GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Fakultät für
Physik

Bachelor's Thesis

Simulationsstudien zur Integration von NHR@Göttingen in das WLCG

Simulation Studies on the Integration of NHR@Göttingen into the WLCG

prepared by

Adrian Leone

from Erlangen

at the II. Physikalischen Institut

Thesis number:	II.Physik-UniGö-BSc-2026/04
Thesis period:	6th April 2026 until 10th July 2026
First referee:	Prof. Dr. Arnulf Quadt
Second referee:	Prof. Dr. Thibaud Humair

Zusammenfassung

In wenigen Jahren wird GoeGrid, ein Teil des Worldwide Lhc Computing Grids (Wlcg) von Cern, auslaufen. GoeGrid nutzt, neben seinem eigenen Cluster, das Cluster mit dem Namen Emmy als opportunistische Ressource. Emmy ist dabei Teil des Nationalen Hochleistungsrechnens (Nhr) und hat im Gegensatz zu GoeGrid keine gesonderte Speichermöglichkeit für am Cern generierte Daten. Zusätzlich hat Emmy nur eine Firewall mit geringer Bandbreite, um Daten zu beziehen, und bezieht daher teilweise seine Daten von GoeGrid. Ziel dieser Arbeit ist es, mittels Simulationen Möglichkeiten zu finden, die die Situation um Emmy bezüglich des Wlcgs verbessern. An diesem Ziel ist diese Arbeit jedoch gescheitert, da ein Bug in der Simulationssoftware nicht innerhalb des Zeitrahmens dieser Arbeit behoben werden konnte.

Stichwörter: Nhr, Wlcg, Computernetzwerke

Abstract

In a few years, GoeGrid, a part of the Worldwide Lhc Computing Grid (Wlcg) from Cern, will be retired. GoeGrid uses, apart from its own cluster, the cluster named Emmy as an opportunistic resource. Emmy is part of Nationales Hochleistungsrechnen (Nhr) and, unlike GoeGrid, has no special storage capabilities for data generated at Cern. On top of that, Emmy has only a firewall with limited bandwidth for data retrieval, so it retrieves part of its data from GoeGrid. The goal of the work is to find ways to improve the situation around Emmy concerning the Wlcg, using simulations. This work has failed to meet this goal, as a bug in the simulation software could not be fixed within the time allotted for this work.

Keywords: Nhr, Wlcg, Computer networks

Contents

1. Introduction

Our lives would be unthinkable today without technologies made possible by discoveries in physics. Even though not visible to the naked eye, particle physics lays the foundation of physics by studying it at the smallest scale and seeking to uncover the most fundamental facts about our universe.

This is where CERN enters the picture. Established in 1954, CERN has to credit numerous findings in particle physics to itself. Over time, CERN has grown not just in personnel and particle accelerator size, but also in the amount of data it generates and stores. This is necessary to find even the rarest of events with statistical significance. CERN continues this trend to this day by shutting down the Large Hadron Collider (LHC) to upgrade it to the High-Luminosity LHC. This means the LHC will be improved to generate even more data.

But how can all this data be processed in a timely manner? For this, a computing grid has been set up. These are clusters of computers distributed across multiple research institutions and countries worldwide. This specific grid for processing data from the LHC is called the Worldwide LHC Computing Grid (WLCG).

In Göttingen, there is one of the computer clusters. This cluster, being part of the WLCG, is called GoeGrid. GoeGrid also utilises another cluster named EMMY as an opportunistic resource. EMMY is only utilised as an opportunistic resource because it is part of NHR and thus does not exclusively serve particle physics. NHR, short for Nationales Hochleistungsrechnen, is an organisation that manages computing resources available to scientists at German universities free of charge.

GoeGrid has, aside from compute nodes, storage nodes. On these storage nodes, a chunk of the data generated by the LHC is stored for processing on the compute nodes. EMMY only has compute nodes, with no extra storage for the LHCs data. This is a problem, as, when facing the internet, traffic to and from EMMY compute nodes has to pass through a firewall with only 1 Gb/s of bandwidth. A firewall with this limited bandwidth is, of course, insufficient. For now, this can be somewhat bypassed by having EMMY use the storage resources of GoeGrid. But GoeGrid will be phased out in a couple of years.

To mitigate this problem, this work sets out to simulate possible new arrangements in

and around Emmy, such as a higher-bandwidth firewall and a new storage server.

2. Grid Computing

A computing grid is an interconnected collection of computing clusters, called sites, that work towards a common goal. In addition to compute resources, sites can also offer storage resources. Storage resources can be in the form of hard drives and tape. Storage and compute systems sharing the same site can be expected to have more bandwidth available to connect to each other than systems at different sites, due to physical proximity.

To simulate Monte Carlo events and to process events generated by simulations and by the LHC at CERN, a grid called the Worldwide LHC Computing Grid (WLCG) is being operated. Specifically, this work focuses on the ATLAS experiment component of the Worldwide LHC Computing Grid (WLCG). Components of which will be discussed in the following sections.

2.1. Workflow Management

For the ATLAS experiment, different types of workflows are required, such as processing measured data from experiments and Monte Carlo event simulation [1]. So, to distribute the computing workloads across the grid, automatically and efficiently, a management system is needed.

To implement this system, ATLAS uses multiple software components. An overview of the WLCG workflow is shown in Figure 2.1. Using them allows ATLAS to manage workflows across different sites, leveraging heterogeneous resources.

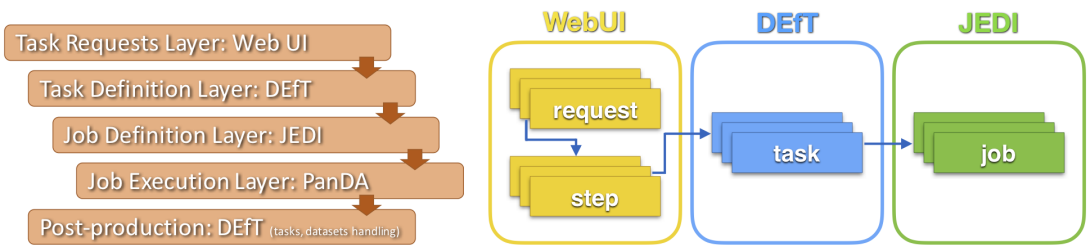

Figure 2.1.: Workflow components overview. Figure from Reference [2].

Web-UI Using the Web User Interface (Web-UI), workflows such as processing measured data and Monte Carlo event generation are defined, as mentioned above. With workflows defined, users can use the workflow as a template to form a request, filling in input datasets and other parameters. The user's request is broken down into steps for each input dataset.

DEfT Beneath the Web-UI, the Database Engine for Tasks (DEfT) sets unset parameters to their defaults and checks their compatibility with the specified software. After processing the steps into tasks, similar past tasks are looked up to avoid duplicate computations.

JEDI The Job Execution and Definition Interface (JEDI) receives the tasks sent from DEfT. JEDI interfaces with Rucio, see Section 2.2, to verify the input data information.

PanDA PANDA manages the computing resources allocated to each task. Also, PANDA shares the database with JEDI to store metadata for past and running tasks and jobs.

To ensure smooth and efficient operation, pilot jobs are utilised. Pilot jobs are used to estimate resource usage for subsequent job definitions. They also help catch job failures before the job actually starts, preventing wasted resources.

Some job failures are recoverable. A catalogue of known errors is kept for this. For example, faulty software cannot be recovered until it is fixed. This catalogue defines the number of retries for each error: retry jobs with recoverable errors, or stop jobs with irrecoverable errors. System and site failures can be mitigated by redistributing jobs across the grid.

2.2. Storage Management

To manage the data for the ATLAS experiment, which would not fit on a single computer and whose total exceeds 800 PB as of 2023 [1] and used storage capacity over time can be seen in Figure 2.2, a dedicated storage system spanning multiple storage nodes is needed. For this purpose, the ATLAS experiment at CERN has developed its own system, Rucio [3]. Rucio is now also used by other experiments.

Aside from detector data, Rucio is also used for storing simulation and user data [3]. Rucio works across heterogeneous storage media and across all the sites. Institutions providing storage can set policies for the stored data, such as quotas and access restrictions.

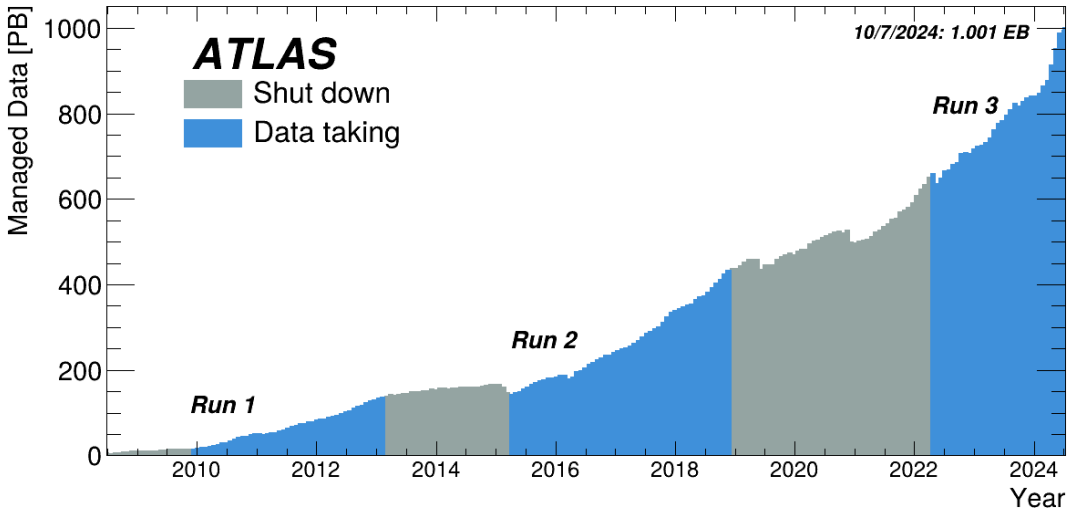

Figure 2.2.: Amount of data managed by Rucio over time. Image from `https://rucio.cern.ch/`.

Atlas utilises one Rucio instance, managed by Cern, unifying all the storage units that are part of it. This allows all data to be accessed via a single interface and namespace. To keep things organised, datasets submitted to this Rucio instance must follow specific naming schemes.

Rucio interfaces with PanDA and ProdSys. This allows users submitting jobs to specify input data as needed. Transparent to the user, this input data is then copied to the site running the job as needed. Alternatively, users can manually download data using Rucio that they have access to.

In addition to providing users with access to files, Rucio also has an automatic replication system. This allows data fulfilling certain criteria to be replicated to specific locations and on specific media. For example, measurements from the Atlas detector will be replicated to a site in another country via tape storage.

3. Modelling

To model the WLCG, three pieces of software, named SimGrid, Wrench and DCSim, built on top of one another, will be discussed in the sections below. Each of these pieces provides different levels of abstraction from actual hardware. The last section discusses scaling down the model to reduce computation while maintaining accuracy.

3.1. SimGrid

SimGrid[1] lays the foundation by implementing multiple essential computer components, called resources. On these components, activities can be run. They are spawned by user-defined actors. When the activities run on resources, their usage is emulated. Actors can also dynamically create other actors. In total, all actors are managed by an operating system-like actor called maestro. The maestro schedules actors and keeps track of time.

At the beginning of the simulation, the maestro starts the actors. The actors start up their user-defined activities. If possible, they return immediately. Otherwise, when all actors are blocked, the maestro fast-forwards the simulation to the time at which the first activity completes.

The following computer components are modelled.

CPU CPUs are modelled only having a time-dependent processing speed. This processing speed abstracts away different architectures and features, such as caches. Although such architectures can, in principle, be taken into account by building a combination of storage and CPU resources on the platform, replacing a CPU.

Storage Like the CPU model, storage is abstracted into a transfer speed. But in addition to speed, a seek time is added.

Network While networks can be modelled at the packet level, as is done to analyse network protocols, this is too computationally expensive for systems of the size of the

[1] `https://simgrid.org/`

WLCG.

To resolve this problem, the model can be simplified to bandwidth and latency. This implemented model in SimGrid simulates the network with high accuracy [4]. Very small data transfers cannot be modelled due to the discrete nature of packets over a network. This does not significantly impact the WLCG simulation, as data transfers exceed this threshold. In addition to bandwidth and latency, the network layout has to be specified.

3.2. Wrench

The next piece in the software stack is Wrench[2]. It is an abstraction on top of SimGrid, described in Section 3.1. Instead of computer parts, it models services. In reality, the corresponding services run on one or more complete physical computers.

Bare-Metal Compute Service The Bare-Metal Compute Service provides a basic platform for jobs to run on. The job is assigned to a single host, and every host that runs jobs runs a bare-metal compute service. It runs and gets control of a fraction of the host's CPUs and RAM. In addition, local storage for intermediate outputs can be configured.

HTCondor Compute Service The HTCondor Compute Service consists of one machine that provides the actual service, with Bare-Metal Compute Services assigned to it to handle jobs. Additional services have been implemented to facilitate starting and scheduling jobs on different machines based on availability and machine features. This one is not used in this work, as DCSim has its own HTCondor-like scheduler, see Section 3.3.

Storage Services The Storage Services model simulates lookup and delete requests, as well as read and write requests. For the latter, the data transferred over the network is also simulated, including a configurable data buffer.

Jobs The job bundles global input files with the actual tasks to be done, including individual input and output files. Generally, the job then consists of reading the input file, performing computations on the data read, and finally writing the output data. In addition, dependencies between tasks can be defined.

More fine-grained control can be achieved with compound jobs, which are used in DCSim. These jobs consist of individual actions such as basic file operations, computation,

[2] `https://wrench-project.org/`

and sleep, but custom actions can also be defined. Custom actions can be used to pass messages similar to those in the real grid.

Execution Controllers The execution controllers provide more abstract functionality by hooking into chain activities. Thus making it possible, on the one hand, to create managers for job creation, execution, and data movement, and, on the other hand, to create monitoring to measure usage of the various resources.

3.3. DCSim

At the top of the software stack, extensions to Wrench have been developed and will be described in the paragraphs below. These extensions have been developed with WLCG-like workflows in mind, but can be used more generally. The whole software stack has been unified under the name DCSim[3].

Like on the real WLCG there are two kinds of jobs, as described below, in terms of the way they transfer input files. Because of their different data-transfer dynamics, they need to be modelled separately.

Copy Jobs For Copy Jobs, all input data is first copied to the node on which it will be processed. After which, it will be processed. Output data is copied back after the job is done.

Streaming Jobs Due to the discrete nature of HEP events, input data can be split into blocks, which can then be processed individually. To reflect this, Streaming Jobs have been implemented. Each block of input data is read and processed. Afterwards, the output data is written. The computation time for each block is assumed to be proportional to its size.

Caching Action Caching is implemented using Jobs' custom actions. Caching works by requesting data through a proxy that acts as the cache server. Cache servers within the specified scopes are queried to determine whether the input data is available. If it is available, data is transferred from the cache. Otherwise, the data must be transferred from another storage server and saved on the cache server for subsequent requests. When the cache runs out of space, the data with the most distant access date is evicted.

[3] `https://github.com/HEPCompSim/DCSim`

Workload Execution Jobs, streaming or not, can be bundled in workloads. Specifically, workloads execute the same code but on different input data. During the simulation, a Workload Execution Controller (see Section 3.2) is spawned and interacts with DCSim's own Scheduler Service, see the next paragraph.

Scheduler Service The Scheduler Service keeps a list of all active jobs and all compute hosts in the simulation and interacts with the Workload Execution Controller. When a task's submission time is reached inside the simulation, it calls the Workload Execution Controller to create the job. Based on the job's CPU core and RAM requirements, it searches for free resources. When a match is found, the job is submitted and moved from the list of active jobs to the submitted jobs list. After the job terminates, it calls back to the Scheduler and is removed from the list of submitted jobs, freeing its memory.

This closely follows how real scheduling systems operate. On top of that, it has the benefit that jobs only use RAM when they are running in the simulation, so only when they are actually needed. By doing that, RAM usage in the simulation is significantly reduced.

Monitoring The existing monitoring does not capture all desired information and consumes excessive resources in large simulations. So the monitoring has been replaced.

The monitoring stores a job identifier, the virtual host machine name, job start and end times, cache hit rate, and processing and file transfer durations.

3.4. Complexity

Simulations must be completed within a specified time. For example, it cannot take longer than the time allocated for this work. The simulation time and RAM usage scale with the simulation's complexity. Thus, simulating parts of the grid can become unfeasible. To resolve this problem, the simulation can be scaled down. But the underlying structure of the simulation has to stay the same. For this number of jobs, the number of CPUs, the amount of RAM, and the network speed must be scaled by the same factor. Also, the network's structure has to stay the same. Scaled-down simulations have been shown to yield results that do not differ significantly from those of their full-sized counterparts [5].

Scaling down has its limits: If the number of jobs on a given system is too low, thus not filling the queue, the dynamics of the system change. Another problem arises because the number of jobs on a system is an integer. So scaling down beyond that point would make it impossible for that system to run a job. Combining multiple systems, such as

storage and compute, can also reduce complexity. But again, this can change the system's dynamics, as the network is an influential component of the grid.

4. Configuration of the simulation

The simulator DCSim alone provides modules for simulating arbitrary systems. The modules have to be configured to model the desired system. This is done using configuration files, as described in Section 4.3. The origin of the data is described in Sections 4.1 and 4.2, where the former concerns this work's simplified version of the WLCG, and the latter concerns the workloads running on this grid.

4.1. Computing Infrastructure

In the context of this work, WLCG is reduced to the sites DESY, KIT and GoeGrid. DESY and KIT are limited to their use as storage providers for GoeGrid. The performance of GoeGrid, amongst other things, depends on the time required to transfer the input files required for a workload. This is because for copy jobs, CPU time is wasted waiting for the file transfer to complete before any computation can start. For streaming jobs as well, time would be wasted if the transfer of input files cannot keep up with the computation. Simplifying further, each compute, or storage cluster, is modelled as a single computer with all the CPUs, RAM, and Storage Capacity.

Network Bandwidth Usable Network Bandwidth for GoeGrid and EMMY to Göttingen is $2 \times 25\,\text{Gb/s}$ from DESY and $100\,\text{Gb/s}$ from KIT, although there are two $25\,\text{Gb/s}$ connections between DESY and Göttingen, the dynamics of the network choose one connection over the other effectively allowing only $25\,\text{Gb/s}$ of bandwidth to pass through anyways. In Göttingen, the GoeGrid and EMMY compute clusters share a common firewall with a bandwidth of $40\,\text{Gb/s}$. After which, there is a second firewall for EMMY only with a bandwidth of $1\,\text{Gb/s}$. Between the storage of GoeGrid and EMMY there is a $100\,\text{Gb/s}$ link without an extra firewall. From the common firewall in Göttingen to GoeGrid, there are no more relevant bottlenecks in the network. The same applies to the connection between GoeGrid's storage and compute clusters. The Network Bandwidths connecting with relevant components are shown in Figure 4.1.

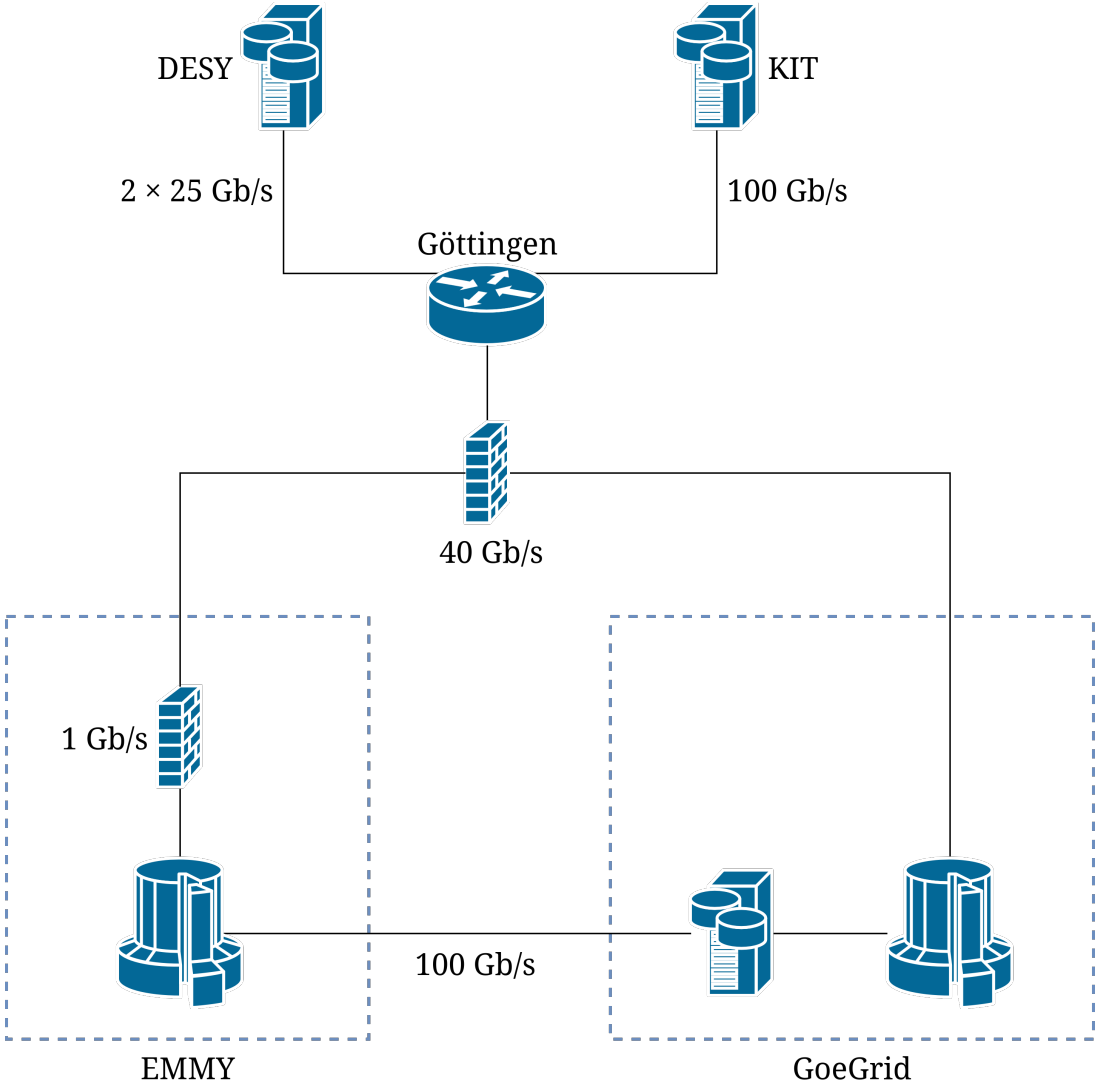

Figure 4.1.: A simplified sketch of this work's model, limited to the components relevant to this work. Drawn using `https://app.diagrams.net/`.

Storage Speed and Capacity In addition to Network Bandwidth, the read and write speeds of drives on storage servers can create another potential bottleneck. For every site's storage cluster, the total bandwidth is assumed to be 40 Gb/s. As for Storage Capacity, it is assumed that the storage clusters have sufficient capacity to accommodate all files inside them.

Computing Capabilities For the Computing Capabilities of GoeGrid and EMMY[1] the total amounts of CPU cores and RAM, and the average HEPSPEC per CPU core, are listed in Table 4.1.

[1]From EMMY only `standard96s` nodes from `Phase 3` are used for the WLCG. Official Documentation: `https://docs.hpc.gwdg.de/how_to_use/compute_partitions/cpu_partitions/index.html`.

Table 4.1.: Total number of logical CPU cores, total RAM, and average HEPSPEC per CPU for GoeGrid and EMMY in the context of WLCG. For GoeGrid, the number of CPU cores is shared with ATLAS, since GoeGrid is not exclusive to ATLAS. For EMMY, this is the total number of CPU cores, not just those used by ATLAS.

Site	#CPUs	RAM	avg. HEPSPEC / CPU
GoeGrid	11104	32561 GB	10.6
EMMY	21120	113080 GB	15.1

4.2. Workloads

For this work, monitoring data from PANDA, spanning from 2023-08-01 until 2026-06-07, is used. This monitoring data comprises many fields, but only those listed below are relevant to this work. Workloads running inside the simulation will be characterised by these past jobs.

`wall_time` Time the job took to run, measured in seconds.

`processingtype` Type of the job. Amongst other things, these types indicate whether the job is a copy job or a streaming job. `processingtype`s starting with `gangarobot` are test jobs and thus filtered out. `processingtype`s starting with `panda-client` are analysis jobs and thus renamed to `analysis`. Every other `processingtype` is left as is. Jobs with a `processingtype` of `analysis` are streaming jobs, and all others are copy jobs.

`computingsite` The PANDA queue name the job is a part of. Jobs are filtered such that the `processingtype`s starts with either `GoeGrid` or `EMMY`, resulting in the `processingtype`s of `GoeGrid`, `GoeGrid_LODISK`, `GoeGrid_NHR`, `EMMY`, `EMMY_BKG`, `EMMY_DESY` and `EMMY_KIT`.

`jobstatus` The status the job terminated as. It can indicate success or some kind of failure. Here, `jobstatus` is filtered for successful jobs with a value of `finished`.

`corecount` The number of CPUs assigned to the job.

`cpuconsumptiontime` The compute time used across all CPUs, measured in seconds. This excludes the idle time of CPU cores, for example, due to data staging.

`minramcount` Amount of RAM assigned to the job, with corresponding unit stored under `minramunit`.

`inputfilebytes and outputfilebytes` Total size of all input and output files, measured in bytes.

4.3. Configuration Files

To run the simulator DCSim, three files, namely the workload and dataset configurations and the platform file, must be provided, as described in the following subsections.

4.3.1. Platform File

The Platform File represents the grid, from Section 4.1, on which jobs run. The file is formatted as XML.

In this file, the GoeGrid and EMMY compute clusters are represented as worker hosts, while the storage clusters of GoeGrid, DESY and KIT are represented as storage hosts. The compute clusters are assigned their corresponding values for `core` (the number of CPUs), `ram` (the amount of RAM), and `speed` (the average HEPSPEC per CPU). For the storage clusters, not relevant values are assigned to `speed` and the other fields used in the compute clusters are left undefined. But `read_bw` and `write_bw` are set to their corresponding storage read and write speed.

Furthermore, the connections between clusters are represented as links with either their corresponding bandwidth, if relevant, or with an arbitrarily large value. Links connect clusters and routers via routes that describe the source and destination.

Samples of the Platform File are shown in Listing 4.1.

Listing 4.1: Platform File samples.

```
<host id="EMMYCompute" speed="15.1f" core="21120">
  <prop id="type" value="worker"/>
  <prop id="ram" value="113080GB"/>
</host>
<link id="EMMYComputeGoeFirewall" bandwidth="1Gbps" latency="0us"/>
<route src="EMMYCompute" dst="GoeFirewall">
  <link_ctn id="EMMYComputeGoeFirewall"/>
</route>
```

4.3.2. Workload and Dataset File

The Workload and Dataset Files correspond to the jobs explained in Section 4.2. Corresponding Monitoring Data columns and Configuration File keys are listed in Table 4.2. The required unit conversion and processing are described in the following paragraphs.

Workload File The Workload File is a JSON dictionary, in which every workload is a key. Every workload key corresponds to another dictionary characterising the workload.

Listing 4.2: Workload Configuration File example.

```
"copy.GoeGrid": {
 "infile_datasets": [ "copy.GoeGrid" ],
 "submission_time": 10,
 "workload_type": "copy",
 "num_jobs": 70,
 "cores": { "type": "histogram", "counts": [], "bins": [] },
 "memory": { "type": "histogram", "counts": [], "bins": [] },
 "flops": { "type": "histogram", "counts": [], "bins": [] },
 "outfilesize": { "type": "histogram", "counts": [], "bins": [] }
}
```

Each workload is assigned an input dataset (`infile_datasets`) with the same name. The submission time (`submission_time`) is set to 10 s. The workload type (`workload_type`) is set to either `copy` or `streaming`, corresponding to copy and streaming jobs, respectively. For `num_jobs`, the ratio of a specific job to the total is calculated, then multiplied by the desired total number of workflows for the simulation. Figure 4.2 shows the shares of all workloads. Before binning `cores`, `memory`, `flops` and `outfilesize`, the unit of `memory` has to be converted to bytes and for `flops`, the `cpuconsumptiontime` has to be multiplied by the corresponding cluster's average HEPSPEC per CPU. The unit of `outfilesize` must remain bytes. For binning, NumPy's `histogram`[2] is set to `auto`, except for `cores`, where there is a bin for each integer between the minimum and maximum number of CPUs found in the monitoring data.

An example workload from the Workload File is shown in Listing 4.2. A histogram of all workloads by RAM amount is shown in Figure 4.3, and the rest is shown in Appendix A.

[2]Official Documentation:
`https://numpy.org/doc/stable/reference/generated/numpy.histogram.html`

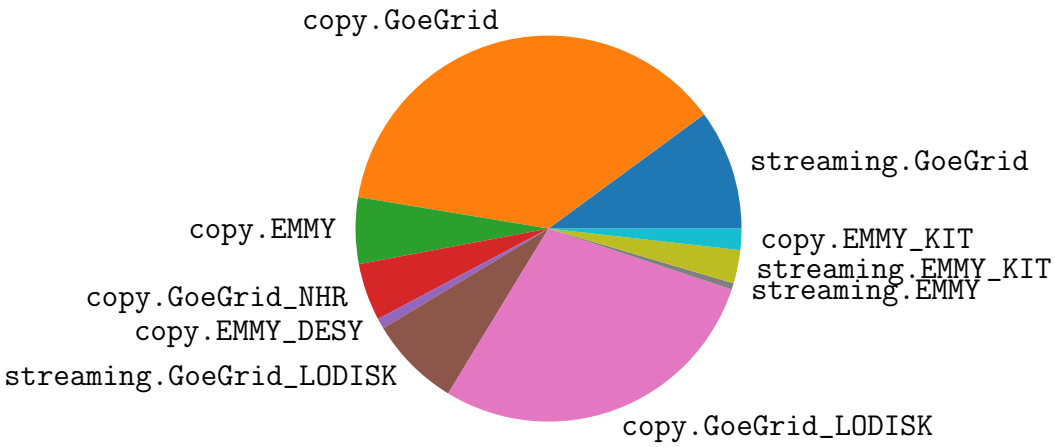

Figure 4.2.: Shares of all workloads

Dataset File Similar to the Workload File, the Dataset File is a JSON dictionary in which each dataset is a key. Each dataset key corresponds to a dictionary entry characterising the dataset.

For each dataset, the `location` is the cluster, as defined in the platform file, that stores it. `num_files` is the same number as `num_jobs` in the Workload File; that way, each workload is assigned one dataset. `filesize` is processed analogous to `outfilesize` in the Workload File.

An example dataset from the Dataset File is shown in Listing 4.3 and a histogram of all the `filesizes` is shown in Apendix B.

Listing 4.3: Dataset Configuration File example.

```
"copy.GoeGrid": {
 "location": [ "GoeGridStorage" ],
 "num_files": 70,
 "filesize": { "type": "histogram", "counts": [], "bins": [] }
}
```

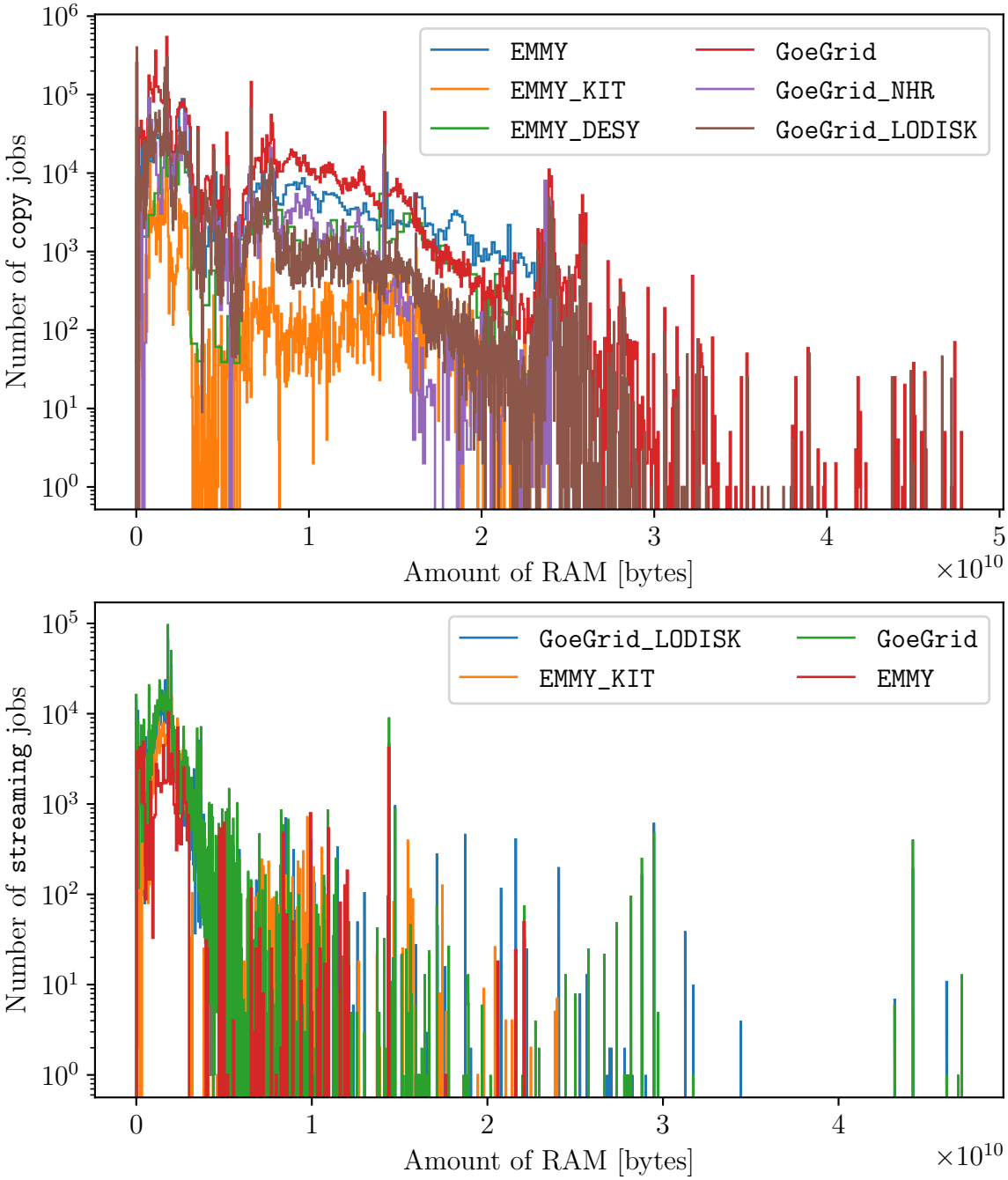

Figure 4.3.: Number of jobs with a specific amount of RAM for different PanDA queues

Table 4.2.: Relevant column names in the monitoring data and their corresponding keys in the configuration files. Unit conversion and processing are still required.

Monitoring Data Column	Configuration File Key	Configuration File
`processingtype`	`workload_type`	Workload
`corecount`	`cores`	Workload
`minramcount`	`memory`	Workload
`cpuconsumptiontime`	`flops`	Workload
`outputfilebytes`	`outfilesize`	Workload
`inputfilebytes`	`filesize`	Dataset

5. Result

When the simulator DCSim[1] runs with the configuration files from Chapter 4, the program crashes with a segmentation fault. In Section 5.1, the error is traced back to its source.

5.1. Identifying the Error

Searching for the problem is simplified by setting the desired total number of workloads to three. Due to rounding errors, two workloads are created. One is from the `GoeGrid` queue, and the other is from the `GoeGrid_LODISK` queue, both of type copy. In addition to reducing the desired total number of workloads, DCSim is set to a higher verbosity level[2].

The last entries in the log that appear before the crash are shown in Listing 5.1. These lines reveal a problem: the Configuration File specified one `GoeGrid` and one `GoeGrid_LODISK` workload, but according to the log, the same `GoeGrid_LODISK` workload is submitted twice.

Listing 5.1: DCSim log before the crash.

```
[...] There are 1 jobs to schedule at time 10.000000
[...] Submitting job job_copy.GoeGrid_LODISK_0 [...]
[...] There are 1 jobs to schedule at time 10.000000
[...] Submitting job job_copy.GoeGrid_LODISK_0 [...]
[...] Job Manager got a wrench::JobManagerWakeupMessage message
```

The line `Submitting[...]` in the log is from `WorkloadExecutionController.cpp` at line 142. From there, the problem is traced back: The function to which this line belongs is named `createAndSubmitJob` and is also called twice with the `GoeGrid_LODISK` workload. The caller of the previous function is named `schedule` and starts in `JobScheduler.cpp` at line 50. In the context of the function `schedule`, the current object `this` refers to the same object in both calls. The function `schedule` is called in the function

[1] DCSim is built using commit hash `bafdcc435525f79cfa2fa75b691f52d4051a6d95` from the `fix-install` branch.

[2] This is done using the `-wrench-full-log` argument.

`main` of `WorkloadExecutionController.cpp` at line 215, using `this->job_scheduler->schedule()`. The job scheduler being the same is not the root problem. In line 206, both workloads are added to its array of execution controllers, using `this->job_scheduler->addExecutionController(this)`

Going back to the function `schedule`, in `JobScheduler.cpp at line 57`, of which an extract is shown in Listing 5.2, a loop iterates over the execution controllers. The first time the function `schedule` is called, there is only the execution controller for the `GoeGrid_LODISK` workload to iterate over. The second time the function `schedule` is called, it iterates over the `GoeGrid_LODISK` and `GoeGrid` workloads execution controllers. The execution controller for the `GoeGrid_LODISK` workload had already been processed when the function `schedule` was first called, and its memory was freed. Thus, when the function `schedule` is called a second time, it tries to access freed memory, resulting in a segmentation fault.

Listing 5.2: Looping over freed memory.

```
// Go through the workload execution controllers in order
for (auto const &ec: this->execution_controllers) {
// TODO: Remove the execution controller from the list
[...]
ec->setJobSubmitted(job_name);
```

The comment at line 58 notes that removing the execution controllers from the array is still pending. Editing the code to simply remove processed execution controllers from the array within the same function also crashed the program, indicating a larger issue.

This means that doing simulations as this work set out to do is not possible until these problems are fixed.

6. Outlook

Before further work can be conducted, the issue described in Chapter 5 and any possible follow-up problems must be resolved. After the aforementioned problems are resolved, further developments to DCSim, as described in Section 6.1, can be made, and further research, as described in Section 6.2, can be conducted.

To conduct further research, a measurement of simulation quality is needed. One comparable attribute shared by both the measured and simulated jobs is wall time. More about wall time in, see Section 4.2. Thus, the simulated wall time can be compared to the measured wall time, returning a measure of accuracy.

6.1. Improvements to DCSim

A flaw in the current version of DCSim is that workloads cannot be assigned to sites, unlike datasets. Thus, real jobs cannot be accurately represented in the simulation, as they might end up on a different compute cluster than intended. With this feature added to DCSim, an improvement to simulation accuracy is therefore expected.

6.2. Further Research

This work was not able to accomplish its original goal, therefore further research is needed. Starting points for further research are listed below.

Calibration Not every detail can be modelled accurately in the simulation. For example, discrepancies in the Platform File, such as bandwidths, can be adjusted to account for unknown bottlenecks. Another example is changing the number of CPU cores a site has in the platform file, since neither GoeGrid nor Emmy is used exclusively by Wlcg.

Varying the Model After calibration, estimates of the performance of possible new arrangements around Emmy can be made. For example, increasing the bandwidth of

Emmy's firewall or increasing the capacity of a new storage server to identify the point at which performance gains become negligible.

7. Summary

This work set out to simulate possible new arrangements in and around Emmy, as in the future, without GoeGrid, will not be able to run as a meaningful part of the Wlcg. These arrangements included a firewall with more bandwidth or a new caching server. It has not managed to accomplish this, as a bug in the simulator DCSim prevented research from proceeding within the time allotted for this work.

A dataset of PanDA monitoring data was acquired and processed into configuration files for the simulator DCSim, and a platform file representing the part of the Wlcg that is in and connected to Göttingen was also created. From there, no meaningful simulations have been able to run.

A. Workload Configuration Attributes

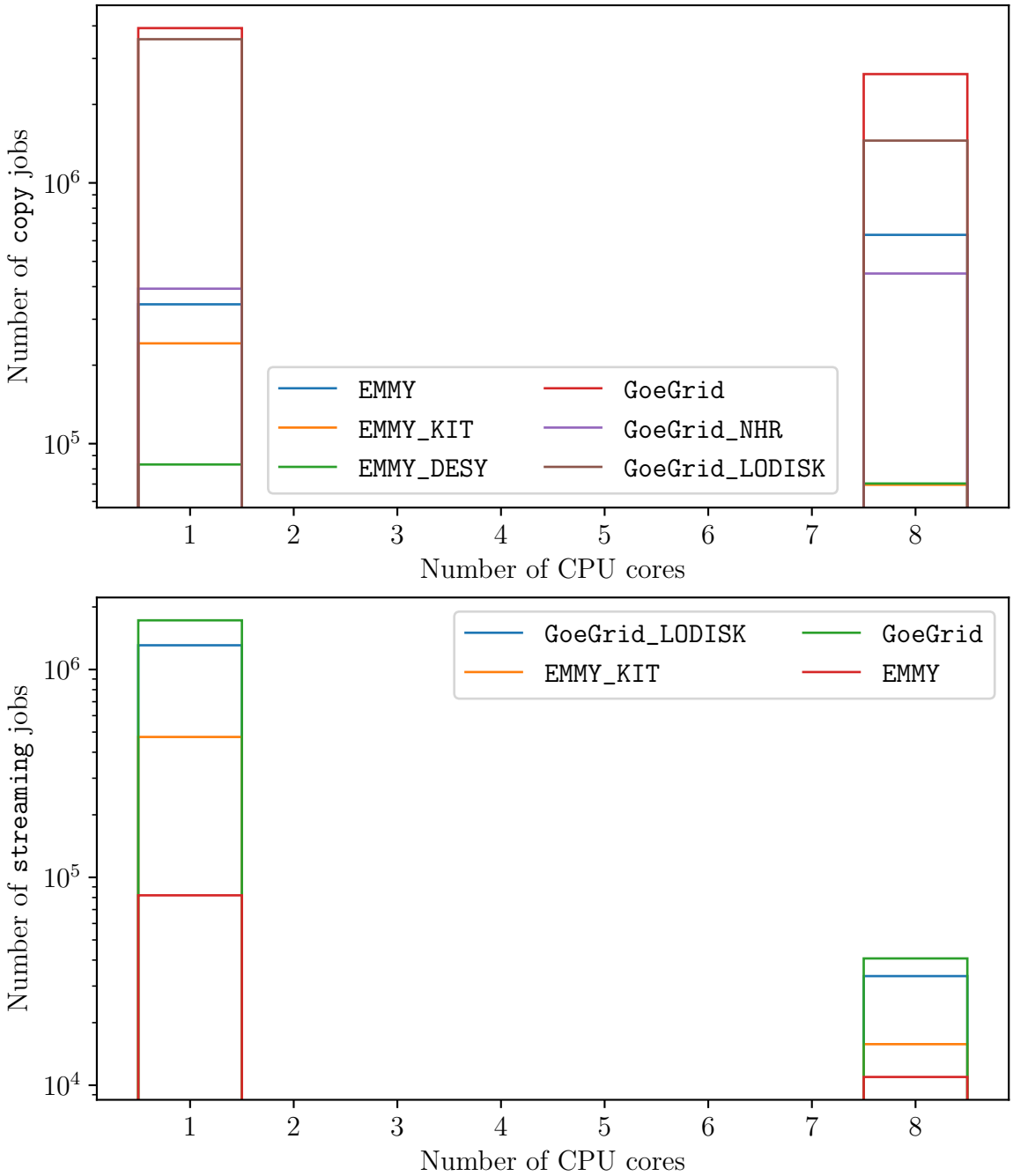

Figure A.1.: Number of jobs with a specific number of CPU cores for different PanDA queues

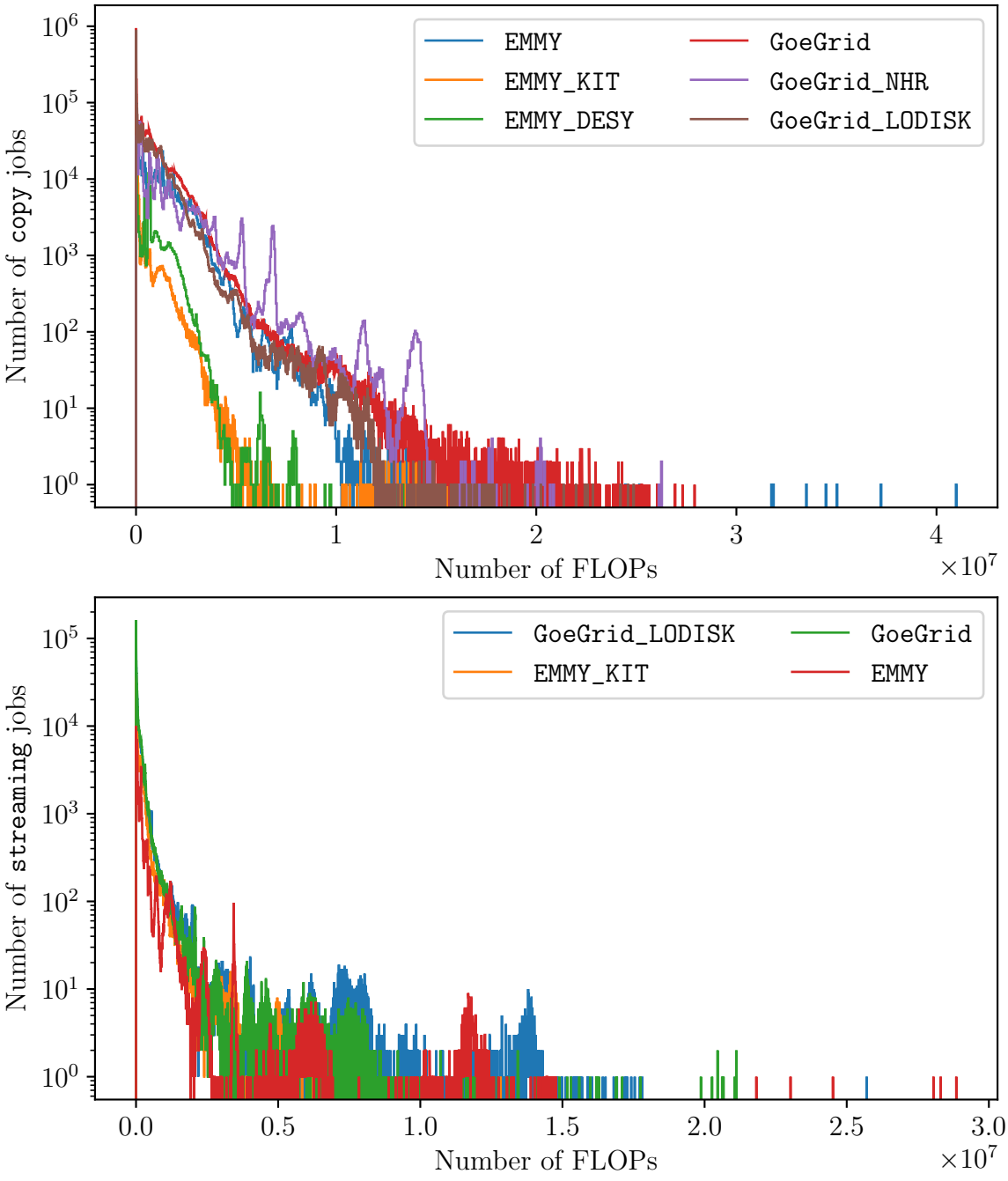

Figure A.2.: Number of jobs with a specific number of FLOPs for different PanDA queues

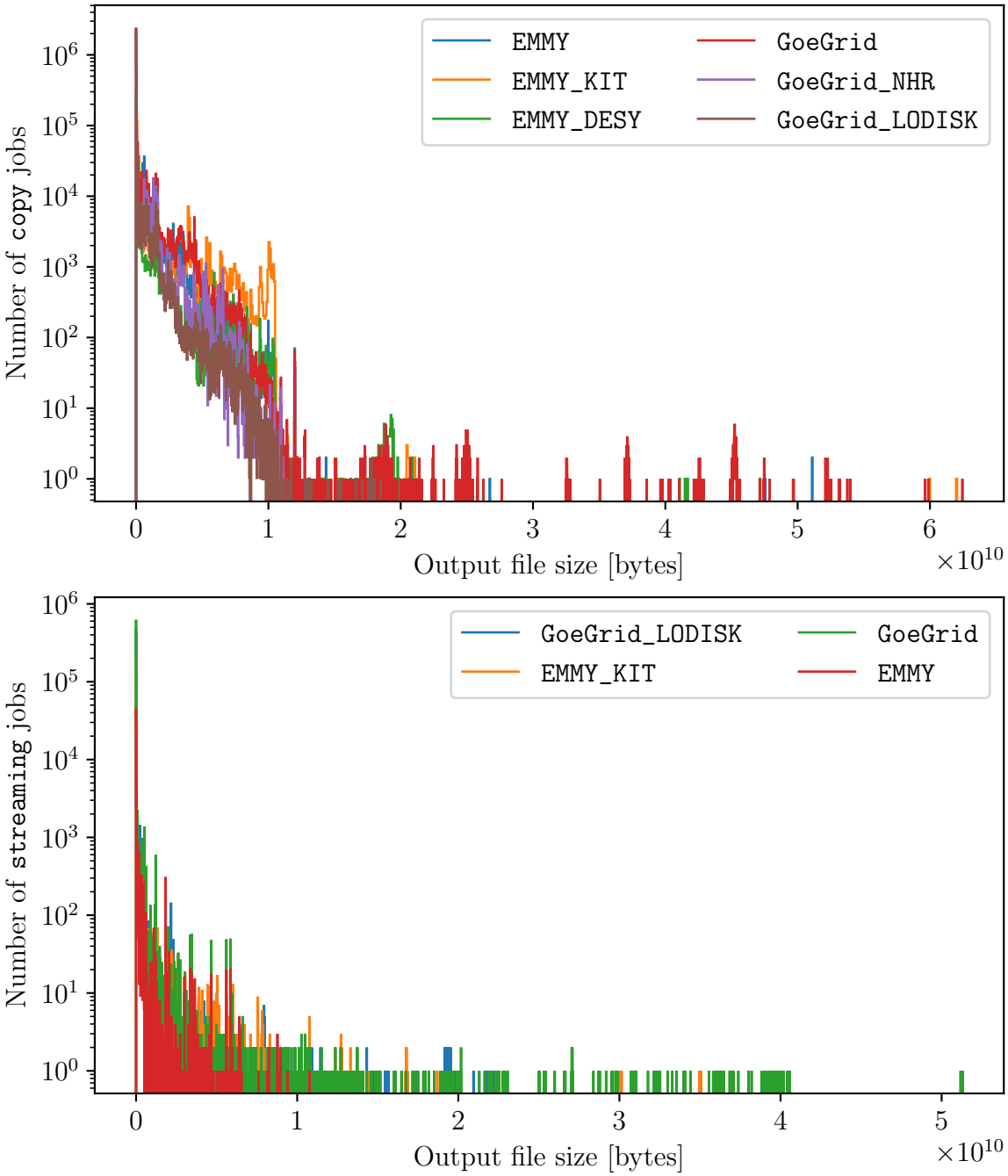

Figure A.3.: Number of jobs with a specific amount of total filesizes of output files for different PanDA queues

B. Dataset Configuration Attributes

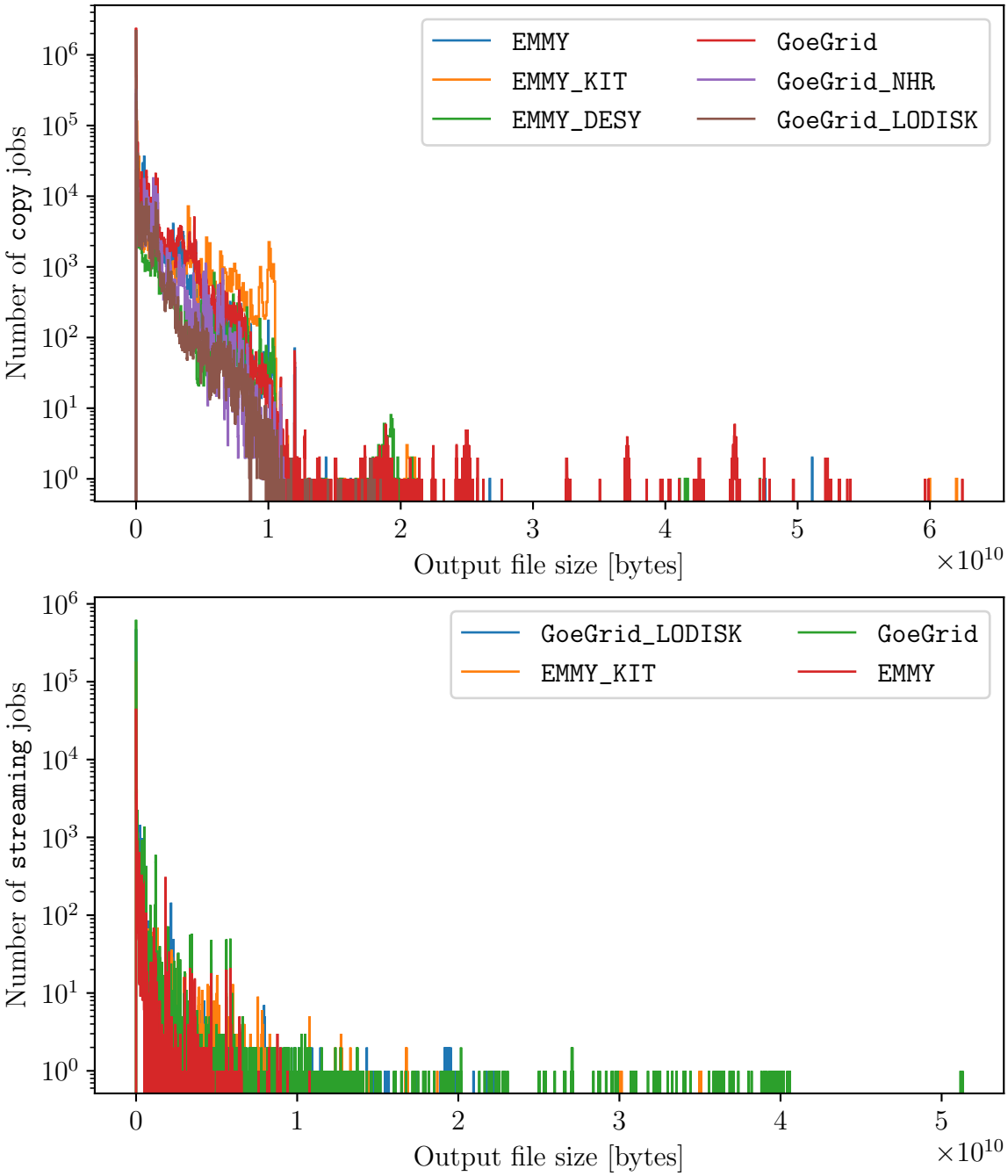

Figure B.1.: Number of datasets with a specific total filesize of input files for different PanDA queues

Bibliography

[1] G. Aad, et al. (ATLAS), *Software and computing for Run 3 of the ATLAS experiment at the LHC*, Eur. Phys. J. C **85(3)**, 177 (2025)

[2] M. Titov, et al., *Advanced Analytics service to enhance workflow control at the ATLAS Production System*, EPJ Web Conf. **214**, 5 (2019)

[3] M. Barisits, et al., *Rucio: Scientific Data Management*, Comput. Soft. Big Sci. **3**, 19 (2019)

[4] P. Velho, et al., *On the validity of flow-level tcp network models for grid and cloud simulations*, ACM Trans. Model. Comput. Simul. **23(4)** (2013)

[5] M. Horzela, et al., *Modeling Distributed Computing Infrastructures for HEP Applications*, EPJ Web Conf. **295**, 8 (2024)

Acknowledgements

I would like to thank everyone who has helped me reach this far 💚

Erklärung nach §13(9) der Prüfungsordnung für den Bachelor-Studiengang Physik und den Master-Studiengang Physik an der Universität Göttingen:

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe.
Darüberhinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, im Rahmen einer nichtbestandenen Prüfung an dieser oder einer anderen Hochschule eingereicht wurde.

Göttingen, den 4. August 2026

(Adrian Leone)