pull batch script is:

```
get -R -p REMOTE_PATH LOCAL_PATH
bye
```

2.2 SSH/scp

`scp`[3] copies files and directories whole, and is capable of doing both push and pull transfers. This means that it does not try to identify changes and only transfer the changes, instead transferring everything and overwriting. While very simple, this has the downside that no synchronization is done, requiring the transfer time to upload all data.

In the past, `scp` used its own custom file transfer protocol but now uses the SFTP protocol by default. This is significant because while some compute centers require using a VPN for SSH shell access, a subset of these provide an externally available host that does allow SFTP since the OpenSSH server (among other SSH servers) can have that protocol enabled while disabling shell access thereby limiting security issues (less security issues with transferring files than with executing programs).

One special feature of `scp` is its ability to actually transfer between two different remote hosts, letting the machine it is run from be the intermediary host but without actually having to store the data locally. This can be particularly useful when a point-to-point transfer is impossible. In this event, a user's only choice is to download data to an intermediate host (usually the user's own machine) from one center and then upload it to the second center. For tools that, unlike `scp`, cannot transfer between two remote hosts, that intermediate host must have enough local storage for the whole transfer or the transfer must be broken up into smaller chunks each of which can fit in the storage of the intermediate host.

2.3 SSH/rsync

`rsync` (`https://rsync.samba.org`) is a file synchronization tool that transfers the differences between the source and the destination, and is capable of doing both push and pull transfers. While this makes the algorithm more complicated, it means that subsequent synchronizations of previously transferred data only have to transfer the changes. This can greatly decrease the required transfer time in many cases.

But, due to needing to calculate the differences to transfer, the `rsync` program must exist on both ends. `rsync` either launches `rsync` on the remote host via SSH, or the remote host runs an `rsyncd` server to connect to. Since the former is not accessible via the SFTP protocol, `rsync` will not work with remote

[3]`https://man.openbsd.org/scp.1`

hosts where the SSH server only allows SFTP (say, to let `sftp` and `scp` work) or has been explicitly allowed as a shell program in the SSH server config. This means that `rsync` will generally work in less situations than `sftp` and `scp`.

Additionally, unlike `scp`, `rsync` does not support transfer between two remote hosts. This means that if a point-to-point transfer is impossible, `rsync` must first transfer files to the storage of the intermediate host and then transfer them to the second compute center.

2.4 S3/rclone

The tool `rclone` can be used to migrate data from and to cloud storage. It allows the syncing, transfer, compress and mounting. Different types of cloud storage solutions are supported, e.g. S3 object stores. It can be used from Linux, Windows and Mac OS; therefore allowing an easy adaptation[4]. Additionally, unlike `scp`, `rclone` does not support transfer between two remote hosts. This means that if a point-to-point transfer is impossible, `rcone` must first transfer files to the storage of the intermediate host and then transfer them to the second compute center.

2.5 S3/MinIO Client

`MinIO Client` supports transfers between local filesystems and S3 cloud storage systems. It allows the usage of standard UNIX commands like ls, cat and cp to access and manage the data on the cloud storage systems[5]. Additionally, unlike `scp`, `MinIO Client` does not support transfer between two remote hosts. This means that if a point-to-point transfer is impossible, `MinIO Client` must first transfer files to the storage of the intermediate host and then transfer them to the second compute center.

3 Security Considerations

3.1 SSH

3.1.1 Basics

Many NHR centers restrict SSH shell access except through an SSH jumphost. This is not a problem for `sftp`, `scp`, and `rsync` which are easy to configure to use jumphosts. Many also restrict SSH shell access, allowing only SFTP access, on some nodes which prevents `rsync` from working but still lets `sftp` and `scp` work.

But, the big consideration with regards to users has to do with SSH authentication. Point-to-point transfers mean that one has to SSH into the node of one NHR center and initiate the transfer to/from a node of a different NHR center via an SSH based method. This means that the user must authenticate

[4]`https://rclone.org/`

[5]`https://github.com/minio/mc`

to the second center while logged into the first, which has a number of security implications and requires consideration of the security policy.

With SSH public key authentication, an NHR center's security policy may quite reasonably forbid users from placing their SSH private keys on any machines other than ones they directly control (which is not the case for nodes of another NHR center) or place SSH keys for external machines on a machine controlled by the center. That means that a user cannot just place their private key on the node of the first center to facilitate transfers to/from the second center. Even if the security policy of both NHR centers allowed it, it is still a **VERY BAD IDEA!** Those SSH keys can be stolen, and the decryption passwords gotten by merely logging the SSH session.

SSH password based authentication (for the NHR centers that enable it) has the same issues. For very good reasons, an NHR center may not allow a user to enter their password to the center on machines not controlled by the user or to enter passwords to machines not controlled by the center. But even if both NHR centers allowed it, it is still a **VERY BAD IDEA!** The passwords can be gotten by merely logging the SSH session.

This leaves SSH agent forwarding as the only remotely tolerable option between NHR centers that don't share user authentication (e.g. GWDG Göttingen and ZIB Berlin). But naive SSH agent forwarding is still opening Pandora's box because, by default, the SSH agent says yes to every key request. There are two possible ways to mitigate this.

3.1.2 SSH-agent confirmation

First, if the SSH agent prompts the user on their personal machine for confirmation for each request, this risk is mostly mitigated. This represents the currently advertised solution at most NHR centers. An attacker would need to get the timing perfect to get their request in after the user hit ENTER on the command but before the user's command sent the request, but the user would very quickly notice something is weird because they would then get a second request (of course, this does depend on the user knowing what this means and the reaction time required to stop an attack being long enough that the user has even a hope of stopping it). The trick, if using OpenSSH's included SSH agent, is using `ssh-add -c`. However, due to the added and irregular latency from the user having to click yes on the confirmation every time, this makes benchmarking more difficult. It also doesn't work for exhaustive benchmarking with many many trials. But it is the best option to recommend to users.

3.1.3 authorized_keys

The second option is to use dedicated single purpose SSH keys for any SFTP based tool and each non-SFTP based tool for each NHR center and lock them down in `$HOME/.ssh/authorized_keys`[6]. Of course, the best security comes from using this and the SSH-agent asking for confirmation. The idea is to set

[6](`https://man.openbsd.org/sshd#AUTHORIZED_KEYS_FILE_FORMAT`)

the `command` option for each single purpose key to restrict what it can do, as well as the `restrict` option to disable all forwarding among other things. For `rsync`, one could set it to the `rrsync` script[7] to further restrict what directory `rsync` can be run on as well as other things like if it should be read-only, write-only, or no files can be deleted. For pull transfers with SFTP based tools, one could use `internal-sftp -R` to make it so that SFTP access is readonly. This limits the damage that can be done if the special use SSH key in the forwarded SSH agent gets compromised from not asking for confirmation, particularly the the `rsync` key since it would only be able to affect the directory specified in its `command` field. Unfortunately, the key for SFTP based tools would still be rather dangerous as that one could still read all the user's files. This is acceptable for benchmarks with functional accounts, but being able to read all the user's files means that for SFTP, this is not good enough to recommend to users. But left out is SFTP based push transfers which need write access. OpenSSH's `internal-sftp`, and its higher overhead and separate executable `sftp-server` provide no option to contain them to only a specific directory, so they can't be used by themselves to give the proper security. However, if the target machine has unprivileged user, mount, and PID namespaced enabled; one could use a script like the following to lock down access to a single directory (second argument) for either tool (first argument) in what is essentially a primitive container:

```
#!/bin/bash

# First argument, $1, is the tool (sftp or rsync)
# Second argument, $2, is the path to the directory where the transfer
#   will take place in

# Determine the actual command to run at the end for the transfer.
if test "$1" = "sftp"
then
  CMD=sftp-server
else
  CMD=SSH_ORIGINAL_COMMAND
fi

# Make the directory for the execution trap
rm -rf TRAP
mkdir -p TRAP
cd TRAP
tar -xzf TRAP_TARPATH.tgz

# Get the absolute path for unshare for use after PATH has been changed.
UNSHARE=‘which unshare‘

# Launch the first namespaces and in those namespaces:
# 1. Make the bind mount
# 2. Change PATH
# 3. Launch the second set of nested namespaces, chroot, and launch the transfer program
exec unshare -mpU -fr --mount-proc bash -c \
    'mount --bind $2 ./tdir; export PATH="/usr/bin"; exec $UNSHARE -mpU -fr --mount-proc -R ./tdir $CMD'
```

Where `TRAP` is the directory the execution will be trapped in and `TRAP_TARPATH.tgz` is the path to a tarball containing the contents for the `TRAP` directory (it needs

[7](`https://download.samba.org/pub/rsync/rrsync.1`)

to be made fresh each time in case the transfer tool damages it on an earlier iteration). Note that `TRAP_TARPATH.tgz` must be outside of `TRAP` to protect it. Note that, as found out from the `trappedssh` project (it is a spinoff of the idea presented here) that tries to do this in a more systematic way for all users (as an alternative to an SFTP-only host), it is very difficult to get these kinds of shell scripts right and not have string bugs that open Pandora's box by running code controlled by the SSH client outside the sandbox. `TRAP_TARPATH.tgz` would be pre-prepared to have the following directory structure:

tdir/ Directory that will be the bind mount point for the directory to access.

lib*/ Library directories for any libraries needed by the programs in `usr/bin`.

usr/bin/ Directory for all executable programs that are used for the transfer on this end including `sftp-server` for SFTP based tools and `rsync`.

usr/lib*/ Directories for any libraries needed by the programs in `usr/bin`.

Note that for the transfer, the directory at `$1` becomes `/tdir` in this execution environment. With this script, then the `command` field in `.ssh/authorized_keys` would be `"TRAP_SCRIPT sftp DIR"` for SFTP based methods and `"TRAP_SCRIPT rsync DIR"` for `rsync` where `TRAP_PATH` is the path to the script and `DIR` is the directory that the transfers should take place in. On systems with the enabled unprivileged namespaces, this will generally be safer than `rrsync` or `internal-sftp -R` due to the additional isolation. But do note, however, bugs in the namespaces implementations can pose security risks.

3.2 VPN

Rather than leave SSH access to the frontend/transfer nodes open to the entire internet, many NHR centers require a VPN to access the frontend nodes. The NHR centers requiring a VPN are

- RWTH
- TUD
- TUDa

This makes transfers to those centers impossible unless one is on the VPN, which other NHR centers are **NOT**. This necessarily means that push transfers to an NHR center requiring a VPN are impossible from other NHR centers without the use of virtual machines or heavily modified file transfer tools combined with userspace networking. As that is high overhead and requires advanced knowledge and tools, we cannot recommend users to use such methods. Thus, we only consider pull point-to-point transfers and transfer by intermediate host with these NHR centers.

For transfers between two NHR centers that require a VPN, point-to-point transfers are not feasible at all except for very advanced users and thus we can

Center	Jumphost	SSH Out	Transfer Nodes	VPN	SSH Protocols inside VPN	SSH Protocols outside VPN
FAU	✓	✓				shell, SFTP
GWDG		✓				shell, SFTP
RWTH		✓	✓	✓		shell, SFTP
TUD	✓	(only dedicated hosts)	✓	✓	shell, SFTP	
TUDa		✓		✓		shell, SFTP
ZIB		✓				shell, SFTP

Table 1: The NHR centers participating in this project and whether they require SSH jumphosts, allow outgoing SSH connections, have dedicated transfer nodes (login/frontend nodes are used otherwise), have a VPN, and what SSH protocols are allowed inside and outside the VPN if present (blank means either not applicable or nothing).

only consider transfer by intermediate host. Depending on the VPN configurations, it can be difficult or impossible to configure a single machine to be in both VPNs securely, so using `scp` with both remote hosts simultaneously is not a good idea. Instead, a user would need to connect to the VPN of the source compute center, download some or all of the data, disconnect from that VPN, connect to the VPN of the destination compute center, and then upload the data; repeating as necessary till all data is transferred.

4 NHR Center Overview

An overview of the different NHR centers participating in this project and how they can be connected to is given in Table 1. Some NHR centers have dedicated links to each other, which can help facilitate faster data transfer than over the open internet, which are

- ZIB–GWDG (2 × 10 GBit/s Ethernet)

Additionally, some NHR centers have direct connections/cooperations with other non-NHR HPC centers such as

- JARA (Jülich Aachen Research Alliance) (JSC–RWTH)

5 Hot, Warm, and Cold Data

While large amounts of data are generated and/or processed on HPC clusters, HPC clusters are not meant for long term data storage. However, interest in locally available data might increase by publications when researchers from other centers might want to reuse or analyze the data themselves. In this case, the research data will already be stored in locally available storage and research data management systems. The accessibility and the performance of these systems is an important factor in the reusability of the research data.

Center	Storage Name	Storage Path	File System
FAU	FASTTMP	`/lustre/...`	Lustre
GWDG	WORK	`/scratch/usr/...`	Lustre
RWTH	HPCWORK	`/hpcwork/usr/...`	Lustre
TUD	Workspace	`/scratch/ws/...`	Lustre
TUDa	HPC_SCRATCH	`/work/scratch/...`	GPFS
ZIB	WORK	`/scratch/usr/...`	Lustre

Table 2: Employed file systems for the performance study

Every cluster provides a `HOME` directory, which is where the data is the full temperature range (hot since it is very available, cold because it is also backed up) can be placed but often the performance and/or quota is small. This data is not meant for sharing between users. Most clusters provide one or more “work” directories for hot data with improved performance and capacity, but often meant for medium-term storage as opposed to long term storage. Often, project sub-directories are provided in addition to user sub-directories, which makes the “work” directories good places to put hot data meant to be used by more than one user. Most also have some sort of “temporary” storage which is the hottest but meant only for very short term storage (e.g. automatically deleted after a very short time interval). Some clusters provide access to a “permanent” directory for cold data to be stored long term with backups, with low performance (e.g. a tape library) and often limited capacities.

The transfers we consider are for between the hot and warm storages of the different NHR centers, so this includes the `HOME` directory and “work” directories. But “permanent” directories are excluded.

6 Data Transfer Benchmarking

6.1 Methodologies

6.1.1 Manual Benchmarking

The main focus of the benchmarks was the performance of large scientific and “hot” data between NHR benchmarks. Therefore, the main work storage was chosen as the source and target of the transfers. This storage is typically different from the home directories, usually realized by a Lustre or similar filesystem. Table 2 shows the file systems used within the studies.

To mimick a realistic transfer, a 10 GiB source file was created on each site. The file was filled with random data to foil any compression that a tool may try to use. That source file was then transferred five times from and to the participating NHR centers. For each transfer, a different target file was used to avoid optimizations due to in-situ replacement of an existing file. Out of the five transfers, the one with the median runtime was selected for reporting the value. It was also verified that the local storage performance exceeds the network bandwidth between the NHR centers.

6.1.2 Beginnings of an Automated Benchmarking Program

An automated transfer benchmarking program has been built which can do `rsync`, `scp`, and `sftp` with a batch script. It first loads definition files to get the sizes of the benchmarks, the storage stores at each NHR center to benchmark, the login and transfer hosts for each NHR center, and the usernames and other variables to use for each NHR center. Then it iterates over every pair of NHR centers it has been instructed to benchmark and does the following:

1. Connect to both NHR centers via SSH.
2. From each NHR center, probe which SSH protocols (shell and SFTP), if any, the other one allows on its login and transfer hosts, if any.
3. Iterate over each storage pair to benchmark between (home with home, work stores with work stores, etc.)
   (a) Iterate over each file size to benchmark
      i. Iterate over each center in the pair to be the source of the data
         A. Make the files to transfer. The number of files that are made is determined by the target total size (from the configuration files) and the maximum number of allowed files (some NHR centers have a maximum allowed inode usage in some data stores).
         B. Iterate over every combination of transfer method, transfer direction (push/pull), and host type (login/transfer) that the probed SSH protocols indicate are possible. For each combination, transfers are done one after another till the minimum number of trials and minimum transfer time has passed (these are set by CLI arguments to the program). Each transfer tool is operated in a way that the time needed to actually do the transfer can be measured while the time to establish the SSH connection and authentication are excluded (though the total time is still recorded). After each transfer, the files on the target are deleted.
         C. Delete the files on the source.
4. Close SSH connections.

The benchmarked file sizes are 0 B, 1 B to 256 MiB in multiplicative steps of 16, and 1 GiB to 8 GiB in multiplicative steps of 2.

The program works between GWDG and ZIB, which have a shared authentication system and thus don't require the SSH agent at all.

Results During the execution, we identified two challenges: One is that it is slow, taking about 3 full days for just GWDG and ZIB, though this can be improved by making the range of file sizes smaller or parallelizing over NHR center pairs to do as many pairs that don't overlap with each other as possible at a time. The second problem is the bigger problem. For a single pair of NHR centers, the program can do thousands of individual benchmark trials which means thousands of SSH authentications. This makes use with SSH agent forwarding with confirmation prohibitively difficult because the operator must then click the confirmation thousands of times per NHR center pair. For all NHR centers, this would be many tens of thousands of confirmations over a few weeks assuming confirmation was instant (with confirmation, several weeks or a few months). Additionally, with such a large number of authentications, the security benefits of asking for confirmation begin to diminish as the operator tires and cannot tell a confirmation request for a benchmark apart from a confirmation request that an adversary might be making on the respective NHR center. So, we are looking into using functional accounts with special purpose keys with `$HOME/.ssh/authorized_keys` as described in Section 3.1.3 just for benchmarking where we can turn SSH-agent key confirmation off with little impact in the event of a compromise. Though this will limit the possible benchmarks that can be done between NHR centers where one or both do not allow unprivileged user, mount, and PID namespaces (SFTP-based pull transfers and `rsync` via `rrsync` would still be reasonably safe).

6.1.3 S3 data transfer

The performance of file transfers between the different NHR centers and different S3 storage solutions offered by the centers was investigated.

Configuration NHR@Göttingen provides an NVME-based Ceph cluster to provide solely an S3 interface.

Origin / Target	GWDG S3	RWTH S3	TUD S3
GWDG Cluster	40MiB/s	44 MiB/s	15 MiB/s
RWTH Cluster	155 MiB/s	109 MiB/s	19 MiB/s
TUD Cluster	62 MiB/s	62 MiB/s	48 MiB/s

Table 3: Uploading files from the computing cluster to a S3 storage system. Ten files with each 10gb were transmitted with `rclone`

The results of the measurements show a strong impact of the selected tool on the transfer speed. Our experiments show that e.g. transfering data through the minIO client can result in a performance increase of at least 200% when compared with rclone.

Comparing the results for NHR@Göttingen with a proper S3-Warp benchmark, as it was performed last year in the storage report, one can see that the full bandwidth wasn't nearly satisfied, which peacked back then at 12314 MiB/s

Origin / Target	GWDG S3	RWTH S3	TUD S3
GWDG Cluster	215 MiB/s	138 MiB/s	118 MiB/s
RWTH Cluster	534 MiB/s	489 MiB/s	488 MiB/s
TUD Cluster	199 MiB/s	189 MiB/s	191 MiB/s

Table 4: Uploading files from the computing cluster to a S3 storage system. Ten files with each 10gb were transmitted with `minIO`

for 64MiB filesizes for the mixed Bandwidth measurement. An important remark is, that the Ceph-cluster has a very efficient caching, thus distorting this value for the mixed measurements.

Other Centers In a different context the file transfer to other centers' S3 storage solutions was investigated using `rclone` and `minIO`. The median results of five measurements are shown in 5 and 6. Again `minIO` reaches higher speeds than `rclone`, though not always by as much as in the previous benchmarks. As neither tool allows point-to-point transfers, the transfer speeds involving the RWTH cluster are most relevant, providing approximations of the upper bounds for corresponding transfers between S3 storage systems.

Origin/	RWTH	RWTH	GWDG	UDE*	UzK*	ZIH*
Target	Cluster	S3	S3	S3	S3	S3
RWTH Cluster	-	261	253	251	101	34
RWTH S3	237	-	268	220	108	34
GWDG S3	186	141	-	159	100	35
UDE* S3	175	131	144	-	107	32
UzK* S3	100	111	111	111	-	31
ZIH* S3	36	31	32	33	39	-

Table 5: Copying files between the RWTH data transfer nodes and S3 storage systems. The centers marked with an asterisk are not members of NHR. All values are given in MiB/s. 20 files with 1 GiB each were transmitted with `rclone`

7 Results

In order to compare the transfer performance, the reported transfer time of the employed tools was used together with the known file size to derive the transfer rate. The rate is reported in MBit/s to facilitate a comparison with the network connections between the centers. Table 7 shows the outcome of the experiments using the mentioned tool. For easier overview, the results are also depicted in Fig. 2, where arrows indicate the start and end of data transfers. Each arrow is annotated with the maximum bandwidth achieved in the experiments. The transfer rates are for 10 GiB files via the manual benchmarking (all rates except

Origin/ Target	RWTH Cluster	RWTH S3	GWDG S3	UDE* S3	UzK* S3	ZIH* S3
RWTH Cluster	-	898	1248	365	111	763
RWTH S3	656	-	544	411	109	275
GWDG S3	299	270	-	228	108	196
UDE* S3	434	320	332	-	109	110
UzK* S3	108	111	110	110	-	110
ZIH* S3	186	183	172	176	100	-

Table 6: Copying files between the RWTH data transfer nodes and S3 storage systems. The centers marked with an asterisk are not members of NHR. All values are given in MiB/s. 20 files with 1 GiB each were transmitted with `minIO`

between GWDG and ZIB) and 8 GiB files via the automated benchmarking tool (just for rates between GWDG and ZIB). Note that the data transfer rates are not symmetric. The speed of a file transfer depends on the data center from which the transfer originates. In addition it is also important to take the direction of the transfer into account.

We were only able to benchmark the different tools as a function of file size with the automated benchmark tool for one pair of NHR centers (GWDG and ZIB). An example plot of the file transfer times and data transfer rates between ZIB and GWDG is shown in Figure 1. For this pair of NHR centers, we found that for small file sizes, `rsync` performs much better than `scp` and `sftp`. However, for larger file sizes, `scp` and `sftp` catch up with `rsync` and sometimes exceed its performance, as seen in Table 7. From this, we conclude that `rsync` is much more efficient with transferring metadata, which dominates for small files; but that it can sometimes be less efficient for the actual file data itself, which dominates for large files. While the bandwidth limit for the tools is different for different pairs of NHR centers and likely the minimum time to transfer each file even if it has zero size (at the very least, it is latency bound, which is determined by the total fiber/coax distance, the number of hops, etc.), we expect that the pattern is similar for each pair of NHR centers. From experience between a variety of different machines in various locations, `rsync` dominates for hundreds and thousands of small files which suggests this is not an unreasonable expectation that the pattern is similar between each other pair of NHR centers.

8 Recommendations

Based on the results presented in the previous sections and discussions during the monthly Data Management meetings, recommendations regarding a few selected points are proposed.

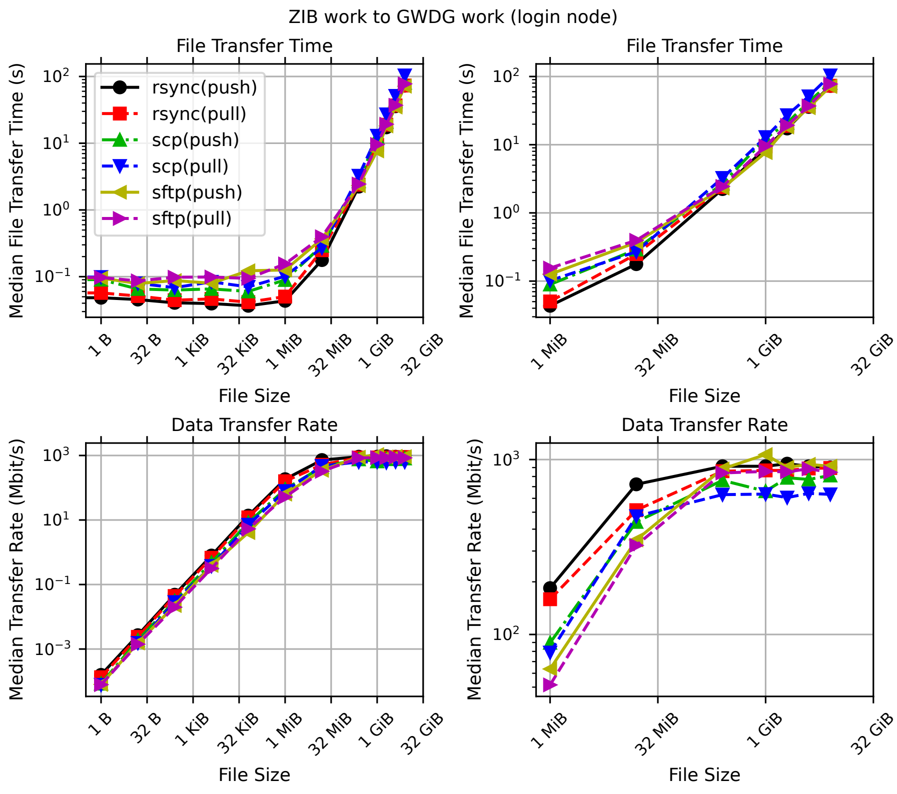

Figure 1: The transfer performance for transfering files from ZIB to GWDG as a function of file size for the two sites' work storage systems. The top two plots show the median transfer time per file as a function of file size, and the bottom two show the median data transfer rate as a function of the file size. The left two plots who the whole range of file sizes, while the right two plots are zoomed in to show only file sizes $\geq$ 1 MiB. Each different transfer method and direction combination are shown as different lines. All plots share the same legend, which is shown in the top-left plot.

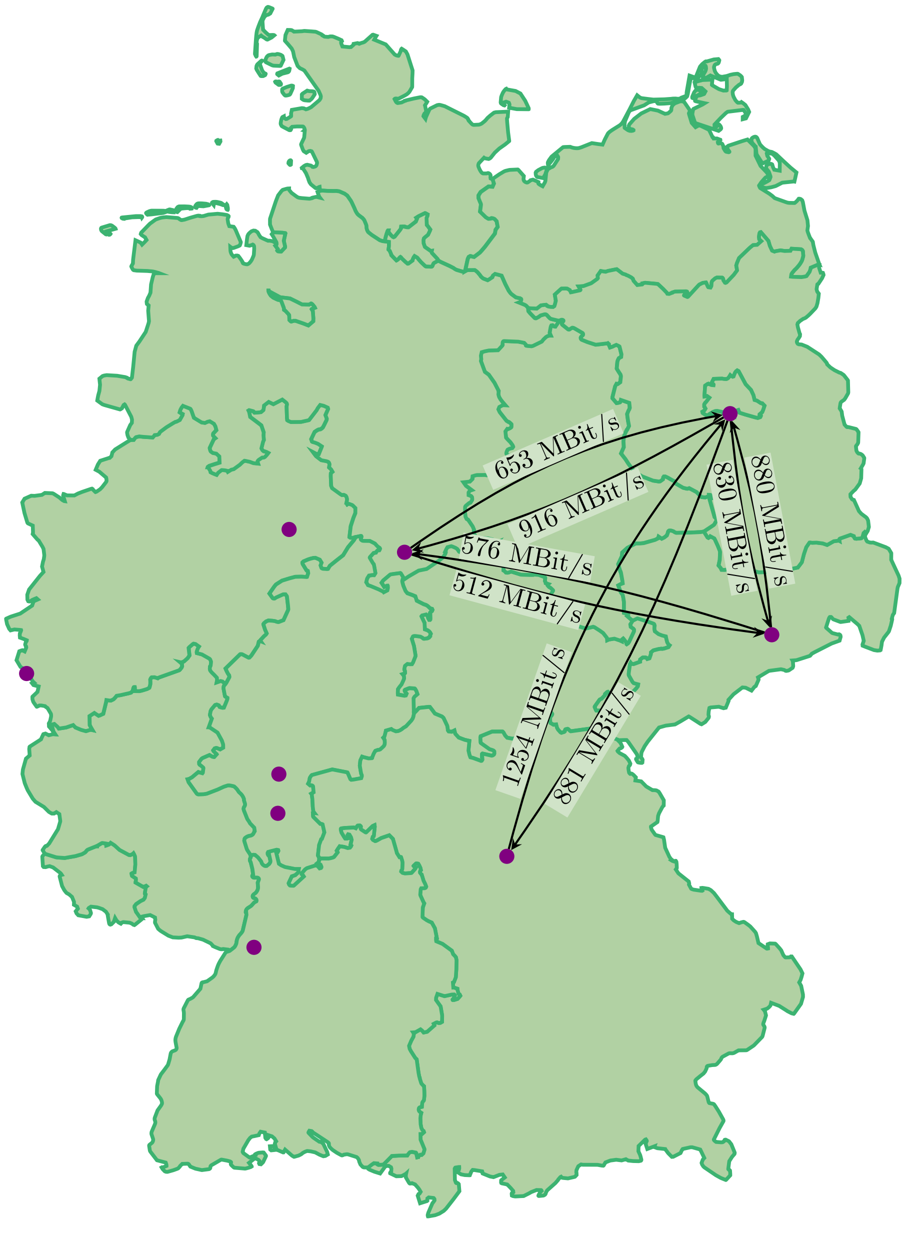

Figure 2: NHR centers in Germany and the maximum bandwidth between them as measured in the data transfer experiments.

Origin/Target	FAU	GWDG	TUD	ZIB
				rsync
FAU	—	?	?	1232
				1254
			rsync	sftp
GWDG	?	—	512	653
			496	647
		rsync		rsync
TUD	?	576	—	880
		433		666
	rsync	sftp	rsync	
ZIB	690	916	830	—
	881	845	846	

Table 7: Maximum data transfer rate between the NHR centers in MBit/s for large files (10 GiB or 8 GiB depending on the benchmarking method). The origin of the data transfer is the center in which the data transfer is issued. The origin is shown in the first column, the target in the first row. The first line in a cell shows tool that achieved the best rate. The second line tells the transfer rate from the origin to the target (origin → target, push), and the third line the reverse direction (origin ← target, pull).

8.1 S3

Using S3 as an indirect medium for data transfers is **not** recommended. We could observe that a) the transfer speeds were (much) slower compared to rsync/scp and b) it has to be done twice, once for uploading, and once for downloading. However, data is usually only staged on an HPC system for compute purposes, which only covers part of the entire data life cycle. That means that data has to reside somewhere else. This main storage could be done via S3, particularly since most implementations offer a high durability. S3, due to its HTTP interface, has no security implications for the HPC system itself. Therefore, we recommend that users should take a storage system into account, which offers an S3 interface when choosing a storage system to hold their data during the other, interactive, parts of the data life cycle. If so, users should keep in mind that this will expose the secret and access keys of their bucket to an attacker on the HPC system. Therefore, we recommend to use keys with limited privileges, i.e. restricted to read access, or restricted to limited prefixes. In addition, if no FUSE mount is required, we recommend to use the minio client to sync data rather than rclone due to its massive speed advantage.

8.2 Data Mover Nodes

During the project we had vivid discussions about data mover nodes, as some centers offer them while others do not. In general, data mover nodes are frontends, which solely serve a single purpose: Moving data. This allows, for instance, to only allow a locked down access, e.g. only provide sftp access, and not full shell access. This in turn can enable providers to open up the firewall to provide easier data transfers to users. In addition, these nodes can be equipped with higher network bandwidth compared to other nodes. To summarize, there are two main advantages:

- **Security:** If a center uses a firewall and only offers access to the frontend nodes via a VPN, data mover nodes can be located outside of the firewall, since one can configure the SSH server so that only SFTP is allowed, but no shell access. This limits the overall exposure of the system, while offering convenient and fast data transfer services.
- **Slurm Integration:** Data mover nodes can be hosted as a Slurm partition. This enables the users to "sbatch" a data staging job. This has the advantage that jobs which depend on the successful data transfer can be dealt with using Slurm job dependencies, easing this burden from the user to manage these dependencies in a different manner.

However, there are also concerns that data mover nodes will add further complexity to a system, both for users and admins. On the one side, users have to be aware of the data mover nodes and have to use them, which might not be the default for all users which will still use the login nodes for data transfers. In addition, these nodes also require additional effort by admins, particularly if these nodes are located outside of the firewall and are therefore very exposed. Of course, the latter could be handled by only allowing access from IP-addresses upon explicit user request.

Last but not least, there is the general problem of dark data, i.e. data which is there, which no one really cares that it is there. By offering services for large scale, bulk transfers, users could be encouraged to just bulk-transfer their data blindly, skipping work-intensive filter steps.

8.3 Securing Connections

The easiest way is to restrict the SSH server to not allow shell access. One could further provide documentation and training to users on how to use the ssh config to restrict the capabilities of keys, and offer similar services for rrsync. The rather advanced method described in Section 3.1.3 was intended to pitch an idea and raise awareness.

It can also be recognized, that in the case of a security event it might be useful to offer a different way for users to access their data. For this, WebDAV might be a good solution, which could also be used beyond pure emergency access, since it can also work with a second factor.

9 Conclusion

Within this report we analysed the data transfer between the different NHR centers and provided reference values, guidelines; and best practices for researchers and operators of compute clusters to improve their data transfer rates.

Within the previous sections the performance of the data transfer between the centers was analysed. Based on this analysis, a few general recommendations for data transfer from, to and between the NHR centers can be made.

The first step for every researcher should be an analysis of the current situation. The following questions can provide a guideline: What amount of data should be transfered?

- Where is the data generated?
- What amount of data should be transfered?
- How many files should be transferred?
- Is it necessary to transfer the complete data or is only a subset required for the computation?
- Where should the data be processed and stored?

After the questions above the data can be transferred. The measurements within this report can than be used as a reference value, in case the achieved transfer rates are constantly significantly smaller than the values presented, the local infrasturcture might enforce a bottleneck on the overall transfer speed.