Exploring Scheduling for On-demand File Systems and Data Management within HPC Environments

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

Dipl.-Inf. Mehmet Soysal
geb. in Heidelberg

Tag der mündlichen Prüfung:	21.12.2020
Hauptreferent:	Prof. Dr. rer. nat. Achim Streit
Korreferent:	Prof. Dr. rer. nat. Martin Frank

Eidesstattliche Erklärung

Erklärung zur selbständigen Anfertigung der Dissertationsschrift
Hiermit erkläre ich, dass ich die Dissertationsschrift mit dem Titel

Exploring Scheduling for On-demand File Systems and Data Management within HPC Environments

selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Regeln zur Sicherung guter wissenschaftlicher Praxis am Karlsruher Institut für Technologie (KIT) beachtet habe.

Karlsruhe, 14.3.2021	*Mehmet Soysal*
Ort, Datum	Name

To my family

Acknowledgement

This work would not have been possible without the help of a number of important people, whom I would like to thank warmly at this point. The thanks that go to these wonderful people can hardly be put into words and you will always be in good memories for me.

First of all I would like to thank Prof. Dr. Achim Streit for his supervision and Prof. Dr. Martin Frank for his support and co-supervising this work.

I would especially like to thank Prof. Dr. Achim Streit for giving me the opportunity to do my doctoral thesis under his supervision. It should be mentioned that before I already worked for several years as a system administrator at SCC. I have had the desire to pursue a PhD for quite some time. Therefore, I am very grateful that Prof. Streit has expressed his trust in me and made this opportunity possible. During the four and a half years of my doctoral thesis you were at my side with valuable advice. You always kept the quality of my research in mind and enabled me to publish my work. I would also like to express my deep appreciation to Prof. Dr. Martin Frank. Your continuous support of my doctoral studies have always given me a good feeling and that my work is recognized.

I am especially grateful to Dr. Marco Berghoff for all the time, patience (especially) and energy he has put into supervising my work. Without him, my work would probably still be in its early stages. His patience in reading and correcting my papers cannot be expressed in words. His suggestions to improve my “creative” plots are always in good memory. I will also not forget that he never let me down, despite my stubbornness and always found uplifting words. And not unmentioned should be that he is the only LaTeX artist I know and a real virtuoso in dealing with it, thanks a lot for your support in dealing with my LaTeX problems.

I would also like to thank colleagues, who gave me important information and contributions for my work. Thorsten Zirwes, Michael Weimer, Samuel Braun, Marco Bergoff, Marie Weiel and Markus Götz which contributed their use cases to confirm the arguments of my work. Also I want to thank Doris Ressmann and Roland Laifer which provided detailed information about storage systems and procurements. In particular I was able to talk to Roland about storage systems and parallel file systems. With his experience he was always a good colleague to talk to and he was a great support in the collecting key data in the domain of parallel file systems. I would also like to thank Richard Walter, who sacrificed his time and helped to do the tests regarding the network topologies.

Thanks again Marco Berghoff and Thorsten Zirwes for your time, which you both spent to proofread my work and to note only “very few” corrections ;).

I would also like to thank all the department heads I had in this department:
Prof. Dr. Rudolf Lohner, Dr. Holger Marten, Dr. Olaf Schneider and now more recently Dr. Jennifer Buchmüller for always having an open door for discussions. In particular, I would like to mention Horst Gernert and how he always took care of employees in his position as deputy head of department. Hope you enjoy your retirement.

Special thanks goes to: Maren Hattebuhr for being the most jolly and cheerful person, and for the weird singing during the boring car rides, Thorsten Zirwes for his "wonderful" cinema film decisions in which he sometimes dragged us along. Karl-Heinz Schmidmeier and Frauke Bösert for beeing really wonderful colleagues, Rolf Mayer und Ngan Long Tu for the best coffee to scrounge in the Campus South building, Markus Götz for having a open ear when needed and for being one of the friendliest and helpful colleagues. Elnaz Azmi and Markus Strobel for the nice company on the way to work and that Markus gave me a taste for high quality rums.

I am also thankful to all my current and former colleagues, for countless enjoyments, lively lunch break discussions, which created a warm and inspiring atmosphere. Many colleagues and doctoral researchers with whom I have spent a lot of free time outside work, from social evenings to sports activities. During my time as a doctoral researcher, my attitude changed and I often got a different perspective on a lot of things. This is due to the colleagues with whom I spent a lot of time, both on and off the job. So I would like to thank these colleagues again for the great time. Also thank you very much: Thorsten Zirwes, Michael Weimer, Samuel Braun, Marie Weiel, Markus Götz, Eileen Kühn, Markus Fischer, Maren Hattebuhr, Matthias Schnepf, Kirsten Wohak, Benjamin Ertl, Nico Schlitter, Marco Berghoff, Diana Gudu, Peter Krause, Matthias Schnepf, Karl-Heinz Schmidmeier, Frauke Bösert, Roland Laifer, Hartmut Häfner, Nikolai Pfisterer, Elnaz Azmi, Markus Strobel, Elizaveta Dorofeeva and everyone else I forgot to mention.

Zusammenfassung

Immer mehr wissenschaftliche Bereiche verwenden zunehmend High-performance computing (HPC)-Systeme. Computersimulationen sind oft kostengünstiger als echte Experiemte und ermöglichen einen Einblick, welcher in der realen Welt oft nicht verfügbar ist. Als Beispiele wären Crashtests in der Automobilindustrie, Erdbebenforschung, oder die aktuelle SARS-Covid-19 Forschung zu nennen. Es ist weitaus günstiger eine Crashtest zu simulieren, als ein Auto gegen eine Wand fahren zu lassen. Im Falle der Erdbebenforschung wird hier eine Möglichkeit gegeben, einen Einblick in die Erde zu bekommen, welche sonst nicht möglich wäre. Aufgrund der aktuellen Lage bzgl. SARS-Covid-19 werden verstärkt Supercomputer genutzt, um Erkenntnisse über den Virus zu gewinnen.

HPC-Systeme und ihre Anwendungen unterliegen einer stetigen Veränderung. Die Systeme werden immer größer, heterogener und damit komplexer. Darüber hinaus sind neue Anwendungsfälle und Nutzungsprofile entstanden. Der Wandel bei den Anforderungen lässt sich anhand der Nutzung der Systeme erkennen. Bei den traditionellen HPC-Anwendungen handelt es sich primär um rechenintensive, parallele und skalierende Simulationen. Dabei ist es üblich Datenoperation soweit wie möglich zu reduzieren. In jüngster Zeit gibt es neue Nutzer Communities, welche riesige Datenmengen erzeugen, verarbeiten und analysieren. Dieser Trend wird als daten-intensives Rechnen bezeichnet. HPC Systeme und die bisherigen parallelen Speichersysteme sind gewöhnlich nicht für diese neuartigen Workloads ausgelegt. Zusätzlich haben immer mehr wissenschaftliche Communities einen erhöhten Kapazitätsbedarf, welcher sich mit den parallelen HPC-Filesystemen nur mit enormen Kosten realisieren lassen.

Aus der bisher geschilderten Situation ergeben sich gleich mehrere Herausforderungen. (i) Das Speichersystem muss eine sehr hohe Spitzenleitung bereitstellen, sowohl beim Durchsatz als auch bei der Metadaten Leistung. (ii) Die Speichersysteme müssen effizienter ausgenutzt werden, um mehr freie Kapazität auf den schnellen Speichersystemen zu haben.

In meiner Arbeit habe ich unter diesen Rahmenbedingungen, welche in HPC Zentren vorhanden sind, Konzepte erarbeitet. Für die unter (i) geschilderte Herausforderung werden dynamisch per Software erzeugte Dateisysteme (On-Demand-Dateisystem) genutzt. Für die Herausforderungen unter (ii) wurde ein Konzept zum Datenmanagement entworfen. Dadurch können die verschiedenen Hierarchieebenen stärker miteinander gekoppelt und erweitert werden. Für Benutzerinnen der HPC-Systeme wird durch das Konzept eine transparente Nutzung der verschiedensten Speichersysteme ermöglicht. Die

HPC-Betreiberin hat hierbei die Möglichkeit, neue Speichersysteme und -technologien transparent zu integrieren.

In meiner Dissertation präsentiere ich Strategien und Leistungsuntersuchungen in einer Simulationsumgebung, um die oben genannten Herausforderungen zu adressieren. Teile der Dissertation sind im Rahmen des DFG-Projektes ADA-FS[1] entstanden, welches ein Teil des Schwerpunktprogramms Software for Exascale (SPPEXA) [1] war. ADA-FS zielt darauf ab, die I/O-Leistung für hoch-parallele Anwendungen durch verteilte On-demand-Dateisysteme zu verbessern. Dazu wurde erforscht, wie jobspezifische temporäre Dateisysteme effizient für HPC-Umgebungen bereitgestellt werden können.

[1] Advanced Data Placement via Ad-hoc File Systems at Extreme Scales

Abstract

More and more scientific disciplines are increasingly using HPC systems. Computer simulations are often less expensive than real experiments and provide insight that is often not available in the real world. Examples are crash tests in the automotive industry, earthquake research, or the current SARS-Covid-19 research. It is far more economical to simulate a crash test than to let a car crash into a wall. In the case of earthquake research, this is a way to get a view of the earth that would otherwise not be possible. Due to the current situation regarding SARS-Covid-19, supercomputers are used to gain knowledge about the virus.

High-performance computing (HPC) systems and their applications are subject to constant change. The systems are becoming larger, more heterogeneous and thus more complex. In addition, new use cases and usage profiles have emerged. The change in requirements can be seen in the use of the systems. The traditional HPC applications are primarily compute-intensive, parallel and scalable simulations. It is common practice to reduce data operations as much as possible. In addition, there have recently been new user communities that generate, process and analyze huge amounts of data. This trend is called data-intensive computing. HPC systems and existing parallel storage systems are usually not designed for these new types of workloads. In addition, more and more scientific communities have an increased capacity requirement, which can only be realized with the parallel HPC file systems at enormous costs.

Several challenges arise from the situation described above. (i) The storage system must provide a very high peak performance, both in terms of throughput and metadata performance. (ii) The storage systems must be used more efficiently to provide more free capacity on the fast storage systems.

In my work, I have developed concepts under these conditions, which are available in HPC centers. For the challenge described under (i), dynamically software-generated file systems (on-demand file system) are used. For the challenges under (ii) a concept for data management was designed. This allows the different hierarchical levels to be more closely linked and extended. For users of HPC systems, the concept enables transparent use of the most diverse storage systems. The HPC operator has the opportunity to integrate new storage systems and technologies transparently.

In my dissertation I present strategies and performance tests in a simulation environment to address the above mentioned challenges. Parts of the dissertation are part of the

DFG project ADA-FS[1], which was part of the priority program Software for Exascale (SPPEXA) [1]. ADA-FS aims to improve I/O performance for highly parallel applications through distributed on-demand file systems. For this purpose, research was conducted on how to efficiently provide job-specific temporary file systems for HPC environments.

[1]Advanced Data Placement via Ad-hoc File Systems at Extreme Scales

Table of Content

Glossary

Glossary & Acronyms

AVX Advanced Vector Extensions (AVX) is an extension of the x86 instruction set for microprocessors from Intel and AMD..

CBB Constant Bisectional Bandwidth.

DFG The Deutsche Forschungsgemeinschaft (DFG) (English: German Research Foundation) is a German research funding organization.

FCFS First Come First Serve (FCFS) is a scheduling algorithm that executes tasks in order of their arrival. It is an easy and simple scheduling algorithm.

HCA The Host Channel Adapter (HCA) is a component of the InfiniBand that serves as the connecting component between the processors and external peripherals. The HCA has direct access to processors and memory and is responsible for data transport..

HPC High-performance computing.

IEC The International Electrotechnical Commission (IEC) is an international standardization organization for standards in electrical engineering and electronics..

LHC The Large Hadron Collider (LHC) is a particle accelerator at the European Nuclear Research Centre CERN near Geneva..

MPI Message Passing Interface (MPI) is a standardized and portable standard to function on a variety of parallel computing architectures.

MPP A Massively Parallel Processing (MPP) System is a parallel computer that has a large number (thousands) of independent execution units..

NAStJA Neoteric Autonomous Stencil code for Jolly Algorithms.

NVMe Non-volatile memory express (NVMe) is an logical-device interface specification for accessing non-volatile storage media. Storage devices can have serveral physical form factors, including solid-state drives (SSDs), PCI Express (PCIe) add-in cards and other forms. Up-to-date NVMe devices are using PCIe with a 4 lanes interface which offers a transfer rate of approx. 7.8 GiB/s.

ODFS On-demand file system (ODFS) are created dynamically on request. These only serve a specific purpose and in contrast to dedicated storage only exist during a short period.

OpenFOAM Open-source Field Operation And Manipulation (OpenFOAM) is a C++ toolbox for the development of numerical solvers, and pre-/post-processing utilities for the solution of continuum mechanics including computational fluid dynamics.

PFS A parallel file system is designed to store data across multiple servers and to offer high-performance access..

POSIX The Portable Operating System Interface (POSIX) is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. In this thesis we always refer to file system compatibility.

SATA Serial ATA (SATA / Serial AT Attachment)[3] is a computer interface that connects to mass storage devices such as hard disk drives, optical drives, and solid-state drives. Third-generation SATA (released in 2008) interfaces has a transfer rate of 6 Gibit/s.

SCC The Steinbuch Centre for Computing (SCC) is the central computing center of the Karlsruhe Institute of Technology.

SI The International System of Units (SI, abbreviated from the French Système international (d'unités)) is the modern form and widely used system of units for physical quantities..

SIMD Single instruction, multiple data (SIMD) is a classification of parallel computers by Flynn.

SMP Symmetrical multiprocessor system (SMP) is a multiprocessor architecture in which two or more identical processors share a common address space..

SSD A solid-state drive (SSD) is a solid-state storage device that uses integrated circuit assemblies to store data persistently, typically using flash memory, and functioning as secondary storage in the hierarchy of computer storage. It is known as a solid-state device or a solid-state disk, even though SSDs lack the physical spinning disks and movable read–write heads used in hard drives ("HDD") or floppy disks.

SWF The standard workload format (SWF) was defined in order to ease the use of HPC workload logs and models. Programs need only to parse a single format to analyze workloads or simulate system scheduling.

Units

Units based on International System of Units (French *Système international (d'unités)*) (SI) and International Electrotechnical Commission (IEC) [2]

1 B	Byte: 8 bit
1 KiB	Kibibyte (Kbyte) according to IEC: 2^{10} Byte
1 kB	Kilobyte according to SI: 10^{3} Byte
1 MiB	Mebibyte (Mbyte) according to IEC: 2^{20} Byte
1 MB	Megabyte according to SI: 10^{6} Byte
1 GiB	Gibibyte (Gbyte) according to IEC: 2^{30} Byte
1 GB	Gigabyte according to SI: 10^{9} Byte
1 TiB	Tebibyte (Tbyte) according to IEC: 2^{40} Byte
1 TB	Terabyte according to SI: 10^{12} Byte
1 PiB	Pebibyte (Pbyte) according to IEC: 2^{50} Byte
1 PB	Petabyte according to SI: 10^{15} Byte

Some conversion 1 PiB $\Longleftrightarrow$ 1 PB

$$1\,\text{PiB}/1\,\text{PB} = \frac{2^{50}}{10^{15}} \approx 1.13 \approx 114\,\text{TB}$$

$$1\,\text{PiB} \approx 1\,\text{PB} + 114\,\text{TB}$$

$$1\,\text{PB} \approx 0.888\,\text{PiB} \approx 909\,\text{TiB}\ \ 506\,\text{GiB}\ \ 588\,\text{MiB}\ \ 416\,\text{kB}$$

1 Introduction

“A supercomputer is a device for converting compute-bound problems into I/O-bound problems.”
— Ken Batcher [3]

High-performance computing (HPC) systems, or so-called supercomputers [1], and their applications are subject to constant change. The systems are becoming larger, more heterogeneous and thus more complex. In addition, new use cases and usage profiles are emerging. The changing requirements are evident in the use of the systems. Traditional HPC applications are primarily computationally intensive, parallel, and scalable. It is common practice to reduce data operations as much as possible [4, 5]. Recently, new user communities are emerging that generate, process, and analyze huge amounts of data [6–10]. This trend is called data-intensive computing and presents a challenge for classic HPC architectures [11]. HPC systems and existing parallel storage systems are generally not designed for these new types of workloads. In contrast to computational capacity, which can be used exclusively, data storage is a shared resource and is used in a shared manner by all users of an HPC system. The drawback is, that individual users with data-intensive workflows can slow down the overall system.

In classical HPC applications, file operations often take place in intervals, typically with snapshots of the simulation stored on a storage system [10]. During data-intensive simulations, a large number of file operations is continuously generated, which creates constant pressure on the storage system [10]. Due to the increasing demand of classical applications and the new data-intensive communities, storage systems are increasingly becoming a bottleneck. In addition, data-intensive communities have an increased capacity requirement. The situation described above presents several challenges. (i)

[1] In this thesis, the terms HPC system and supercomputer are used as synonyms.

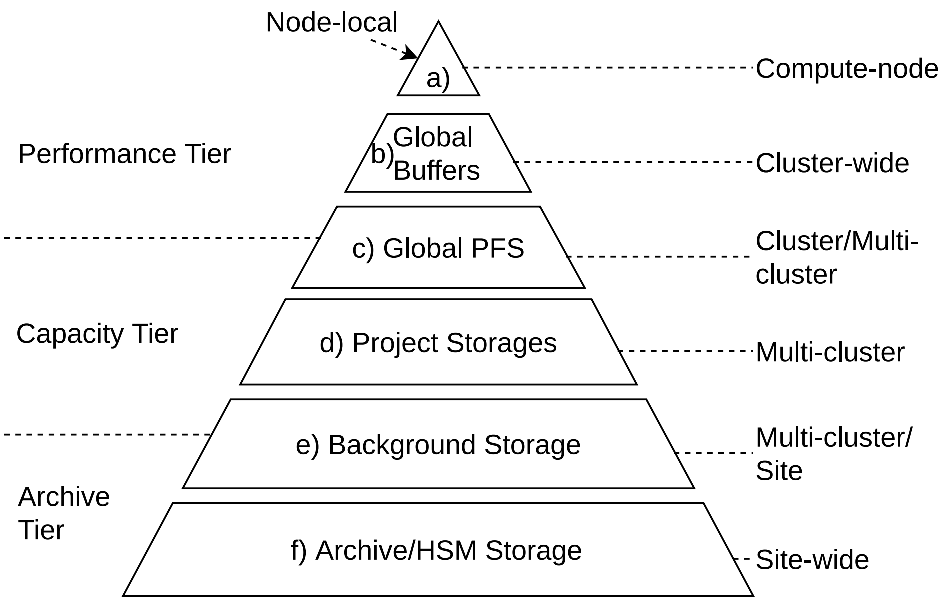


Figure 1.1: Storage Tier Pyramid

The storage system must provide a very high peak-performance, both in terms of throughput and metadata performance. (ii) The storage systems must be used more efficiently, to provide more free capacity on faster storage systems. The opening quote defines a supercomputer, as a device to turn compute-bound problems to I/O-bound. Therefore, the overall goal of this work is to turn I/O-bound problems back into compute-bound problems.

1.1 State of the Art

A common solution, to meet the new challenges, is to arrange storage systems in different tiers [12]. Tiered storage refers to a data storage environment, that consists of two or more types of storage. The differences between the tiers can be distinguished by four attributes: price, performance, capacity and function. These tiers can be presented as a storage pyramid. Figure 1.1 shows a possible storage pyramid within HPC centers. A very powerful storage system, a global parallel file system (PFS) is usually directly connected to the HPC system (c). Fast caches and buffers (b) can be used to reduce the load on the parallel file system. With the appearance of solid-state drives (SSDs) they have also found their way into computing nodes. For example, each compute node of the ForHLR II [13] has a SSD. These SSDs (a) can only be used locally in each node. Parallel use across nodes is not intended or possible (so far).

In addition, HPC centers often already have various types of storage that extend the hierarchy downwards (d–f see Figure 1.1). On the right side the visibility of the storage systems is marked. Cluster or Multi-cluster means here, that a storage system is directly connected to the clusters and can only be used by them. Often policy rules decide whether a storage is connected to multiple clusters. In contrast, there is storage which can be used independently of the clusters and also serve other purposes, so-called site-wide storage. These storages can be used with different protocols and tools. However, there is no general automatism that allows the data to be transferred to different storage levels and systems. Users must act manually here.

In this work, concepts have been developed under the conditions that typically can be found in HPC centers. To address the challenge described under (i), dynamically software generated file systems (on-demand file system) are used [14]. A temporary file system is created, with node-local SSDs, that can be used across nodes. This file system can be created for a single HPC job, a project or a campaign as needed. In this way, it is possible to create an on-demand file system tailored to the user's needs. This allows the HPC operator to provide an extremely fast file system at no extra cost. To meet the challenges under (ii), a data management concept was developed. For users of HPC systems, this data management concept enables transparent use of various storage systems. The HPC operator has the possibility to integrate new storage systems and technologies without the need to train the users.

To meet the above-mentioned challenges, different solutions have to be developed that extend the storage levels "upwards" (i) and more tightly link them "downwards" (ii):

Challenge 1

Conceptual design for deployment and integration of an on-demand file system (upper tier) and evaluation of its impact on applications.

The first step is to integrate an on-demand file system into HPC systems. Normal operation of HPC systems and the previous operating model should not be affected. Since there are different setups of such systems, a way to offer such a concept has to be designed for different environments. It must also be shown, if and how the performance of an application is affected. Until now, the requested resources have been available exclusively for a user job. With an additional on-demand file system, the computing resources now have also to serve an on-demand file system.

Challenge 2

Analysis of HPC job workloads to determine when data must be available on various storage systems.

Planning and coordination of data staging is one of the challenges. It is necessary to plan, when data is going to be copied to an on-demand file system. The timing of copying data plays a decisive role here. In the case of data staging, it is important to minimize its influence on running jobs. It is particularly important, that the resources must be known in advance, if data should be staged in advance.

Challenge 3

Development of a concept for data management across hierarchical levels (lower tier).

The final challenge is data management in HPC environments. Here, various storage systems must be made available to users by means of a simple abstraction layer. Furthermore, the solution must be able to easily integrate into the environment without complicating existing workflows.

1.2 Scientific Contributions

Overall, the contributions of this thesis focus on the conceptual design, its evaluation and demonstration in real-world scenarios. The provision of a very powerful on-demand file system and data management of large-volume data, are technology-open and therefore generally applicable. New technological developments can be incorporated with minimal effort. Furthermore, studies have been published which clearly show that premature data staging is hardly possible without affecting the operating model. On the other hand, it was demonstrated that the advantages of on-demand file systems outweigh the disadvantages. Each paper is discussed in more detail below.

Contribution 1

Analysis of job metadata for enhanced wall time prediction.

Various job metadata and their impact on wall time predictions were investigated. For job wall time prediction, machine learning methods and workload traces of large HPC systems have been used. Metadata which were previously ignored and not examined were considered. The evaluation shows that the accuracy of the wall times can be improved with automated methods. For both workloads examined, the deviation was drastically reduced. For 60 % of the users, the deviation was reduced from about 7.4 hours to about 1 hour and 1.4 hours. This contribution is associated with Challenge 2.

Contribution 2

Using On-demand File Systems in HPC Environments.

A simple solution for applications with very high I/O demands were presented in this publication. The proposed solution is to create a private parallel file system on-demand for an HPC job. This private file system uses the node-local storage devices, e.g., solid-state-disks (SSD). The developed solution is easy to add to HPC environments and requires only minimal configuration to the system setup. The conclusion of that work is, that the impact on running applications is manageable and the advantages for applications that generate a high load, outweigh the disadvantages. In some cases, applications may run slower, but the reduction of pressure on the global file system is prevailing in these cases. This contribution is associated with Challenge 1.

Contribution 3

On the Quality of Wall Time Estimates for Resource Allocation Prediction.

The node allocation behavior of a HPC batch scheduling system was evaluated in this publication. The goal is to determine, whether it is possible to stage data in advance, based on scheduler predictions. It is shown, that wall time estimates must be excellent to reliably predict node allocations. In reality, the required accuracy for data staging in advance is hard to achieve. Therefore, the behavior of a (standard) batch scheduler has to be modified in order to enable efficient data staging in advance. Following this work, different methods have been conceptualized to allow on-demand file systems and data staging. These solutions serve as proof of concept and demonstrate that the basic ideas work on production systems. This contribution is associated with Challenge 2.

Contribution 4

Ad-hoc file systems at extreme scales.

Methods for integrating the approach, of an on-demand file system (ODFS), into an HPC system were evaluated in this publication. It was demonstrated as a feasibility study on a real machine. The advantages and disadvantages of such an approach were shown. It was demonstrated, that an ODFS can be integrated into the normal operation, without affecting the operation model. In addition, the influence of on-demand file systems on the execution of HPC jobs and user applications were investigated. This contribution is associated with Challenge 1 and 2.

Contribution 5

JAWS: Just Another Workspace Suite

To meet the challenge for data management a framework has been developed – “JAWS”. As a centralized data management service, JAWS orchestrates the movement of data between the different storage tiers. It can be easily integrated into HPC environments to facilitate scientific work. JAWS simplifies the data management by abstracting the different storage types and technologies as well as the infrastructure. By using a batch system for data movement, the efficient use of the infrastructure is increased. The data and directories are merged into a logical unit, so-called workspaces. These workspaces will be provided with additional metadata in the central JAWS service and improve the data management for operator and user. JAWS is not a new type of storage or file system. Instead it helps users to orchestrate the amounts of data that are generated in scientific computing. It provides an abstraction layer of the various data storages available. In addition, it is possible to set additional metadata in the central service, which improves the retrieval and management of the data. A prototype implementation of JAWS has been developed which is freely available and licensed under the MIT license. This contribution is associated with Challenge 3.

1.3 List of Publications

Most of the contributions in this thesis have already been published in peer-reviewed conference proceedings. The publications relevant to the content presented in this thesis are listed in the following. This list only show publication with me as main author.

- **Mehmet Soysal**, Marco Berghoff, and Achim Streit. *Analysis of job metadata for enhanced wall time prediction.* In Workshop on Job Scheduling Strategies for Parallel Processing, pages 1–14. Springer, 2018.
- **Mehmet Soysal**, Marco Berghoff, Thorsten Zirwes, Marc-André Vef, Sebastian Oeste, Andre Brinkman, Wolfgang E. Nagel, and Achim Streit. U*sing On-demand File Systems in HPC Environments.* 2019 International Conference on High Performance Computing and Simulation (HPCS), 2019.
- **Mehmet Soysal**, Marco Berghoff, Dalibor Klusáček, and Achim Streit. *On the Quality of Wall Time Estimates for Resource Allocation Prediction.* In Proceedings of the 48th International Conference on Parallel Processing: Workshops, ICPP 2019, pages 23:1–23:8, New York, NY, USA, 2019. ACM.
- **Mehmet Soysal** and Achim Streit. *Ad-hoc file systems at extreme scales.* High Performance Computing in Science and Engineering'19 (HLRS), 01 2019. (accepted)
- **Mehmet Soysal** and Achim Streit. *JAWS: Just Another Workspace Suite.* International Conference on High Performance Computing and Simulation (HPCS), 2020 (submitted)

Software Availability. Corresponding software and developed tools are publicly available.

- JAWS: `https://github.com/mehsoy/jaws`
- Walltime prediction tools:
  `https://github.com/mehsoy/walltime-prediction-tools`
- On-demand file system tools: `https://github.com/mehsoy/ODFS-tools`

1.4 Thesis Outline

The remainder of this thesis is structured as follows:
Chapter 2 provides the necessary background for the thesis. The basic principles and the structure of the environment are explained. Basic introduction to supercomputers and the environment is given here. In addition, the basics for the data management are explained. Chapter 3 explains the used environment. Here the user applications are explained as well as the system. Chapter 4 continues with the discussion of the on-demand file system concept. At first a short introduction to related work is given. Afterwards the concepts for the deployment and first general applications are tested. Finally, scientific applications are used to demonstrate how a on-demand file system can be used and the advantages and disadvantages of using it are shown. Chapter 5 presents the investigation of the scheduling behavior. After a short overview in the related work, it is shown how wall time estimates can be improved. Afterwards, an evaluation of how good wall time estimates have to be in order to reach the goal of advanced data staging. Chapter 6 deals with the data management. Here the concepts are introduced and use cases were shown, why data management is needed and why it is important in this environment. In Chapter 7, a summary of this thesis is given and an overview of open questions and possibilities for future work is presented.

2

Basics of Supercomputing

“#3 pencils and quadrille pads.”

— Seymour Cray [15],
(when asked what CAD tools he used
to design the Cray I supercomputer.)

This work covers a wide scope in the area of HPC environments: the systems, operation model, production environment, user applications, and system software such as the batch job scheduler or parallel file systems. In addition to the setup of such an HPC systems, the operating and usage model is also important. Therefore, in this chapter an overview of such HPC systems and the environment is given, so that the required background knowledge for this thesis is available.

Before basics of supercomputing are explained, the terms supercomputing and supercomputer should be differentiated. Supercomputers are the systems themselves, i.e., the physical machine. The term supercomputing covers the whole field of supercomputing, i.e., the machines, applications, operation, etc.

2.1 HPC Applications

Supercomputers support research and technical development by enabling scientists to perform simulations, analyze data, or apply numerical methods to solve computational problems. To accomplish this, scientists execute their scientific applications on HPC systems and evaluate the results. Scientific simulations have always been a powerful tool and can provide insights, which laboratory experiments cannot offer. Often, these computer simulations are also much cheaper than real experiments. Computer simulations have changed the fundamental way of gaining insights. Whether some phenomena in the Universe are only suspected or exceed the life span of mankind, we are able to simulate them or prove them through computing capabilities. As such an example, the IllustrisTNG project [16] simulates the formation of cosmological galaxies. Each of these simulations covers a large time span of a universe, from shortly after the Big Bang to the present day. IllustrisTNG simulations make it possible to investigate a wide range of issues related to the temporal evolution of the Universe. Supercomputers are also used in the engineering field. Development of new airplanes would be impossible without supercomputers. Here, simulations of whole aircrafts [17] down to simulation of liquid fuel inside the combustion chamber [18] are performed to explore and improve every single aspect. We also see the results of supercomputers in everyday life, although not everyone is aware of this. An example are weather simulations. These require large computing power that only can be provided on HPC systems. In Germany, these are conducted by "Deutscher Wetterdienst" (DWD) using supercomputers [19, 20].

When using supercomputers, a distinction is made between different types of payload. These payloads are distinguished by resource requirements of the applications.

Parallel applications: Some highly-scalable and tightly-coupled scientific applications run many parallel processes with synchronized execution steps. Some specific hardware and software requirements must be met to achieve good performance under a synchronization model of this kind. As an example, it is important that the computing resources are homogeneous to allow a synchronous execution [21]. Furthermore, the system must have a very fast connection between the individual resources with a low latency. In addition, the resources of the system must be large enough to run the applications. Finally, a simple low-latency communication framework or messaging interface (e.g., Message Passing Interface (MPI) [22]) is required to ease the development of applications, that require communication and synchronization between processes.

High-throughput computing supports scientific applications that require to solve many small tasks for their research. Such applications do not need to be closely coupled and can be run concurrently. Usually the tasks are independent of each other. Such applications are typically very small in terms of resource requirements, but may require more resources in terms of execution time or I/O. An example of such high throughput payloads, are simulation jobs from the Large Hadron Collider (LHC) project [23]. Here, a lot of small jobs are created and distributed to the available resources. These LHC jobs usually only need a single node (or even core) for their computation.

Some scientific **workflows** are based on a different number of applications, which have to be executed in a specific order. After each individual section, an intermediate result is passed to a next stage of the workflow. It can happen that specific parts have different resource requirements. A very simple variant of such a workflow is the classic *pre-processing – simulation – post-processing*. In many simulations, the data is prepared before the actual simulation (pre-processing) and after the simulation the results may visualized (post-processing). To prepare the data in visual form, special hardware can be used, such as graphics accelerators. For more complex workflows, frameworks can simplify the workflow for the user [24, 25]. However, for workflows to be possible at all, it is important that a system scheduler supports the expression of dependencies between jobs.

Applications spending most of their time on I/O are called **data-intensive** [26]. Such applications are not compute-bound, but use most of their run time moving and manipulating data. Parallel processing can be achieved in data-intensive work, e.g., by segmenting the data sets, which are then distributed, processed and finally reassembled. The basic challenges for data-intensive computing are the management and processing of exponentially growing amounts of data as well as the number of metadata operations.

2.2 HPC Systems

The use of high-performance computers (HPC), i.e., computer systems with particularly high memory and performance, is commonly used in many scientific disciplines. Such systems, also known as supercomputers, can have different architectures and design concepts. The acquisition, construction and operation of such a HPC system can be very expensive [27]. To keep these costs as low as possible, it is therefore necessary to operate the computer in multi-user mode. To use the machine simultaneous with multiple users, system resources are handled by a management component, a so-called batch system. Section 2.2.4 explains in detail the components of a batch system.

2.2.1 Hardware Architectures

There are different conceptual architectures of supercomputers. In this section, a short overview of the different architectures is given. To get an overview of supercomputers, the TOP 500 [28] list of the fastest supercomputers is suitable. Twice a year, the 500 fastest systems in the world are published on the TOP 500 list. However, it should be kept in mind that the list is created on the basis of computing power.

Figure 2.1 shows the percentage share of the TOP 500 list from June 1993 to June 2020, based on the design concepts. Single instruction, multiple data (SIMD), and single-processor systems have not been among the TOP 500 since 1997. Large Symmetrical multiprocessor system (SMP) systems were represented until 2003, but have been replaced by clusters, Massively Parallel Processings (MPPs) and constellation systems. Towards the end of the last decade it is clearly visible that only MPP and cluster systems are in the TOP 500. However, a trend can also be seen that the presence of MPP systems in the TOP 500 continues to decline. Today, most of the supercomputer architectures are clusters. The different architecture types are defined below.

In the early days of supercomputing, **single processor** systems were a common option for a supercomputer. In some cases, special processors were developed which were only intended for use in supercomputers. An example for such a system was the

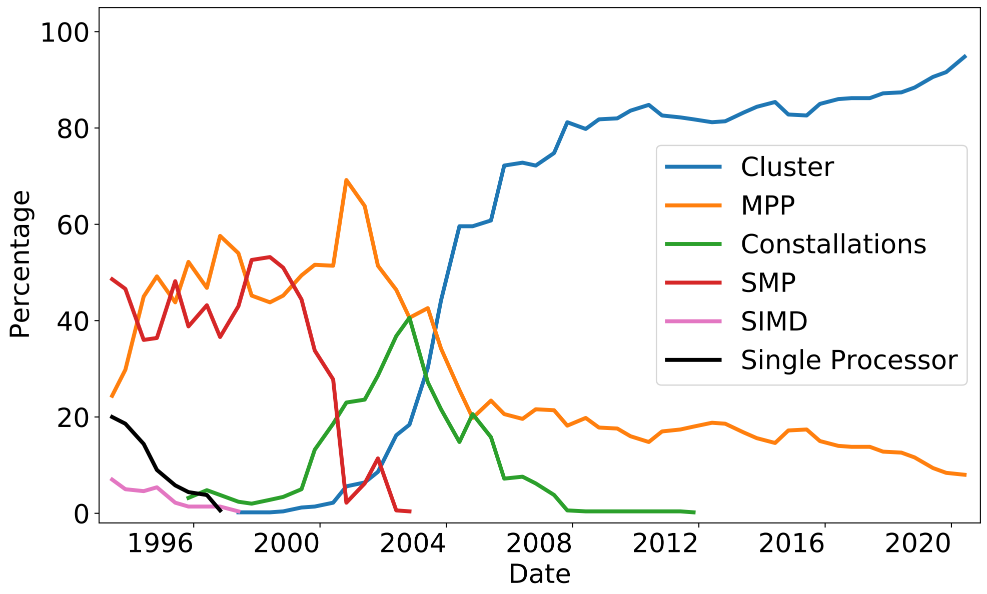


Figure 2.1: Design concepts change within the TOP 500 List. Data Source: `www.top500.org`

Hitachi S3800 [29] which was still represented in the TOP 500 in 1996. **SIMD** [30] is an architecture of mainframe computers or supercomputers, also known as array processors or vector processors. SIMDs are used for the fast execution of similar computing operations on several simultaneously arriving or available input data streams. It is important to note that such machines exploit data level parallelism. There are simultaneous computations, but only a single instruction at a given moment. An example of such a system is the Maspar supercomputer [31]. Today, SIMD supercomputers have disappeared from the TOP 500, but the idea of SIMD is available in today's popular processors as an extended instruction set, such as Advanced Vector Extensions (AVX) [32] in x86 based processors.

A symmetric multiprocessor (**SMP**) [33] system is a multiprocessor architecture in which two or more identical processors have a common address space. This means, that every processor shares the same memory or register with the same physical address. But as every CPU needs to be able to execute every process, the memory-bus becomes a bottleneck. With every additional CPU, the relative gain of performance decreases, because the bandwidth of the storage system is distributed to all processors. In order to avoid this, so-called NUMA (Non-Uniform Memory Architecture) was developed to reduce these problems. An example for such a system was the Sun Fire 6800 which contained up to 192 processors in one system and was listed in the TOP 500 until 2002 [34, 35]. Although these systems have disappeared from the TOP 500, the idea of SMP is more widespread than ever. All desktop computer, laptop, and even smartphones are basically a SMP system, because they have a CPU with multiple cores of the same type.

The terms constellation, cluster, and MPP are based on the classification used in the TOP 500 list. Depending on the point of view, such systems are similar and are classified in the same way by others [36]. The commonality lies in the idea of using identical commodity hardware which are installed in a significant number. The processing elements of such system are often refereed to as “nodes”.

A **cluster** [37] is made up of regular standard components. The individual systems are interconnected using standardized hardware and software technologies, such as Ethernet for networking or the Linux as operating system. In the beginning, so called Beowulf clusters [38] were common, which used a large number of interconnected simple desktop computers or workstations. Today standardized, rack mounted server systems are common, as they are built by many manufacturers. These server systems are not exclusive to HPC, but can be used for this purpose.

MPP [39] systems are similar in structure but usually use proprietary solutions. This can for example relate to the connection network or the operating system. For example: Tianhe-2 [40] is considered a cluster because it uses x86-64 nodes and a regular operating system (Kylin Linux). Sunway TaihuLight is considered as a MPP system, because the nodes have the special architecture SW26010 and operate on their own operating system called Sunway Raise OS [41, 42].

In contrast, **Constellation** [36] systems are classified as having a special design. Such systems usually have a blade center structure. There are several slots/nodes in one chassis. In contrast to cluster and MPP systems, the individual nodes are not independent of the overall system, i.e., they cannot be booted individually and the entire chassis has to be seen as single unit. An example of a successful constellation system is the HP Superdome [43], which made a significant contribution to the TOP 500 performance around 2003/2004.

By considering the classification of constellation systems, cluster, and MPP and compare it with Figure 2.1, a trend is clearly visible. HPC systems consist of individual components (nodes), which do not have the highest computing power. However, a large number of them are coupled with each other and thus offer enormous computing power. This coupling is decisive for the computing power of such systems and is explained in the next section. The three classes differ only in the selection of the individual components. Constellations have components that are proprietary and individual nodes can only be run in a blade center-like structure. The components (processors, network or operating system) of MPP systems may be individually operable, but they are usually proprietary and only make sense in a clustered network. Clusters on the other hand, are made of standardized commodity components that can even be mixed by different manufacturers.

2.2.2 Network Architectures

As mentioned in the previous section, the connection network of a supercomputer plays a crucial role. Here, only a brief introduction to such networks is given and the most interesting topologies are depicted, and explained in detail.

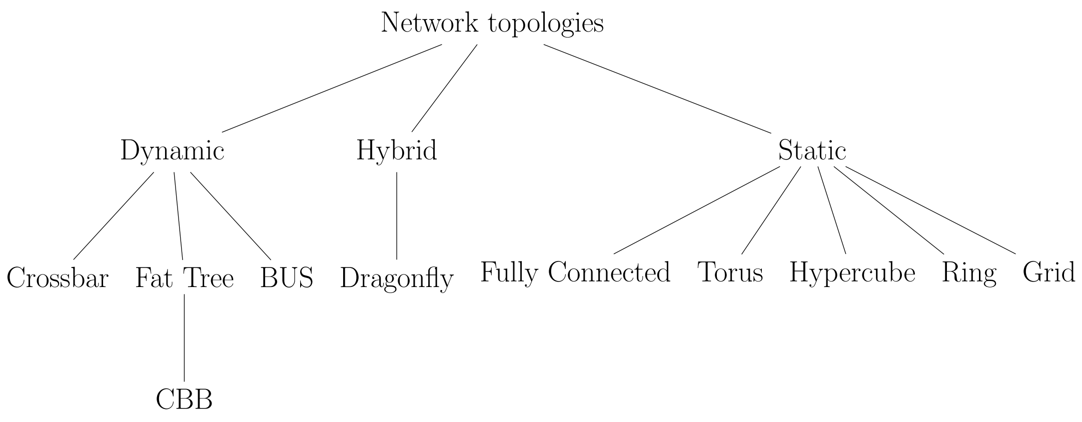


Figure 2.2: Hierarchy of interconnect topologies [44].

As shown in Figure 2.2, topologies can be divided into static and dynamic topologies and a mixture called hybrid [44]. Static network topologies provide fixed connections between the nodes, which means that the links between nodes cannot be changed or reconfigured easily. Dynamic topologies, in contrast, provide re-configurable connections between nodes using switch boxes. The connection path between two nodes is defined through the configuration of all included switches between the nodes. In recent times, hybrid solutions have also emerged. Hybrid networks [45] combine two or more topologies in such a way that the resulting network represents a new topology. Combining a tree topology with another tree is not resulting in a hybrid topology but instead still considered a tree. A hybrid topology is created whenever two different basic network topologies are connected, e.g., many individual trees are combined with a star-topology.

Metrics

To evaluate the usefulness of topologies, there is a huge set of metrics. The metrics describe the quality of a topology with respect to a specific property. In the following some metrics and their relevance are described. For better understanding, the metrics are distinguished into three main attributes: cost, performance, and reliability. These are explained here with corresponding metrics.

Cost The cost attribute contains metrics, that correspond to the resources used and thus reflect the cost of realization. Here the required connection points, switches, and links are determined. It is assumed, that each compute node is an endpoint that requires a number of inbound or outbound edge network cards. A link simply represents a cable, that connects two components of the topology. So with this, the main factors for the costs of a network topology are: number of network cards required per computation node, number of required links, number of switches (if needed). The *degree* of a topology indicates the number of links per node. The *number of links* (cables) needed to realize the topology.

	Degree	Links	Diameter	Bisection width	Connect-ivity
Fully connected	$n-1$	$n(n-1)/2$	1	$n^2/2$	$n-1$
Ring	2	n	$n/2$	2	2
d-dim Torus	$2d$	nd	$d(\sqrt[d]{n}-1)/2$	$2\sqrt[d]{n}^{n-1}$	$2d$
Fat-tree	$n+1/2$	$\frac{(\log_2(n+1)-1)(n+1)}{2}$	$2(\log_2(n+1)-1)$	$n+1/4$	1

Table 2.1: Overview of categorized metrics for various network topologies. n denotes the number of nodes.

Performance In addition to the costs, the performance of a network is another important criterion in the HPC environment. The following metrics provide a number of properties for estimating the bandwidth and latency of a network. The *diameter* of a topology describes the maximum direct distance between two nodes in hops. The *bisection width* specifies the minimum number of links, which must be cut, to split the network into two halves with equal size. Thus, it is a measure for the performance of a network. The bisection bandwidth results out of the bisection width, which represents a bottleneck in the worst-case communication scenario. In switch-based networks, the number of uplinks and downlinks determines the *blocking factor*. If the ratio is 1 to 1, then it is called fully non-blocking. If, for example, there are half as many uplinks as downlinks, then this is a 1:2 blocking factor. More uplinks than downlinks make no sense.

Reliability Another important feature of a network is its reliability. Reliability describes the security in case of failure of individual components. It is obvious that additional resources are needed for redundancy, to increase the reliability of a network. *Connectivity* specifies the minimum number of components or links that must be cut so that the network as such is no longer functional. This means that with higher connectivity, the network is more fail-safe.

These metrics are only a few selected ones, which are important for the discipline of HPC. More details and metrics about topolgies can be found in the literature [45–49].

Topologies

Here some basic topologies for static and hybrid networks are introduced. Table 2.1 shows the formula for the metrics [50] of the selected topologies.

A fully connected network [51] is a static topology where every node is connected to each other, and therefore sets the connectivity to $n-1$ with n for the number of nodes. Figure 2.3 shows a fully connected network for 8 nodes. Since every node is connected to each other no switching components are needed. The number of links

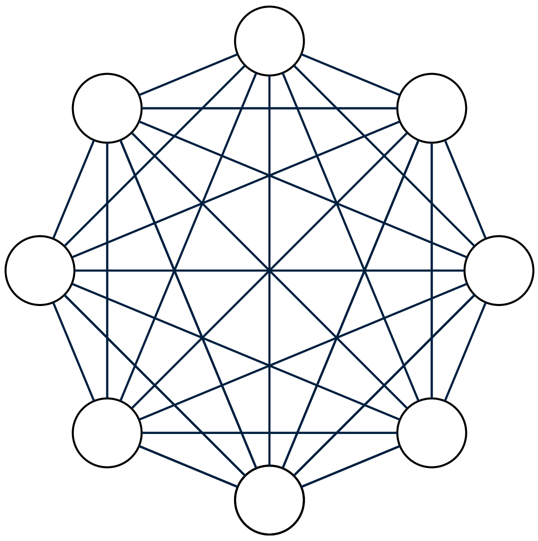


Figure 2.3: Fully connected topology with eight nodes.

is $n(n-1)/2$, for 8 nodes there are 23 links needed. If a connection is blocked and fails, the network can communicate around it. The data is redirected and the network is still operational. This ensures very high reliability in a fully connected network. However, a node cannot have unlimited networking cards. So this topology is only practical for a very small number of nodes.

A ring topology is a closed array of nodes. A ring is a static topology, where every node is connected to its neighbors via two-point connections, thus creating a closed ring. Figure 2.4 shows a ring topology for 8 nodes. A ring does not need any switching components. The data to be transmitted is forwarded from node to node until it reaches its destination. The connectivity is 2 and the number of links is n, 8 links for 8 nodes.

The Fat-Tree-Network is a universal dynamic network for very efficient communication [52]. Figure 2.5 shows fully non-blocking with 8 nodes and 3 switch layers. Every layer needs switching components with increasing number of ports per layer. This is impractical since very large switches are hard to realize or not even available. Also the reliably is reduced, e.g., if the root switch fails, a global communication is no longer possible.

A multi-dimensional torus interconnect is a switch-less static network topology. It is preferably used within very large supercomputers. As an example, the Blue Gene/L and Cray XT3 systems used a 3d-torus network and Bluegene/Q used a 5d-torus [53–55]. A proprietary variant of a torus network is the Tofu topology developed by Fujitsu. The latest variant is a 6d-torus network, called TofuD, used for the Fugaku supercomputer, which is the fastest system in June 2020 [56, 57]. Figure 2.6 shows a 3d-torus network with $4 \times 4 \times 4$ nodes.

For physical installations a Constant Bisectional Bandwidth (CBB) topology is common [52]. Also a lot of supercomputers are installed using a CBB Network, from

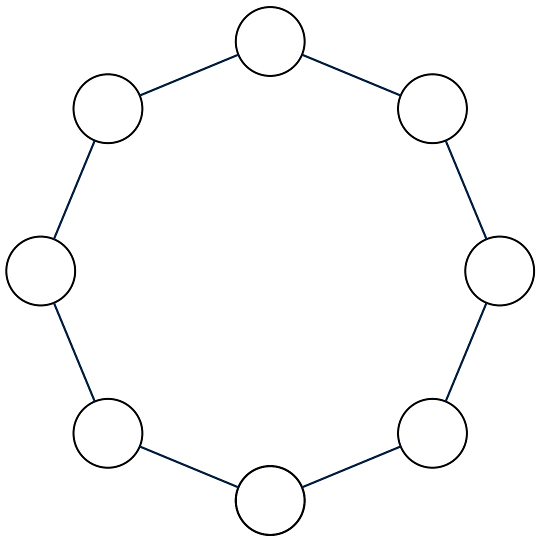


Figure 2.4: Ring topology with eight nodes.

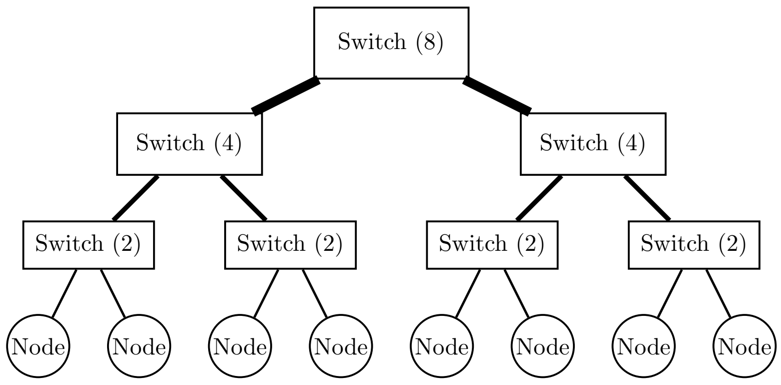


Figure 2.5: Fat-Tree network with 8 nodes.

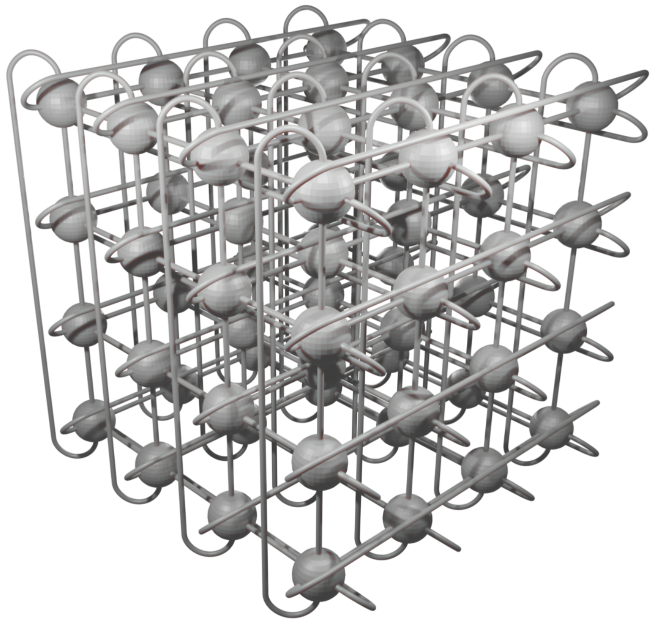


Figure 2.6: Three-dimensional torus interconnection of $4 \times 4 \times 4$ nodes. The connection in each dimension has a loop from the first to the last node. Artwork by Marco Berghoff [58].

entry level supercomputers like bwUniCluster [59], over mid-ranged systems like ForHLR I+II [60, 61], up to the fastest systems in the world like Summit und Sierra [62, 63]. Two versions of a CBB topology are introduced below, one fully non-blocking and one with a blocking factor of 1:8, both with 512 nodes.

An example of a fully non-blocking CBB topology with 512 nodes is shown in Figure 2.7. The interconnect shows a two-level fully non-blocking CBB network. Here, 36 port switches are selected as switching components. A communication, as worst case scenario, has to pass a maximum of 3 switches and 4 links. Regarding the hardware used for the scenario, 29 switches are needed on the first level, the so-called leaf switches. Each of these leaf switches has 18 compute nodes (one has only 8 compute nodes, $28 \times 18 + 1 \times 8 = 512$) and is equipped with 18 uplinks (links to upper level). So each node has logically its own uplink, hence the name fully non-blocking. On the second level are the so-called core switches (sometimes root switch), there are 18 switches. So 512 nodes in a fully non-blocking CCB network need, $29 + 18 = 47$ switches and 512 links to the nodes plus $29 \times 18 = 522$ links on the second level. So a total of 1034 links and 47 switches are needed for 512 nodes, when configured as fully non-blocking.

In contrast, Figure 2.8 shows a case where 512 nodes are also realized with a 1:8 blocking CBB network. Here, only 16 leaf switches are necessary. Each leaf switch has 32 nodes connected (downlinks) and has 4 uplinks, resulting in the mentioned 1:8 blocking factor. Since this case has only $64 (16 \times 4)$ uplinks to the second network layer, only 2 core switches are needed. So for 512 nodes in a 1:8 blocking CBB network $16 + 2 = 18$ switches and 512 links to the nodes plus $16 \times 4 = 64$ links on the second

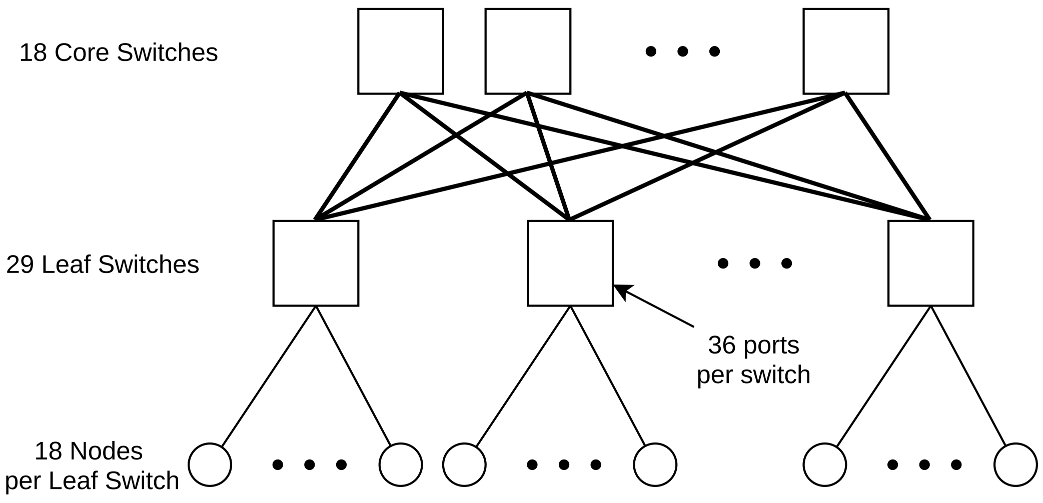


Figure 2.7: A CBB fully non-blocking topology with 512 nodes. Note, the last leaf switch has only 8 nodes.

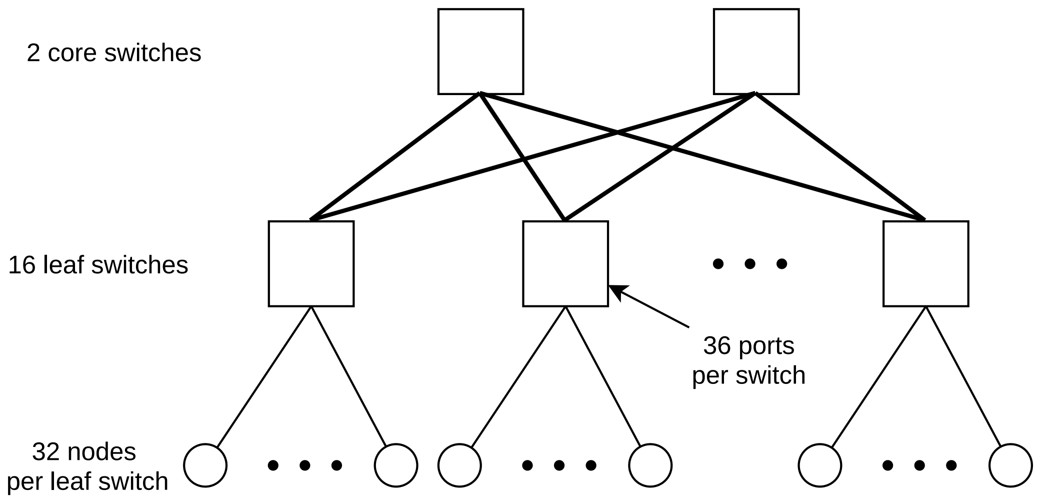


Figure 2.8: A CBB 1:8 blocking topology with 512 nodes.

level. A total of 566 and 18 switches are needed for 512 nodes, when configured in a 1:8 blocking CBB network. Of course, the 1:8 blocking topology has a drastically reduced overall network capacity, but also a drastically reduced number of network components. Almost half of the components are needed to built a 1:8 CCB network, than a fully non-blocking CBB network, with 512 nodes. Thus the procurement costs sink significantly, although the same number of computing nodes is available. Network topologies are a compromise between the cost, performance and reliability characteristics mentioned above. The example of two CBB networks presented here, show that, the cost of a fully non-blocking network increases significantly without adding more nodes. Therefore, it is crucial to choose the right topology for a supercomputer. In order to determine the best fitting topology, an enormous amount of domain knowledge and some preliminary investigations are necessary. For example, it is important to know used applications and how good they can scale or if they are topology aware.

2.2.3 File Systems

Since the early days of supercomputers, there has been a move to combine computers with the aim of pooling their computing and storage capacities. For distributed computing and communication between processes, there is, e.g., MPI which facilitates the work. However, it is difficult to run a program on several distributed computers with their own local storage, if the processes have to work on shared data. Here, clustered file systems are used, which are supposed to abstract from this problem of coordinating accesses and consistency problems. Within a cluster, such a file system ensures orderly access and provides the clients with a coherent view of the data distributed on several physical storage nodes in the form of a large contiguous file system. Such file systems can provide parallel access, as the data is distributed across multiple storage targets for redundancy and performance reasons. Often striping and chunking is used, a method that divides files into a large number of blocks (chunks), with each block being stored on a different storage target. This allows high transfer rates to be achieved even for individual files. Such file systems allow parallel access to individual data chunks. Therefore, such file systems are also called - parallel file systems (PFS).

Although the files and storage elements are distributed, parallel file systems abstract from this distribution and provide a Portable Operating System Interface (POSIX) [64] compliant access to the data. Some PFS, e.g., Lustre [65] and BeeGFS [66] use local file systems (ext4, XFS etc.) to manage their data with remote access and abstract from them. Parallel file systems differ from desktop file systems mainly through separation of metadata and actual data. The difference between metadata and data is explaned in Section 2.2.5.

Figure 2.9 shows a general PFS. A PFS usually has the following main components

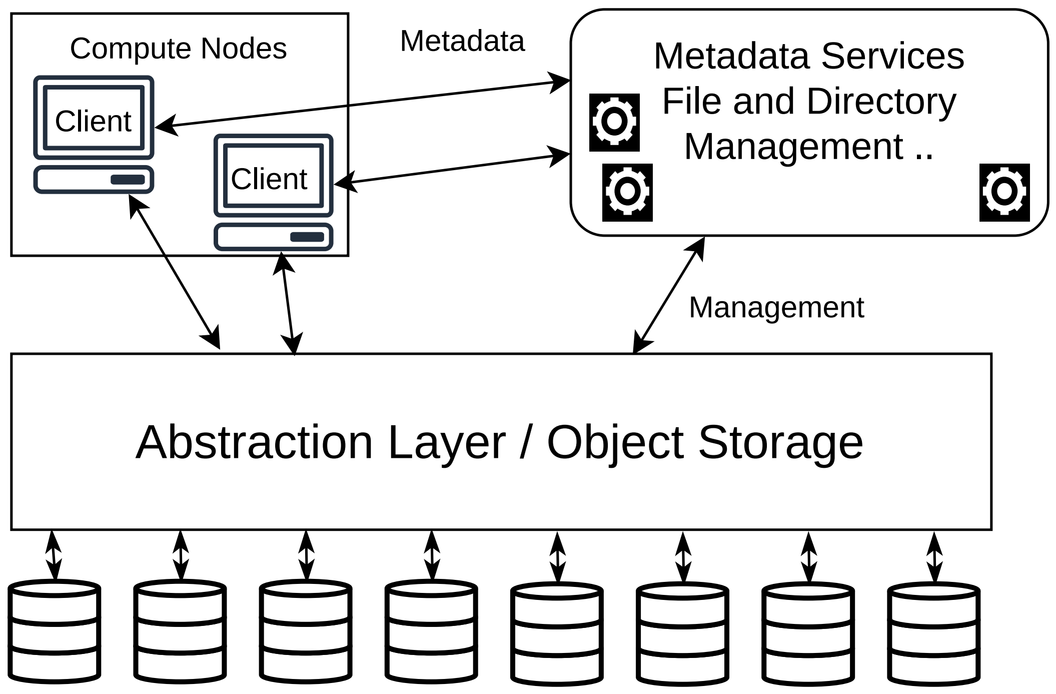


Figure 2.9: Generic illustration of a parallel file system. Compute nodes are connected to the object storage via a client. Object storage contains the data. Metadata are located within metadata services, here also files and directories are managed.

1. Management and metadata services include various services to manage the file system. There is one (or more) Metadata Server (MDS) which holds the metadata of the data to be stored. This usually includes a Metadata Target (MDT) which identifies the physical location of the metadata. In addition, various services can be available which ensure monitoring or fail-over.

2. Object Storage Server is the service, that stores the actual data. A distinction is made between an Object Storage Server (OSS) and an Object Storage Target (OST). An OSS refers to a server that provides the service and an OST refers to the actual device for storing the data. An OSS can therefore have several OSTs.

3. Compute Nodes use a file system client to access the global file system. These clients are often available as a kernel module and allows a system to mount the global file system.

The client only access the file system through a abstraction layer, which hides the complexity of a distributed file system.

2.2.4 Batch Systems

Usually, the execution of processes and management of resources of a HPC system are managed by a Batch System. The following is a brief overview of batch system concepts.

Most of us know how to run our program by typing the name of the executable file at the command prompt or simply clicking on an icon. In the same way, users would like to log-in to their HPC system, run their application at the command prompt. Given their popularity and investment, HPC clusters do not sit idle and wait for a user to execute any commands. Instead, they maintain a waiting list that includes so-called jobs. This waiting list and many other things are managed by Batch Systems.

A *Batch System* is responsible for receiving and parsing the user submitted jobs. If there are no resources available to run the job immediately, the job is added to a queue. A Batch System describes multiple components which work together. These usually consist of a Scheduler and a Resource Manager, as well as tools to use the Batch System, such as submitting jobs.

A *Job* is not simply the user's application, an executable file or a single command. The application needs input data, parameters, environment variables, and so on ... . All these specifications are collectively referred to as a job. The user creates a job script and submits it to the batch system. This job script is simply called "job" in this thesis.

The *Scheduler* identifies waiting jobs to execute, selects the resources for the job, and decides when to start a job. The scheduler commonly provides commands for displaying the limits, access permissions, and service agreements it enforces. In Chapter 2.3 more details about schedulers are given.

The *Resource Manager* is the part of a Batch System, which starts the execution of jobs. It is typically used to control resource usage policies, such as the number of cores or requested memory. A resource manager also maintains and monitors the resources managed and used by the batch system. The task of preparing nodes for the next job or cleaning the nodes after a job, e.g., from left over files, is also done by a Resource Manager. For this purpose, a mechanism is available, so-called job prologue and epilogue. This mechanism is executed, shortly before and after a job, on the allocated resources. Job prologue and epilogue are basically shell scripts used to clean, prepare or test the full functionality of the allocated nodes. An example, for such a tool to clean the nodes is the widly used node health check [67]

There is a variety of software that performs Batch System tasks, such as Portable Batch System [68], Moab Cluster Suite [69], Platform LSF [70] and the open source scheduler Slurm [71]. With these products, the boundary between scheduler and resource manager is blurry. Slurm includes a scheduler, called Slurm scheduler, and the resource manager which is called Slurm daemon. Most of the time both are simply called Slurm. In contrast, Moab is a pure scheduler which is not able to execute jobs by itself. Moab is usually used together with the resource manager TORQUE [72]. In the following, the terms scheduler, resource manager and batch system are used as synonyms for each other. If it is necessary to separate the terms, it is explicitly pointed out.

2.2.5 Metadata

In this thesis, the term metadata is used very often and here is a short overview how it is are used in the context of this work.

First of all, data and metadata have to be distinguished. Metadata are structured data and information about data, such as books, videos or pictures. In the case of an image, for example, the content shown would be the actual data, and the image size or whether it is a black-and-white photograph would be metadata information. In the field of computer science, metadata provides information on resources and data, so that they can be processed automatically. In the context of this work there are different types of metadata. Job metadata is such structured information about jobs and is explained in Section 2.3.2. Furthermore, the Chapter 6 explains metadata about the so-called workspaces in more detail.

In addition, there are the metadata related to PFS. Here, metadata describes, e.g., the file owner, or the creation date of a file and is stored by a separate service. So metadata and data do not have to be in the same place. And this separate service is called metadata server in the context of PFSs. When talking about metadata operations, these are requests or commands sent to the metadata service. This can be, for example, that a file should be created, deleted, moved or renamed, or even just a request to receive the ownership of a file. Within PFS the high bandwidth is achieved by many storage servers and scales with the number of server. In contrast, the metadata service is a central service of a PFS and therefore quickly becomes a bottleneck. When metadata operations are mentioned, it is not about the amount of data written to a PFS, but the number of requests sent to metadata service.

2.3 Scheduling

The creation of a schedule is an important part of all computer systems, where several actors compete for the same resources.

For the different disciplines where schedules are needed, highly optimized schedulers are developed. Accordingly, schedulers can be distinguished both by the way they work and by their specific field of application.

Following are examples of important domains with highly optimized schedulers:

- *The Process Scheduler* is a component of operating systems. It is responsible for the fair management of multiple processes running on a computer [73]. In process scheduling, there is a distinction between short, medium and long term scheduling.
- The *Hard Disk Scheduler* is used to manage reading and writing data on hard disks. An example for the Serial AT Attachment (SATA) protocol is the Native Command Queuing (NCQ) [74].
- *Job-scheduling* is about the correct control of jobs (batch jobs, program starts etc.) in mostly larger IT environments, which are in temporal and other dependencies among each other [75].

For HPC systems, and in particular for this thesis, only the discipline of job scheduling is considered.

2.3.1 HPC Job Classification

HPC batch jobs can be divided into two basic categories according to Feitelson and Rudolph [76]. These two main categories are static and dynamic allocations.

Static allocations include “rigid” and “moldable” job types. Rigid jobs are the most common type of jobs in today’s cluster environments. A rigid job is passed to a batch system with a request in the form of a fixed number of nodes and cores. The number of nodes cannot be changed after the request is passed and during the entire execution. A moldable job is similar to a rigid job except that the batch system may change its resource requirements between the time the job is submitted and started. Therefore, moldable jobs are submitted with a range of requested resources and the desired job run time. As an example, a job asks for a range of 2–4 cores for a run time of 2 to 1 hours. So based on the allocated cores, the run time is adjusted by the Batch System. Both types of jobs have in common that they do not change their allocated resources after job start.

The second form of job is dynamic. Again, there are two types of dynamic jobs – evolving and malleable. An evolving job can change its resource allocation (or reservation) during job execution. Here, the job initiates a change of the allocated resources by dynamically requesting or releasing resources. An example could be that the application, during its simulation, comes to a specific point, where additional resources are needed. Malleable jobs are similar to moldable jobs. The crucial difference is that the batch system can expand or reduce the job’s resources here at any time. The application adapts itself to the changed resources.

The difference between static and dynamic jobs is that once the jobs have been started, the allocated resources cannot be changed. In the case of dynamic jobs, the two types (evolving and malleable jobs) decide whether the job itself changes the resource allocation or whether it can be changed by the batch system. During this work, only rigid jobs have been considered. The evaluated machines and workloads contain only such rigid jobs and dynamics are not available in the operating model.

2.3.2 Job Metadata

HPC jobs have so-called job metadata, which are used to manage and organize jobs. Such metadata can be assigned by the batch system, such as the time of submission, or set by the user, such as a name for the job. Job metadata is also used to indicate resource requirements, such as the number of nodes or wall time. Some job metadata are only known after a job has finished, e.g., the actual value for “Used Memory”. The various metadata are explained in more detail in Chapter 5. In the following only a few metadata are explained, so that the terms are understood in the context of this work.

- *request wall time* or wall time estimate. Requested time in seconds, minutes, or hours to be allocated to a job. Usually given by a user.
- *submit time* represents the time of job submission.
- *start time* the time when the job starts to execute.
- *end time* the time when the job has finished.
- *run time* end – start time, actual time a job was executed.
- The requested *nodes* and *taskcount*—the number of requested cores—is a user requested value. Depending on the system configuration, these values (tasks/nodes) can be converted into each other. Also, depending on the operating model, only whole nodes can be requested and thus the corresponding cores (tasks) are assigned. In the further course of this work, requested tasks and cores are used as synonyms for each other.

Which metadata is available for our investigation can be read in Section 5.3.

2.3.3 Scheduling Algorithms

There is a wide range of scheduling algorithms for HPC batch job schedulers. To explain them all in detail exceeds the scope of this work. Therefore, only a small overview of the scheduling algorithms used in this work is given. A comprehensive overview of scheduling algorithms is given by Lueng [77]. Two common algorithms are described here, a simple FCFS and a back-filling algorithm (see Figure 2.10). These were used in this work for evaluating scheduler behavior. Also, a few important scheduling techniques are explained.

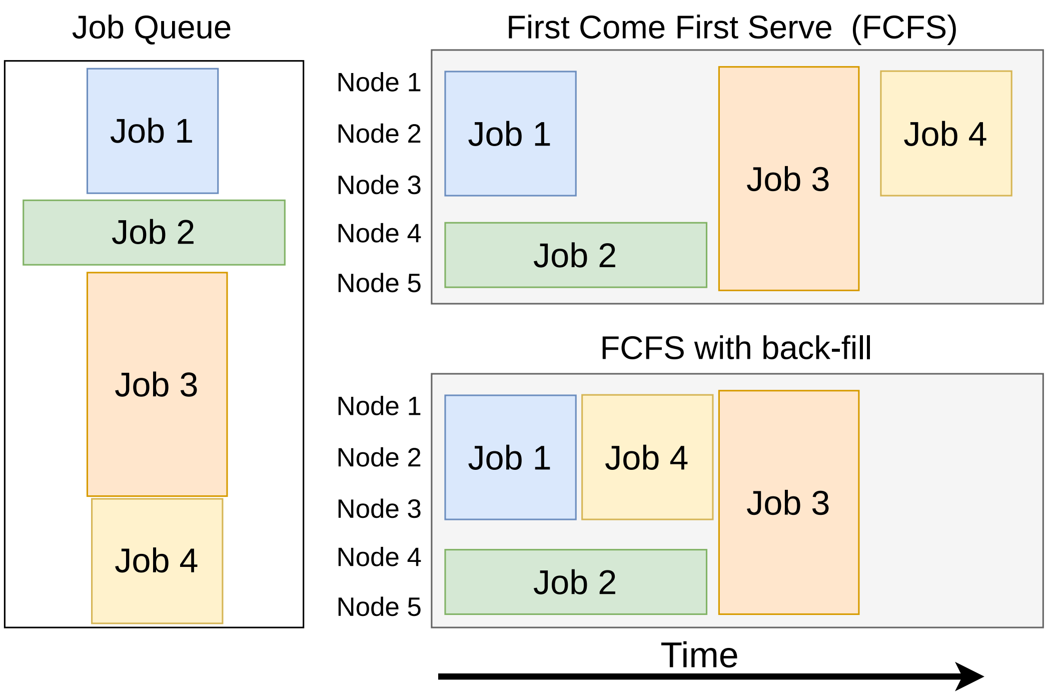


Figure 2.10: Example of Scheduling: FCFS and FCFS + Back-filling. Four jobs of different size are submitted to the job queue (left). On the right are the resulting scheduling for the First Come First Serve (FCFS) algorithms (top) and FCFS with back-filling (bottom).

First Come First Serve (FCFS) is a very simple algorithm. Jobs are queued and processed one after the other. FCFS is therefore fair and predictable[1], but causes drastic fragmentation of the system and therefore a low utilization.

Back-fill Algorithms: Simple algorithms like FCFS can lead to a bad utilization of the system, e.g., if a very large job is waiting for resources. Therefore, back-fill methods are often used to increase the utilization of the system. If a large job is waiting for resources, smaller jobs can be pushed forward. Two common back-filling strategies are "conservative backfilling" [78] and "EASY back-filling" [79]: With conservative back-filling (cons+BF), each job receives a reservation when it arrives in the queue, and jobs are back-filled as long as they do not cause a queued job to be delayed beyond its reserved start time. EASY back-filling is a more aggressive strategy, only the job at the top of the queue receives a reservation. Jobs may jump to the front in the queue as long as they do not delay the job with a reservation [80]. As an extension to the EASY-BF, modern schedulers can, for example, set the number of top prioritized jobs to be reserved. Thus, several high prioritized jobs can have a fixed reservation [81]

Schedule Compression refers to the method used to shorten the duration of a schedule. If a job is completed earlier than expected, the schedule is updated using a schedule compression algorithm [78]. During the update, the jobs are checked one by one and the

[1]Execution order of the jobs is predictable.

start time of each job is adjusted, i.e., it is moved to the earliest possible time window with respect to previously adjusted jobs.

Priority Based Algorithms: There are a variety of factors that can be configured that can affect the priority of a job [82, 83]. Priority can allow for more fairness between users, for example by taking into account the historical resource usage of each user. In this way, users with too much usage can be reduced in priority.

2.3.4 Accounting, Fairshare and more

HPC schedulers contain a number of mechanisms and functions that are necessary for the operation of a HPC system, but which go beyond this work and are therefore not considered further. Among other things, they contain functions and algorithms that determine which resources should be used for specific jobs. The algorithms can take into account, for example, the network topology or special job requirements. For example, in supercomputers with a blocking CBB interconnect topology [84], a large-scale application will run faster if all allocated nodes are not distributed across the whole network.

HPC systems also require some form of accounting or billing to track the usage of consumed resources. Based on this usage, schedulers may adjust the priorities for users and their jobs. This ensures a fair share of computing time. Of course, workload managers and schedulers include functions for basic operations to operate HPC systems, such as managing or checking computing resources (node-health-checks). All these features of HPC batch systems and schedulers are not considered further.

2.4 Wall Time Prediction

Efficient schedulers are characterized by a scheduling, that makes the most efficient use of resources possible. Today's schedulers meet this requirement by rearranging jobs in queues depending on free resources. In practice, however, the efficiency of such an approach is severely impaired by inaccurate user wall time estimates. Often, users could do a reasonable job runtime estimation, because of detailed knowledge about their jobs. Nevertheless, users tend to overestimate time, to prevent jobs being terminated too early by the scheduler. This detailed knowledge is not available without interviewing the user. Without this knowledge, it is difficult for the scheduler to perform exact resource planning. These inaccurate wall times also hinder timely data staging to the computing resources. Therefore, much of this work focuses on improving wall time estimates. A detailed insight into related work regarding wall time predictions is given in Section 5.1. Below is a small general overview of the field of machine learning, as it was used in this thesis.

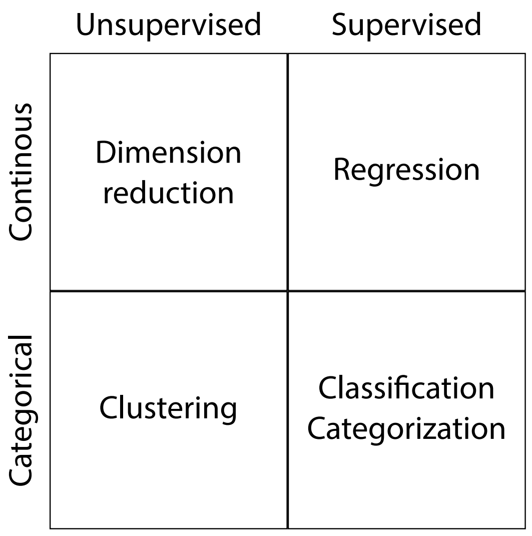


Figure 2.11: Distinction of Machine Learning (ML) concepts.

2.4.1 Machine Learning

Machine learning (ML) is about knowledge retrieval from data. It can also be understood as statistical learning and predictive analytics. In general, machine learning is a method to learn from a set of n samples with a target value and use the learned data to predict the target value from unknown samples. Machine learning has become a favorite tool for solving problems, and is also used for predicting wall times estimates [85, 86]. Machine learning methods can be divided to a few categories for solving different kinds of problems: Supervised and unsupervised learning [87, 88]. Figure 2.11 shows an overview. In unsupervised learning, only the input data is known and no known output data is given to the algorithm. This method is used to cluster similarities within the given data. Another use case is to visualize high dimensional data in two or three dimensions. Unsupervised learning is not covered any further, since it is not used in this thesis.

In **supervised** learning, an algorithm is used to train a model with input data and its associated output. This training process is also called model fitting. A trained model predict the desired output value from new input samples. There are two major types of supervised machine learning methods, called classification and regression. Classification handles problems where a given input is predicted to a predefined category, typical examples are sample datasets for iris flowers or recognition of hand written digits [89, 90]. Predefined wall time categories like short, medium and long are too inaccurate for a good prediction of a wall time.

To predict the wall time in seconds, a *regression* is suitable. There are several regression algorithms to choose from, e.g., Linear Regression, ARDRegression, Lars, Lasso, and Ridge [91–93].

The linear models offered by scikit [94], are based on generalized supervised linear regression. For these methods, a training set of target value and input variables x_i is used to determine the regression coefficients w_i of the linear combination. The predicted value $\hat{y}$ is a linear combination of the input variables x_i,

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \cdots + w_p x_p. \tag{2.1}$$

Prediction takes $O(1)$ time for trained linear models.

A second kind of regression algorithm is based on decision tables/trees (DT). This is a class of divide-and-conquer strategies that utilizes a tree structure to separate data according to their characteristics. The nodes of the tree store rules that determine how their children have been split and the leaves store either the actual outcome, a function that determines the outcome, or the actual data from where the prediction can be made [95].

Automatic Machine Learning The success of machine learning relies on expert knowledge to pre-process the input data, select a suitable algorithm for the ML model, and optimization of hyper-parameters. These tasks are very complex, time consuming, and often beyond the skills of non ML experts. Therefore, there is a high demand for automating the machine learning process, so the use of *Automatic Machine learning* (AUTOML) has gained high acceptance in a variety of disciplines [96–98]. In this thesis, two different AUTOML libraries were used – auto-sklearn [99] and TPOT [100]. Both are using scikit-learn [101, 102] as ML library. In a classical ML process, different models and systems have to be explored until the best is found. AUTOML tools automate the complex work of machine learning optimization and estimates the best performing model out of a range of various classifiers or pre-processors. The training of the model can be time and resource consuming, until an accurate model is found. After training, predictions take only $O(1)$ for a trained model.

2.5 Storage and File Systems

As already mentioned, the various storage systems are categorized into tiers [12]. In addition, the various storage subsystems can be divided into further categories, depending on their configuration and use in HPC centers. Figure 2.12 shows how such storages can be categorized in HPC environments. First, it should be explained, that the capacity increases and cost per byte decreases towards the bottom of the pyramid, so the large capacity storage is at the bottom. In contrast, performance (both bandwidth and latency) increases at the top, but the price also increases at the top of the pyramid.

Storage systems are usually a compromise between performance, capacity and price. Therefore, storage layers can also be classified into such "compromises" [12]. In Figure 2.12, these are referred to as performance, capacity, and archive tier. In the **Performance Tier**, storage systems are present whose main focus is on performance at

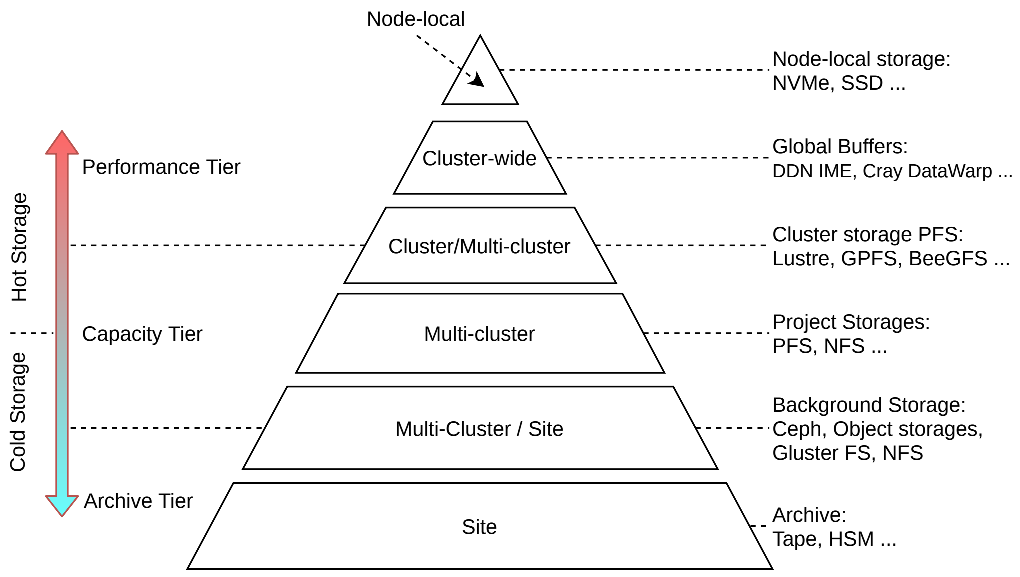

Node-local
Node-local storage: NVMe, SSD ...
Cluster-wide
Global Buffers: DDN IME, Cray DataWarp ...
Cluster/Multi-cluster
Cluster storage PFS: Lustre, GPFS, BeeGFS ...
Multi-cluster
Project Storages: PFS, NFS ...
Multi-Cluster / Site
Background Storage: Ceph, Object storages, Gluster FS, NFS
Site
Archive: Tape, HSM ...
Performance Tier
Capacity Tier
Archive Tier
Hot Storage
Cold Storage

Figure 2.12: Storage tiers available in HPC centers. Tiers can be divided into different categories according to requirements.

the expense of capacity. In the **Archive Tier**, the focus is on maximum capacity, where it is important to provide cost-efficient storage as long-term storage, while performance such as bandwidth and latency play a minor role. As a compromise of the upper and lower tiers, the **Capacity Tier** is in-between. Here, large capacities with acceptable speed to process data is provided. On the left side of the figure, there is a distinction between **Hot and Cold Storage**. This distinction refers to the operation of a HPC system. The distinction is not based on metrics like throughput and latency. A classification is made, whether jobs should start when required data is located on a specific storage system. Of course, a user only wants to start jobs when the data is on a very powerful storage system, but due to the workflow process it may be necessary to get small setup files from a slower storage. So a hot storage classifies a suitable storage for a HPC job. A cold storage, is either not used or not wanted due to policy rules, to be used within a job.

On the right side, different storage solutions are listed, which are used as typical representatives for the tier. Again, it can be seen that there is not always a clear assignment, as a storage solutions could be used in several tiers depending on the configuration. At the top of the pyramid the **node local** storages are shown. After the market breakthrough of NAND-based hard disks, SSDs or NVMe are very often used here. In recent years, a new tier has been established between the compute nodes and the Global PFSs, known as **Global Buffers** or Burst Buffers. Here, usually NAND-based storage systems are used. These are supposed to intercept short write bursts and thus relieve the global file system. How Burst Buffers are precisely used in HPC systems is explained in detail in the related work in Chapter 4.

The latest solutions include DataWarp from Cray [103] and Integrated Memory Engine (IME) from DDN [104]. The function and use of both buffers is fundamentally different.

DDN's IME is used as a transparent buffer between the clients and the global storage system, e.g., Lustre. For a user, the usage is transparent. The IME system shifts heavy I/O operations to the buffer and thus relieves the global storage system. However, for reasons of coherence, every access to the storage system must be routed through the IME buffer. In contrast, the CRAY Datawarp Burst Buffer are dedicated hardware resources in a cluster, which can be requested by the user. The data must then be transferred to these Datawarp nodes before the job is started. This can be specified to the batch system.

The next two tiers, **Cluster and Project Storages**, are very similar. The same storage systems can be used on both tiers, but the cluster storages are usually configured with more performance than capacity. Here the usual parallel file systems such as Lustre [65], BeeGFS [66] or IBM's Spectrum scale (formerly known as GPFS) [105] are widely used. These have now become established in this discipline, as they offer high throughput and are also very robust. Since parallel file systems are relatively expensive to operate and purchase, so-called **Background storages** are used in larger centers. These are slower in terms of performance but offer good latency, since they usually still use rotating disks. Often there are various storages which offer a wide range of access modes, such as Ceph, which can be used directly as object storage or as POSIX file system. Often these storage systems also serve other services and are operated with various appliances. At the very bottom, the long-term storage is located, refered as **Archive Storage**. These offer very cheap storage as long-term storage. Tapes are often used here, which are used in large tape libraries.

Of course, it is important to note that different storage types and file system should be used in a different manner for efficient usage. For example, it is best practice to move data in a archived format to a tape based storage, e.g., tarball or zip-archive.

2.5.1 Parallel File Systems

In this section, a closer look on parallel file systems is given. BeeGFS and Lustre are similar in their architecture (see Figure 2.13 and 2.14). Both file systems offer an MDS service, which can also be used in a distributed way for load balancing reasons. An MDS manages one or more Metadata Targets (MDT). Both offer Object storage servers and associated OSTs. Lustre has a long history and tradition in the field of computing and BeeGFS is a rather young file system, which was first developed by the "Fraunhofer-Institut für Techno- und Wirtschaftsmathematik" with the name FhGFS [106]. In 2014, the spin-off ThinkParQ was founded, which took over the distribution, customer service and professional support for the file system [107]. 2016 FhGFS now named BeeGFS was made open source [108].

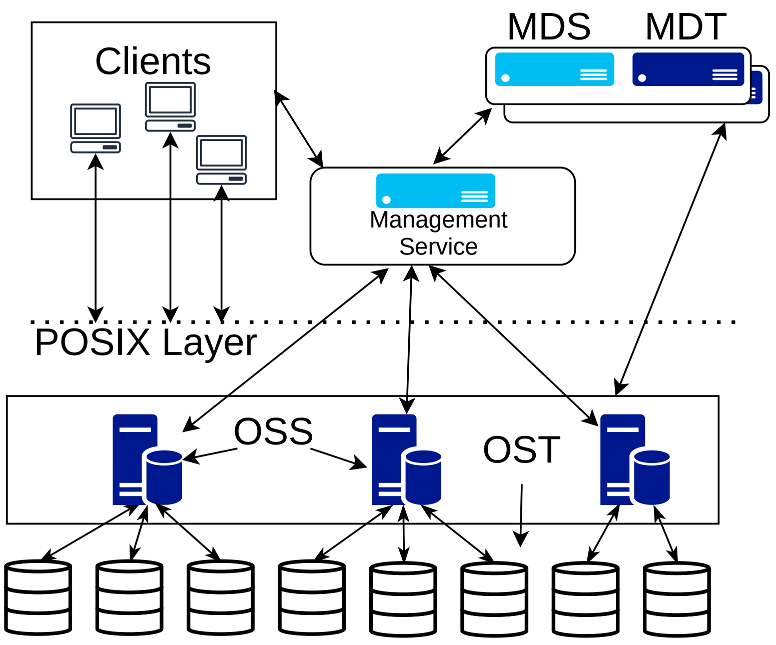


Figure 2.13: Simple overview of BeeGFS file system components. Storage services consist of Object Storage Server (OSS) and Object Storage Targets (OST). Clients access the storage services via a POSIX Layer. BeeGFS can have multiple Metadata Server (MDS) and Metadata Target (MDT). A central management service serves as registry for clients and servers.

2.5.2 Innovation and Current Situation

There are several reasons why the I/O bottleneck has become more severe in recent years. One reason is, that NAND-based storage systems have emerged onto the market. Table 2.2 shows the technical specifications of selected devices. As comparison for a spinning disk, an enterprise product with a SAS interface [109] was selected. The difference between the SATA-SSD and Non-volatile memory express (NVMe)-SSD are the connection and communication interface. SATA-SSD are connected via SATA (Serial AT Attachment), as were the rotating hard disks at that time. The computer communicates via an AHCI (Advanced Host Controller Interface) protocol with the memory controller. Nevertheless, for modern SSDs a new way was developed to reduce this overhead with a communication protocol. With the introduction of NVMe, the use of controllers was completely dispensed. NVMe-enabled SSDs are connected directly to the PCIe bus. Compared to SATA, this reduces latencies and raises bandwidth limits [113]. The gap between read/write throughput of NAND-based and rotating hard drives is significant. NAND-based drives are almost reaching the connection interface speed limit, with up to date PCIe 4.0 with 4 lanes able of providing around 7.877 GiB/s. Significant is also the IOPS increase from rotating disks to SSD. Even the fastest spinning disks available, are only able to deliver a few hundred IOPS. NVMe SDDs are able to deliver up to a million IOPS. One of the main reasons for this gap is, that with spinning disks a read/write head has to be moved physically to the exact position. A SSD does not

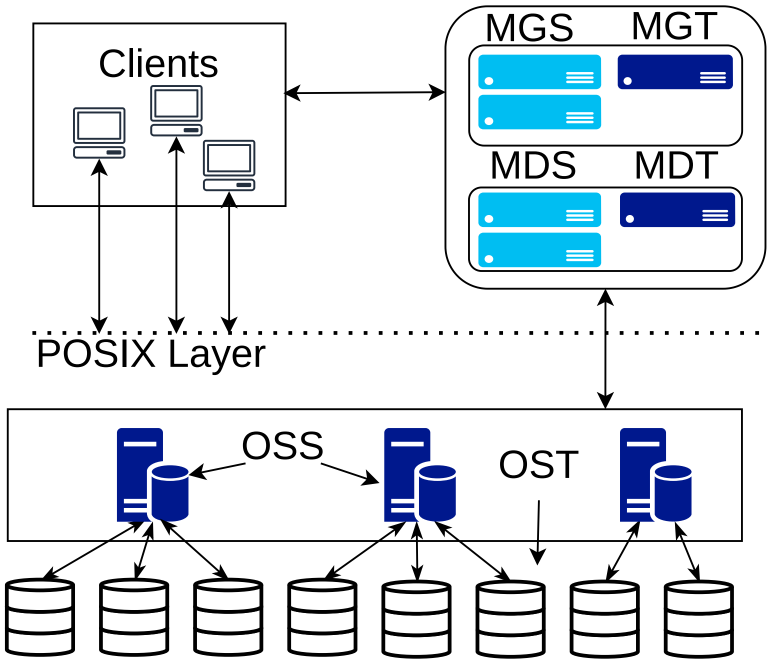


Figure 2.14: Simple overview of Lustre file system components. Storage services consist of Object Storage Server (OSS) and Object Storage Targets (OST). Clients access the storage services via a POSIX Layer. Lustre can have multiple Metadata Server (MDS) and Metadata Target (MDT). Multiple Lustre Management Services (MGS) serves as a management service. Servers and clients connect to a MGS on startup to retrieve the configuration for the Lustre file system.

Drive Type	HDD[109] SAS-3	SSD[110] SATA	NVMe SSD[111] PCIe 3.0 (x4)	NVMe SSD[112] PCIe 4.0 (x4)
IOPS	<100	up to 100 000	~360 000–400 000	~1 000 000
interface throughput (MiB/s)	1500	750	~3 900	~7 800
Read (MiB/s)	257	550	3 500	7 000
Write (MiB/s)	250	520	2 100	6 850

Table 2.2: IOPS and Read/Write throughput for selected devices.

System	Sequioa	Summit	ForHLR II	HoreKa
Year	2008	2018	2016	2021
Computing performance	∼16	∼150	∼1	∼17
Storage throughput (Gib)	1000	2500	50	∼110
Storage capacity (PiB)	∼55	∼250	∼4.8	∼13.8

Table 2.3: Depicted four HPC systems. Sequioa and Summit were the world wide fastest HPC system in the corresponding year. ForHLR II and HoreKa are the fastest systems at KIT.

have any mechanical components. As more and more SSDs have been used in desktops, the I/O bottleneck seems to have disappeared for users. But when users move from their desktop to a HPC system, and thus to the spinning disks, they fall into the I/O trap.

Another reason, for the increasing I/O bottleneck, is the growing gap between the computing and storage part. Table 2.3 shows four systems, two world's top systems and two mid-range systems. As examples, two of the fastest HPC systems "Sequoia" [114, 115] and 'Summit" [62, 116] are shown. Sequoia went into in 2012 and Summit in 2018, both systems were the fastest HPC systen at that time. As comparison, two of the mid-ranged system installed at SCC are shown, ForHLR II [13] which went into production in 2016 and HoreKa [117] which is going to be productive in Q4 2020/Q1 2021. Sequoia had a computing power of 16 PFLOPS (Peta Floating Point Operations Per Second) and a storage system data throughput of 1 TiB/s. Summit offers a computing power of about 150 PFLOPS with a data throughput of 2.5 TiB/s. Computing power has increased by a factor 10 and the throughput of the data storage by factor 2.5. From 2016 (ForHLR II - 1 PFLOPS) to 2020 (HoreKa - 17 PFLOPS), the computing power increased 17 times. In contrast, the performance of the fastest data storage increased from approx. 50 GiB/s to 110 GiB/s. Here, the computing power increased by factor 10 and the throughput of the data storage by factor of 2.2. In both classes, fastest and mid-ranged systems, the computation power increases multiple times faster than the storage throughput. Compared to the growth in available storage capacity, it is obvious that file system throughput is lagging behind.

2.6 Data Management

Data management and general research data management has always been a major task. In 2014, the German Rectors' Conference adopted a recommendation entitled "Management of research data – a central strategic challenge for university management" [118]. This emphasizes also the importance of research data management. It takes particular account for the increasing complexity of research data and the exponential growth in data volumes.

Data management in HPC environments must be viewed from several angles. There is the view of the user, the operator and the scientific view on data management. As an example for the the **scientific perpective**, The Deutsch Forschungsgemeinschaft (DFG) recommends in its document "Sicherung guter wissenschaftlicher Praxis" to keep data for 10 years [119].

> "Primary data as a basis for publications should be stored on durable and secure media for ten years in the institution where they were created."
>
> – DFG Recommendation

Other institutions such as the Helmholtz Association [120] or the KIT [121] are also following this recommendation. However, more points are covered in these documents from a scientific point of view. A scientific result is usually a complex product of many individual work steps and each step has to be cared of. Observation and experiments, including numerical calculations, whether as an independent working method or to support evaluation and analysis, initially produce data. Also legal topics are taken up here, e.g., how data has to be handled if several institutions are involved. Also, specifications are made, if scientists leave their institutions, how to handle data afterwards – "digital heritage".

The HPC Centers' **operator perspective** is bound to the specifications of the scientific facilities and institutions. Therefore, storage for long-term archiving must be offered. Tape solutions are still the best choice for such storage systems, as they usually have less power consumption and tapes last longer than disks [122]. With the amount of data to be stored, special attention must be paid to efficient use of the individual storage devices. Also the price for storing data, should be considered. In addition, to the price per capacity, network infrastructure also plays an important role. It is important to utilize the infrastructure as efficiently as possible. So the primary interest from the operator's point of view is to provide the user with storage at the lowest possible cost and only if necessary a cost-intensive storage. One problem that recurs is the digital heritage of a scientific user. Due to the environment and structure of the scientific staff, it often happens that after the completion of a research phase, the employees change their jobs and/or institutes. In this case, manual intervention is often required to handle the left-over data. It would certainly be helpful if it could be clarified how to handle the data.

From the **user perspective** it is primarily important to pursue scientific activity. This means a user has to prepare data (pre-processing), run simulations, and post-process results. How and where the data is located plays a minor role for the user. Only when progress is no longer possible, due to data management, users are getting interested in it. The user often does not care what kind of storage system is used, as long as a progress is made. If new types of storage are used, users often have to use new tools to copy and manage data. They often do not pay attention to efficient use of infrastructure and storage systems, causing problems such as putting many small files on tape libraries, which is a bad decision.

One reason for bad performance with small files. is tape drive must seek to the position of a file. This seeking reduces the the performance. Therefore, it is recommended to create a archive file containing small files [123, 124]. HPC users have become accustomed to the time it takes to copy their high-volume data. They often do this interactively and therefore often have to run an interactive shell. A queue-based universal data management tool would help the users to simplify the complex task of large volume data management.

2.7 Summary

In this chapter, basics about supercomputing and systems were introduced. It was explained, why supercomputing is needed and for what it is used. Afterwards, an introduction to HPC systems was given as well as the different concepts of such supercomputers were shown. Also the trend towards cluster-like supercomputers has been shown, with the importance of cluster-interconnect for such systems. Basics of network topologies have been introduced as well as an example for a typical network configuration. Afterwards, the components of a supercomputer have been explained, like parallel file systems (PFS), batch systems, scheduling, and the research topic of wall time predictions. At last, a small introduction in the topic of data management within such environments was given.

3 Evaluation Environment

“Evaluation by others is not a guide for me.”
— Bruce Lee [125]

So far, only general explanations about supercomputers have been given and this chapter will go into detail about such a system and its environment. In addition, a large number of system software and tools have been used, which will be explained in this chapter together with the environment. At the end, scientific applications are presented, which were examined in this thesis.

3.1 HPC System

For the evaluation, msathe ForHLR II [13] cluster at the Karlsruhe Institute of Technology was used. The second stage of the ForHLR was commissioned on March 4, 2016. The ForHLR II system is the successor of the ForHLR I [126] cluster, which was put into operation in mid-2014 and shut down in April 2020.

3.1.1 Hardware

ForHLR II consists of 1186 nodes and has multiple Global PFS (cmp. Figure 3.1).

The system consists of the following nodes:

- 5 login nodes, each with two 10-Core Intel Xeon E5-2660 v3 [127] processors with 256 GiB main memory and 480 GB local SSD.

- 1152 computing nodes, each with two 10-Core Intel Xeon E5-2660 v3 with 64 GiB main memory and 480 GB local SSD.
- 21 fat nodes for rendering, each with four 12-core Intel Xeon E7-4830 v3 [128], four NVIDIA GeForce GTX980 Ti [129] graphics cards, 1 TiB main memory and 4×960 GB local SSD
- 8 service nodes, each with two 10-Core Intel Xeon E5-2660 v3 with 64 GiB main memory

Figure 3.1 shows that compute nodes (purple) are divided into several partitions. There is a so-called “Fat” partition with nodes, which are equipped with graphics cards, for rendering tasks and with 1 TiB/s. Besides fat nodes, normal compute nodes with 64 GiB/s constitute the computing power to approx. 1 PFLOP/s. The normal computing nodes are divided into two islands, a small island with 336 and a large island with 816 nodes. Each island has a two-layer CBB network [130]. The nodes are connected to the network by an InfiniBand Host Channel Adapter (HCA). A detailed schematic diagram of the small island topology is depicted in Figure 3.2. 24 nodes are connected with 56 Gibit/s link to leaf switches, and 12×100 Gibit/s uplinks from each leaf switch are connected to the upper core switches. This configuration results in a very small blocking factor of $1 : 1.12$, which is negligible and the network can be considered fully non-blocking.

Besides the computing nodes, there are further service nodes (yellow), which do not contribute to the computing power of the cluster. These service nodes are intended for administrative purposes. On one hand there are so-called login nodes, which serve as an entry point for users. On the other hand, there are other service nodes running services that ensure the operation of the system, e.g., a batch system for job management or different monitoring tools for checking health of the whole system.

3.1.2 Storage and File Systems

Different storage systems are available for users to store their data. Figure 3.1 shows the available storage systems to users of the ForHLR II [13] cluster. Here the hot storage systems (red) are directly connected via the InfiniBand network. Other storages are accessible using a Ethernet network and these storages can be considered as cold storage (blue). The SCC operates multiple clusters that have a similar setup.

On HPC systems there is usually a so called home file system, which hosts the home directories of users. Here users can store their applications and configurations for their work. This storage system could be used as hot storage on ForHLR II, but it is preferred to use the much faster scratch/workspace file systems for this purpose [131]. The IMK file system is actually a project storage, installed and only available for users of the “Institute für Meterologie und Klimaforschung” (IMK). It is directly connected to the InfiniBand network and it is operated like a hot storage. The figure also shows retention policies for some of the global PFS. While the home file system does not have a data

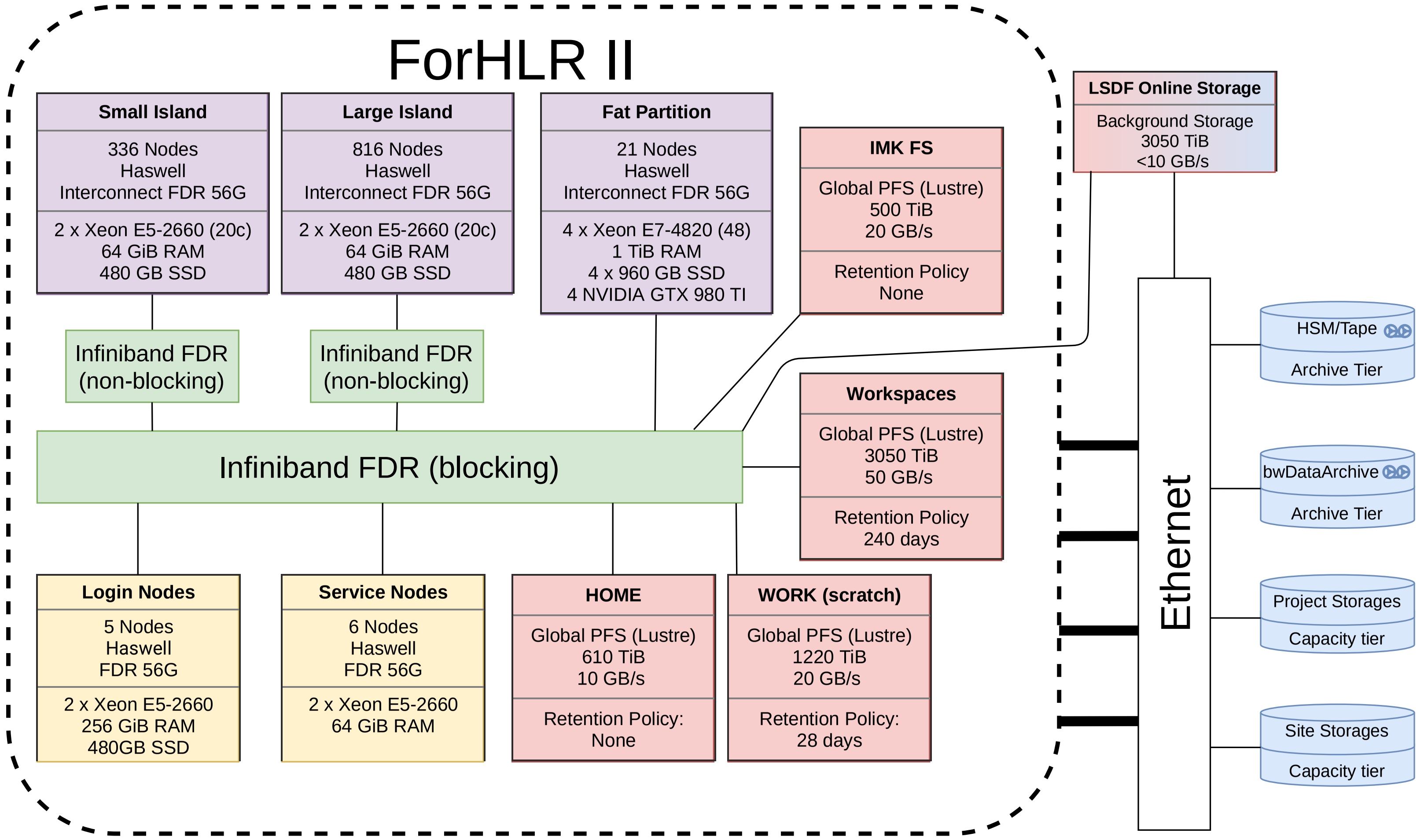


Figure 3.1: Schematic representation of the ForHLR II cluster with attached storage systems. Purple: Computing nodes with different partitions (fat, large and small island), yellow: Service Nodes for login and running the system, green: cluster interconnect, Red: hot storages directly connected to InfiniBand network, Blue: variuos storages accesible over ethernet.

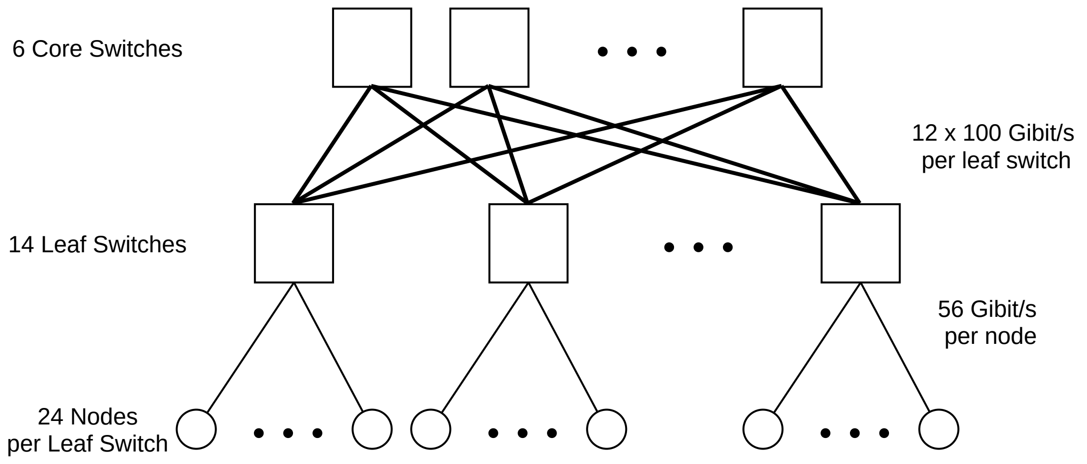


Figure 3.2: The small island consists of 6 core switches and 14 leaf switches. Each leaf switch has 24 nodes with 56 Gbit/s and 2x100 Gbit/s links to each core switch.

retention policy, work and workspaces file systems have a policy for data retention. On the work file system, a maintenance script is checking the modification date of files. If the data was not changed for more than 28 days, they are considered as unused and are removed [131]. This short period is not uncommon for such scratch storage [132–134]. Of course, a user is warned shortly before the data is removed. Deleting data in this way can lead to inconsistent data sets, e.g., if a single file is deleted from the results of a simulation, pretty much all data of the simulation may become useless. Also, this method requires recursively checking file timestamps through the entire file system. This method can generate correspondingly high loads on the file system. The workspace file system has a slightly different approach. Here, files and directories are aggregated to a so-called workspace. These workspaces are considered as a unit and all files in a workspace are treated equally (more about workspaces see Chapter 6) [1]. The LSDF Online Storage [135] has a special role. It is accessible as a side-wide storage over Ethernet, but has recently got an InfiniBand connection to serve as a hot storage within the ForHLR II cluster. The connection to the side-wide storage systems are provided over Ethernet. Here we distinguish between the archive-tier and the capacity-tier. The archive-tier provides a HSM system and a dedicated tape library (bwDataArchive [136]). The capacity-tier includes different project and site-wide storages, some of them may be operated by third parties and can be accessed with tools and protocols from the user space.

3.1.3 Metadata Performance of the Parallel File Systems

In this thesis, often metadata performance of parallel file system were addressed, but concrete numbers were missing. After installation, benchmarks are usually started

[1]Newer installation at SCC do not use a classic scratch file system. Instead space for large data sets is only available with workspaces.

to measure the performance of file system. To test the performance of the metadata service MDtest [137] was used. This benchmark performs various operations on the file system, such as creating and deleting directories and files, and determines an average value after several runs. Table 3.1 lists selected values from this benchmark (detailed list in Appendix A.4)[2]. Creation and reading files also involves a connection to OSTs. A "stat" operation is to retrive metadata, such as the owner of a file and only needs a connection to the metadata service[3].

File system	Directory creation	Directory stat	File creation	File stat	File read
HOME	13 588	364 989	101 641	183 297	160 128
WORK	12 591	352 069	73 432	248 275	163 157
Workspaces	31 715	332 744	82 120	243 150	137 580

Table 3.1: Chosen metadata benchmark values during the acceptance phase with mdtest (`mdtest -d <dir> -i -n 3600 -i -p 10`). More comprehensive list available in Appendix A.4

3.2 System Software and Tools

Different system software and tools have been used and enhanced. In this section a brief report about these is given.

Scheduling Simulator In order to evaluate the planning behavior and to investigate whether advanced data staging is possible, software is required that can imitate the scheduler of an HPC batch system. Such a software is called scheduling simulator. The simulator reads as input a recorded historical job load and then simulates the execution order of the job. Based on the parameter set and algorithms used, the behavior of batch schedulers can be evaluated using various options. The Alea Scheduling Simulator [138] was used, in this thesis, to investigate scheduling behavior. In addition to historical workloads provided by SCC, additional workloads from the Parallel Workload Archive [139] were used. More about this is covered in Chapter 5.

On-demand File System BeeGFS was chosen as the parallel file system for the dynamically generated on-demand file system (ODFS). There are several reasons why BeeGFS was chosen. One is that BeeGFS already comes with tools to create BeeGFS

[2]Please note that both of these benchmarks were done during installation and may have been changed by improvements and updates to the Lustre software.

[3]It should be mentioned, when trying to get the size of a file, a connection is also made to OSTs. Only OSTs know the file size

on-demand and another reason is that it is very lightweight and the server components can be started as user space daemons. Furthermore, it provides a POSIX abstraction layer, which is considered to be the state of the art in HPC environments. Another important reason is, that it can use a usual Linux file system as data storage. This allows to use the node-local available existing file system as a data storage for BeeGFS storage daemons [66, 140, 141]. Nevertheless, one of the work packages of the ADA-FS Project was the development of an on-demand file system, which was especially designed to be used as an ODFS in HPC environments. Such a file system was developed by Vef et al. [142] and shows a significant metadata performance increase compared to conventional file systems. Also the interference is reduced compared to other file systems. However, in this thesis generally available file systems have been used to demonstrate the approach. More about deployment of ODFS is covered in Chapter 4.

Loopback Devices As explained in the previous paragraph, BeeGFS uses a normal local file system to store the data. This simplifies configuration, but can create another problem if the data stored by BeeGFS, needs to be cleaned up. If a HPC job is creating several million files, these files are also stored on the local file system and deleting them may take some time. During the deletion of this temporary data the node is not available for other jobs. To solve this problem, a so-called loop-back device is used. UNIX-like system can use a loop device as a virtual block device. This virtual block device can use a file as underlying storage instead of a physical device [143]. Instead of removing millions of files, only the large file which serves as loop back has to be deleted. Additionally a small performance boost due to the internal caching mechanisms of Linux is received. Similar advantages have been shown by Yamamoto et al. [144]. The global parallel file systems on the compute nodes are mounted again as loop back. In addition, a loop-back device also helps to manage space consumption on each SSD. The loop-back file has a certain size and more cannot be stored in this loop-back file.

Listing 3.1 shows an example on how to create a loop-back device and use as a normal file system.

- 1) create a file filled with zeros and a specified size using the dd command.
- 2) generate a file system structure on the created file.
- 3) mount this fle with loop-back device on /mnt/data.
- 4-6) show the capacity of the mountpoint

Listing 3.1: Example how to create a loop-back drive as virtual disk device

```
~$ dd if=/dev/zero of=test.img bs=1MB count=1000
~$ mkfs.ext4 test.img
~$ mount -o loop test.img /mnt/data
~$ df -m /mnt/data
Filesystem      1M-blocks  Used Available Use% Mounted on
/dev/loop19           923     3       857   1% /mnt/data
```

Tools During this thesis mostly standard tools were used, which are usually available in HPC environments. Many small tools, e.g., to evaluate file system statistics, have been developed ad hoc and adapted to the environment. However, there are different ways to start and stop the ODFS within the batch system and published on GitHub: `https://github.com/mehsoy/ODFS-tools`. To measure the throughput of a file system IoZONE [145] was used and to measure the metadata operation performance MDtest [137] was sued. MDtest is an MPI-based benchmark designed to test parallel file systems and is also used as a sub component for the I/O 500 [146][4].

3.3 Applications

Various applications and use cases were provided by users. These use cases have been selected for their I/O behavior. This behavior caused a significant load on the global PFS and slowed down the whole system. This affects all other jobs and users that are reading or writing data at the same time. As an example for such a bad behaving application, the load of the PFS was recorded using a OpenFOAM use cases. Figure 3.3 shows the average load caused by the use case. The blue line shows the load on the metadata server and the orange line is the average of all object storage servers. The application was only run for about 25 minutes, and the load generated can be seen very clearly. In comparison, the minutes before (0–15) and after (40–60) show the normal load, which the parallel file system is facing before and after the test run.

Such application are identified during the usual operation by the admins. If monitoring systems detect a unusual high load, admins investigates what is causing this. After identifying bad behaving application, corresponding users are getting notified by the administrators. Such a message is shown in the Figure 3.4. Sometimes users do not even know, that their applications have such a I/O behavior. Users that could not change this behavior, were of special interest for further investigation. In the following of this section the applications and use cases are briefly introduced.

The first application which has been investigated is **Open Source Field Operation and Manipulation (OpenFOAM)** [147]. OpenFOAM is a toolkit for computational fluid dynamics (CFD) written in C++. It is a widely used open-source fluid dynamics code [148] for engineering applications. OpenFOAM offers tools for many areas of research, like heat transfer, reacting flows, or multi-phase flows and utilizes MPI for

[4] I/O 500 is analogue for the fastest storage systems in contrast to TOP500 for computing.

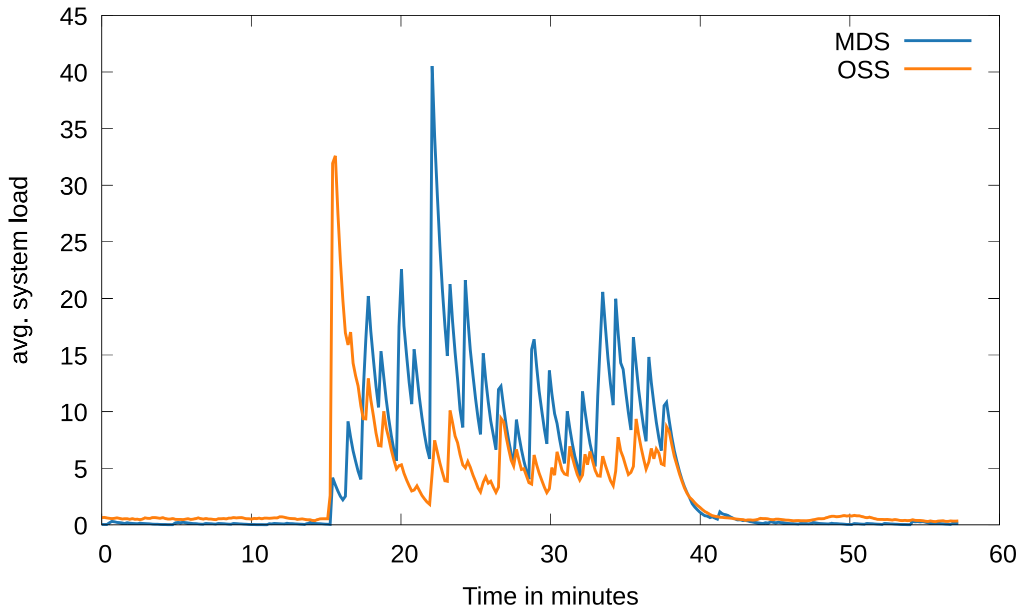


Figure 3.3: Average UNIX load caused on the Lustre file system during a simulation run of 25 min. Minutes 0–15 and 40–60 show the usual load on the Lustre file system.

```
Hello dear user,

we noticed that your job <jobid> on the ForHLR II
make several 100 million metadata operations:
Job <jobid> has done 330,512,988 getattr operations.
Job <jobid> has done 141,624,203 setattr operations.
Job <jobid> has done 327,735,960 close operations.
Job <jobid> has done 327,781,720 open operations.
Job <jobid> has done 330,460,676 mkdir operations.
Job <jobid> has done write operations with
                sum of 1,122,594,488,725 bytes.
[...]
```

Figure 3.4: Example mail send to a user due to massive metadata operation.

parallel communication. For this, the computational domain is decomposed into subdomains which are distributed among the processes. Investigation of I/O performance has been performed with two OpenFOAM cases, both using OpenFOAM v1712 and a custom solver developed for detailed combustion simulations [149, 150].

The second application is the **Neoteric Autonomous Stencil code for Jolly Algorithms (NAStJA)** framework. It is a massively parallel stencil code solver developed at SCC. It is designed for simulations in several scientific domains. Modules for the calculation of the phase-field and the phase-field crystals as used in mesoscopic or atomistic computational material science are available, as well as biological cells can be calculated with the cellular Potts model, a cellular automaton. The framework provides several modules for writing out the results [151].

As third application **super_sph ("Simulation for Smoothed Particle Hydrodynamics")** from *Institut für Strömungsmaschinen* (KIT) was evaluated. The software scales up to 15000 cores and 10^9 particles. The first implementation of the software was writing a file per process and required data-gathering as post-processing. A new implementation is now writing directly to time steps using MPI-IO which makes the data-gathering process unnecessary [18].

The last application is a **swarm-based meta-heuristic** for optimization of composite objective functions comprising multiple responses [152]. During the optimization process, these responses are weighted fully automatically with respect to each other according to the current knowledge state, yielding a dynamically evolving objective-function landscape. The algorithm is implemented in python. It was applied to parameter optimization for data-assisted bio-molecular simulations, a powerful tool to interpret experimental data on proteins in terms of molecular structures. This use case is still under development and was provided by the corresponding community at KIT.

3.4 Summary

This chapter explains the environment, applications and tools we used to do our research. The ForHLR II was described in detail and the storage pyramid was mapped to the resources available at ForHLR II. Also the cluster interconnect was described in detail. The storages where introduced with corresponding retention policys and if they were used as hot or cold storage. The metadata performance of the parallel file systems was shown as comparison for the ODFS in the next chapter. Furthermore, the tools and scientific applications were introduced.

4 On-demand File System

"A multithreaded file system is only a performance hack."
— Andrew S. Tanenbaum [153]

Today's HPC systems use parallel file systems, such as Lustre [65], IBM Spectrum Scale (GPFS) [105], or BeeGFS [66], that offer POSIX semantics, as the storage subsystem. Despite the separation of data and metadata to improve performance, metadata performance cannot scale so easily by adding more hardware to the storage subsystem. The metadata server is a centralized service and requires a lot of synchronization and is therefore much more difficult to manage. This metadata bottleneck still exists today. In addition, parallel file systems (and their I/O subsystems) are often shared by many users and their tasks, whereas computing resources are exclusively usable. Due to the shared use of the parallel file system, users can influence each other in a negative way.

Many of the recently installed large-scale HPC systems, e.g., Sierra [63] and Summit [62], are offering node local storage, ranging from SSDs to NVRAM, as high-bandwidth node-local storage. Also all clusters at Steinbuch Centre for Computing (SCC) are equipped with node-local SSDs, e.g., ForHLR II is offering local SATA-SSDs and the upcoming HoreKa [117] supercomputer will be equipped with faster NVMe-SSDs. As already described in Section 2.2.1, clusters or cluster-like systems are the preferred configuration for supercomputers today. It is becoming more and more popular to equip such systems with node-local storage, because the nodes are mostly standardized components and can be easily and cost-effectively equipped with NAND-based storage. SSDs are also known to have lower failure rates and are therefore well suited for use in compute nodes [154].

This part of the work deals with the goal of expanding the storage pyramid upwards. As already mentioned in the introduction, the approach is to achieve this by creating a fast parallel on-demand file system. Figure 4.1 illustrates the concept. Here are two jobs which occupy three nodes each. In the first case (Figure 4.1(a)), jobs write directly to the global parallel file system. In the second case (Figure 4.1(b)), a private parallel file system is created for each job. This private ODFS is only available on the allocated nodes of a job and uses the node-local storage (in this case the SSDs).

In order to achieve this, some questions and concerns must be answered and possibly disproved. On the one hand, it must be shown that this approach delivers performance at all. Additionally, the interference with the application must be determined, because the resources are no longer exclusively available for the application. The advantages and disadvantages in normal operation must be demonstrated or it should be shown that the advantages of an ODFS outweigh possible disadvantages.

This chapter first gives a short overview of the related work to solve the I/O bottleneck. Then it is introduced how a ODFS can be added as an additional feature to HPC systems. In doing so, the operational model should be affected as little as possible. Following this, results with general benchmarks are shown and how a ODFS behaves in real use cases with scientific applications. At the end, a brief summary and some remarks are given.

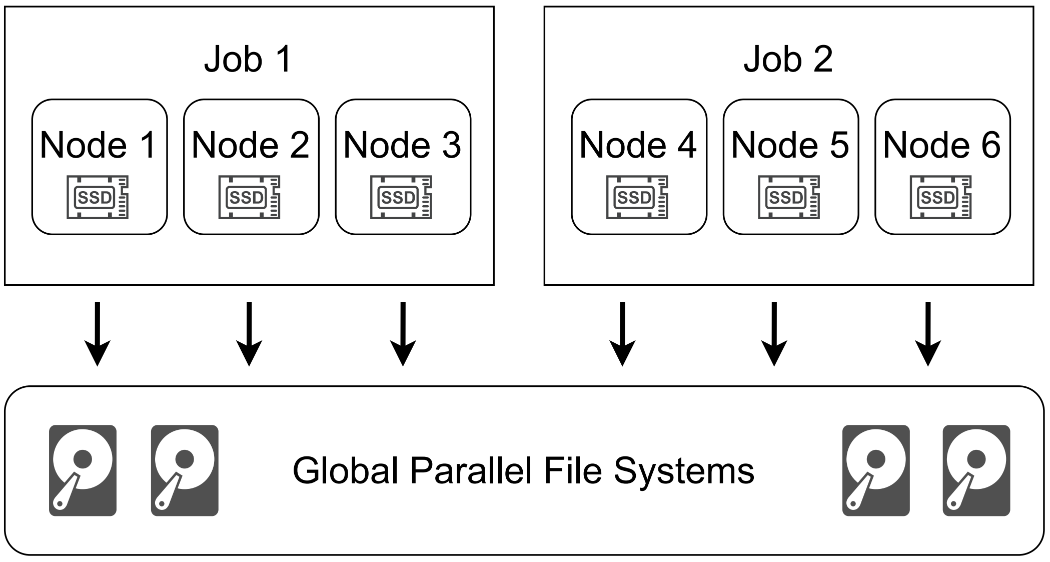
The approach for an on-demand file system and some of the results discussed in this Chapter have originally been published:

- M. Soysal, M. Berghoff, T. Zirwes, M.-A. Vef, S. Oeste, A. Brinkman, W. E. Nagel, and A. Streit. “Using On-demand File Systems in HPC Environments”. In: *2019 International Conference on High Performance Computing and Simulation (HPCS)* (2019) [155],
- M. Soysal and A. Streit. “Ad-hoc file systems at extreme scales”. In: *High Performance Computing in Science and Engineering’19 (HLRS)* (10/2019). Accepted [156].

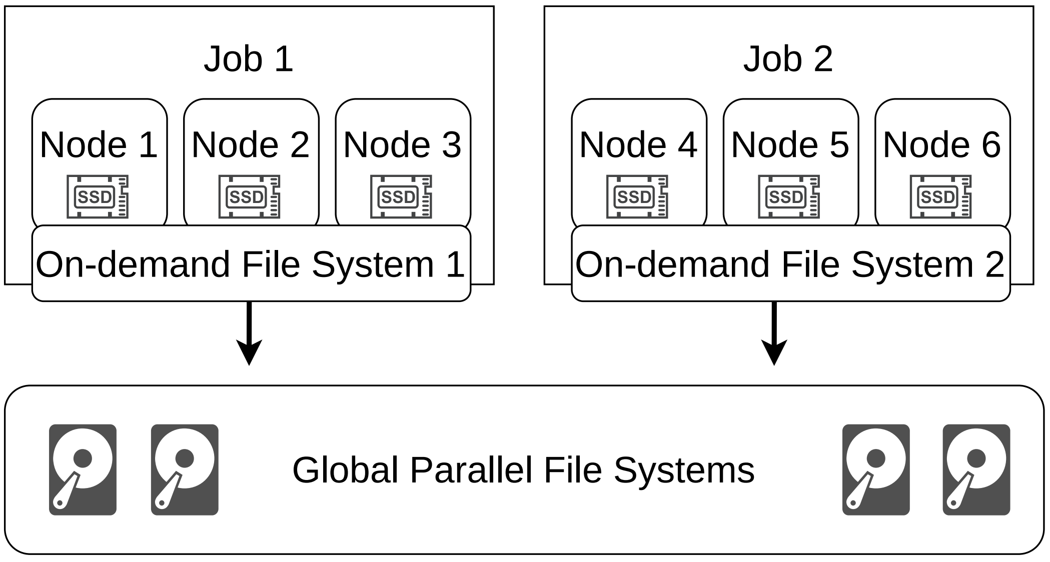
4.1 Related Work On-demand File System

In the past, there have been many developments and innovations to improve I/O throughput and metadata performance. In this section an overview of basic approaches is given with a few examples of each. Although it is difficult to strictly separate the solutions, four categories can be distinguished: hardware solutions, file system functions, and dynamic system reconfiguration and libraries.

Hardware solutions Today’s wide spread use of SSDs in compute nodes of HPC systems has provided a new way of accelerating storage. SSDs provide higher throughput and better random I/O performance than spinning hard disks. SSDs have been considered for parallel file system metadata, as the metadata performance of PFS is a major bottleneck in HPC environments storage [157, 158].



(a) Without on-demand file system. Both jobs writing directly to global file system.



(b) Both jobs have a private on-demand file system.

Figure 4.1: Storage usage (a) direct access to global PFS (b) using ODFS.

A different kind of hardware solutions are burst buffers, which aim to reduce the load on the global file system [159]. These type of burst buffers can be categorized into two groups [160]: remote-shared and node-local. While **remote-shared burst buffers** [104, 161] use central, dedicated I/O nodes to forward I/O operations.The function and use of both buffers is fundamental different. DDN's IME is used as a transparent buffer between the clients and the global storage system, e.g., Lustre. For a user, the usage is transparent. The IME system shifts heavy I/O operations to the buffer and thus relieves the global storage system. However, for reasons of coherence, every access to the storage system must be routed through the IME buffer. In contrast, the CRAY Datawarp Burst Buffer are dedicated hardware resources in a cluster, which can be requested by the user. The data must then be transferred to these Datawarp nodes before the job is started. This can be specified to the batch system. However, analysis has shown that burst buffers cannot prevent congestion under all circumstances. [162]. **Node-local burst buffers** are typically used within compute nodes and may be managed by the PFS, e.g., parallel log structured file system (PLFS) [163]). Examples for such node-local burst buffers are BurstFS [160] or BeeOND [140].

File system features File systems offer interesting new features to improve I/O bottlenecks. GPFS has introduced a highly available write cache (HAWC)[164]. HAWC uses node-local solid-state drives (SSDs) as buffers for the global file system. Lustre has received the Progressive File Layouts (PFL) [165] feature. PFL dynamically adjusts the chunk size and stripe pattern depending on I/O traffic. BeeGFS offers storage pools [166] and allows to group storage targets in different classes, e.g., one pool with small but fast solid state drives, and another large pool for spinning disks. However, such solutions are only available when using the vendor software solution and coupled with the global file system.

Libraries There is a large number of libraries available for improving I/O behavior of an application. High-level libraries, such as HDF5 [167], NETCDF [168] or ADIOS [169], are trying to help users to express I/O as data structures and not only as bytes and blocks. Middleware libaries, such as MPI-IO [170], help to improve usage of parallel storage systems, e.g., data sieving and collective I/O [171]. SionLib [172] is another library for task local file I/O, developed by Forschungszentrum Jülich. When files are accessed in parallel, only the open and close functions are collective while writing and reading files can be done asynchronously.

System reconfiguration Like an ODFS, there are serveral approaches which require a dynamic system reconfiguration. A dynamic remote scratch [173] implementation was developed to create an on-demand block device and use it with local SSDs as a LVM [174] device. A similar solution is recently built, where a central flash storage serves for data-intensive workloads. Logical volumes are created dynamically and can be mounted on compute nodes. This NVMe based storage is able to offer 2 TiB/s throughput [175]. These approaches mentioned above, rely on a dedicated storage system. Using an ODFS such a dedicated storage is not needed for temporary storage. Another software based solution is the RAM-disks Storage Accelerator [176]. It introduces an additional cache layer into HPC systems. It uses the available memory of commodity servers to build a PFS using RAM-disk. The PFS constructed on the RAM-disk is offered to the computing nodes as a very high-speed and low-latency temporary storage. While this is a similar approach, an ODFS uses the node-local disks of compute nodes and do not require other servers to build an on-demand file system.

4.2 Deployment of the On-demand File System

Usually HPC systems use a batch system, such as SLURM [71], MOAB [69], or LSF [70]. The batch system manages the resources of the cluster and starts user jobs on allocated nodes. At the start of the job, a prologue script may be started on one or all allocated nodes and, if necessary, an epilogue script at the end of a job (see Figure 4.2(a)). This, prologue and epilogue mechanism, makes it easy to develop a solution that can be used in almost every environment. Figure 4.2(b) shows the job workflow when using a simple solution. The resource manager executes the prologue on the allocated nodes. During this process, an ODFS is started on the allocated nodes. After the job has finished the ODFS is stopped during epilogue. Afterwards the allocated nodes are made available to other jobs.

The MOAB scheduler is offering a mechanism to inherit once allocated resources and nodes, to other queued jobs. This allows to run different jobs on exactly the same nodes. Additional to this, a staging mechanism [177] is available. However, this mechanism is limited to jobs with only one node as resource request. For this, a job is split into 3 parts, which run as separate jobs and are then linked together. The job scheduler then ensures, by resource inheritance, that all 3 separate job parts are started on the same resources. The staging mechanism has been modified accordingly. The new workflow, using the feature of resource inheritance is shown in Figure 4.2(c)). Here, at the beginning a resource allocation is created. During the first job part, a, ODFS is created and data is staged in. After the stage in job has finished, the allocation is inherited to the next job in the chain. At the end, data is staged out and the ODFS is stopped.

Another solution is to extend the function of the SLURM burst buffer plugin [178]. The plugin executes scripts for allocating dedicated burst buffers and data staging. In this case the burst buffers are dedicated, besides the computing resources, and can be requested like another resource. This mechanism is not applicable with an ODFS, because the nodes for a job are only available from the moment of allocation. To make this burst buffer plugin usable with the idea of an ODFS, the plugin was extended [179]. When this plugin is used for ODFS, the scheduler reserves the required number of

Nodes	8	16	32	64	128	256
Startup time in (s)	10.2	16.7	29.3	56.5	152.1	222.4
Shutdown time in (s)	11.9	12.1	9.4	15.9	36.1	81.1

Table 4.1: BeeGFS startup and shutdown speed with default tools.

nodes and initiates an ODFS (cmp. Figure 4.2(d)). This allows to manage on-demand file systems like burst buffers.

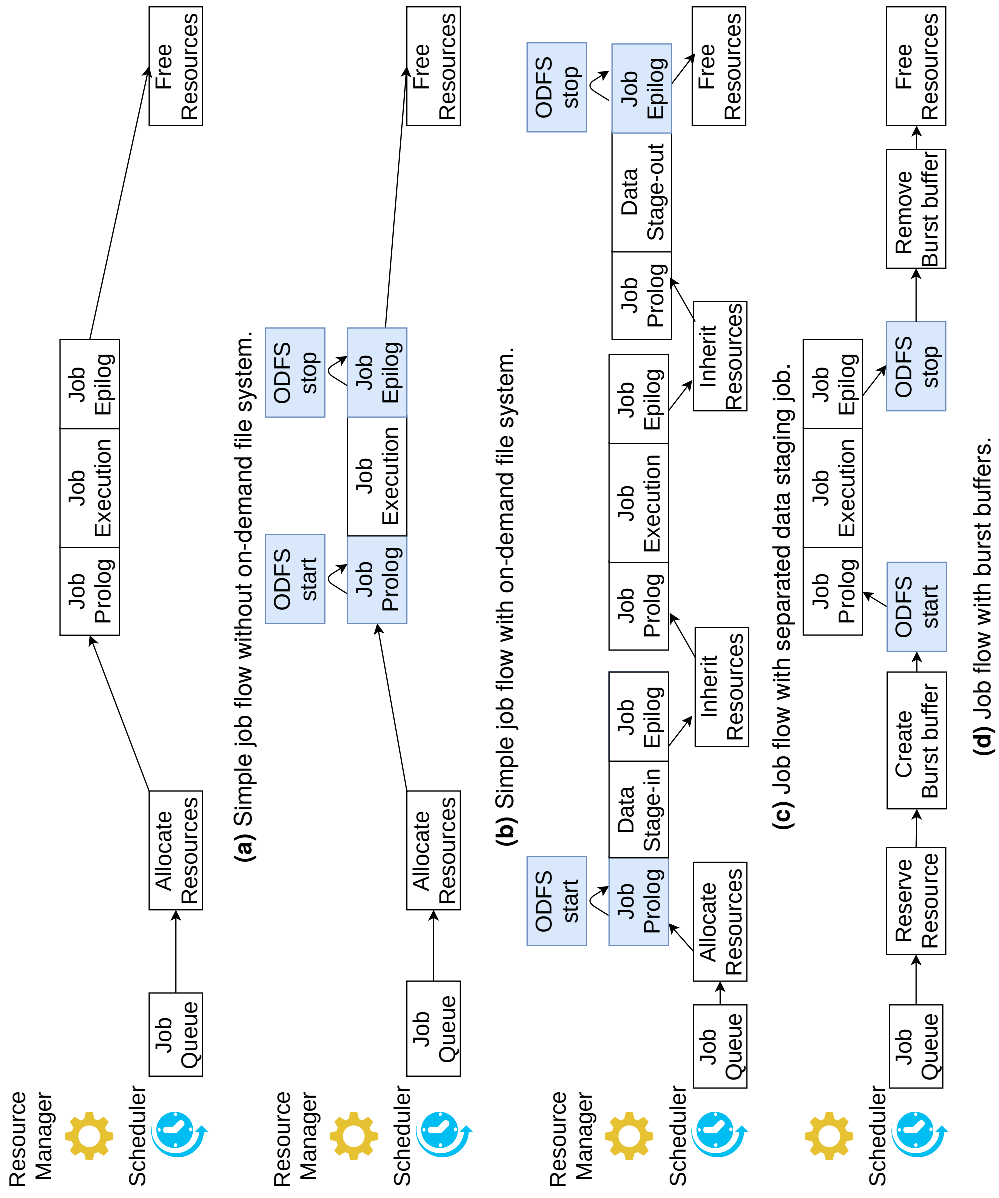
4.3 Generic Benchmarks

In this section, basic benchmarks are shown. These are necessary to answer basic questions, such as how long does it take to start and stop an ODFS or if there are other bottlenecks.

As initial benchmark IoZone [145] is used to measure the maximum throughput of read and write operations. Figure 4.3 depicts the throughput on the on-demand file system (solid line). The results show, that performance increases linearly with the number of nodes and that in this environment the SATA-SSDs are the limiting factor. The small throughput variation, that can be seen, occurs due to normal performance scattering of the SSDs [180]. The dotted lines indicate the possible throughput with assumed speeds of 7 000 MB/s read and 5 000 MB/s write performance for announced NVMe-SSD devices [112].

Another simple, but important procedure is to measure the initialization time of the on-demand file system. Table 4.1 lists the duration for starting and stopping the on-demand file system. The starting times increase up to 222 seconds on 256 nodes. The included tool BeeOND, to start the BeeGFS on-demand file system, has a parallel mode using a parallel distributed shell [181]. However, at startup a sequential region is processed, and by parallelizing this section the startup times for 256 nodes could be reduced to less than one minute [182].

Table 4.2 shows the performance for selected metadata operations. A comprehensive Table is in the Appendix A.2. Here, an ODFS (in this case BeeGFS) was started with the respective number of nodes with a default of one metadata server. To benchmark the metadata performance the MDtest [137] benchmark suite was executed. Three runs were performed and the mean values are shown in the table. The most important operations listed are creation, stat, and read of directories and files. Creating an object within a PFS is an expensive operation. To create an object an entry in the metadata server has to be created, also an initial object has to be created on an object storage server. Also the number file reads are important, because these operations also cause “locking” and “file open” operations. The values in Table 4.2 show that the number of metadata operations



(a) Simple job flow without on-demand file system.

(b) Simple job flow with on-demand file system.

(c) Job flow with separated data staging job.

(d) Job flow with burst buffers.

Figure 4.2: Jobflow sequence with different option to start an on-demand file systems.

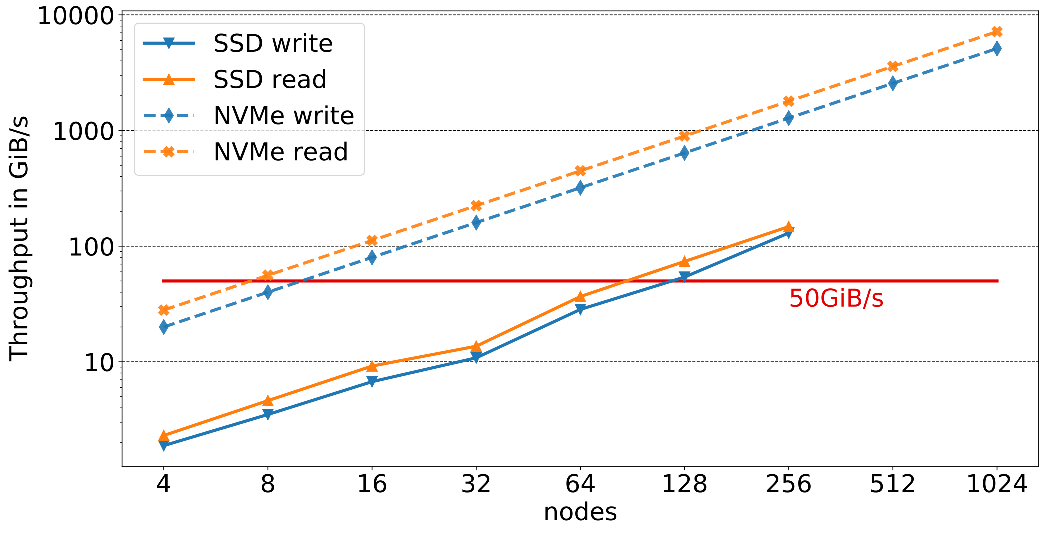


Figure 4.3: Measured read/write throughput (solid) and extrapolated (dashed) with announced speeds of NVMe-SSDs [112]. Horizontal red line shows the performance of the fastest PFS attached to ForHLR II.

Node	Directory creation	Directory stat	File creation	File stat	File read
4	6 208	104 989	10 384	69 001	23 542
8	7 649	129 858	19 186	101 042	36 261
16	7 620	123 337	26 761	113 917	38 302
32	7 447	115 644	27 208	114 139	38 056
64	7 217	115 369	26 799	103 164	37 305
128	6 702	113 868	26 464	105 713	34 888

Table 4.2: Depicted metadata operation per seconds with mdtest (`mdtest -d <odfs> -u -n 3600 -i 3`). A complete table is available in Appendix A.2.

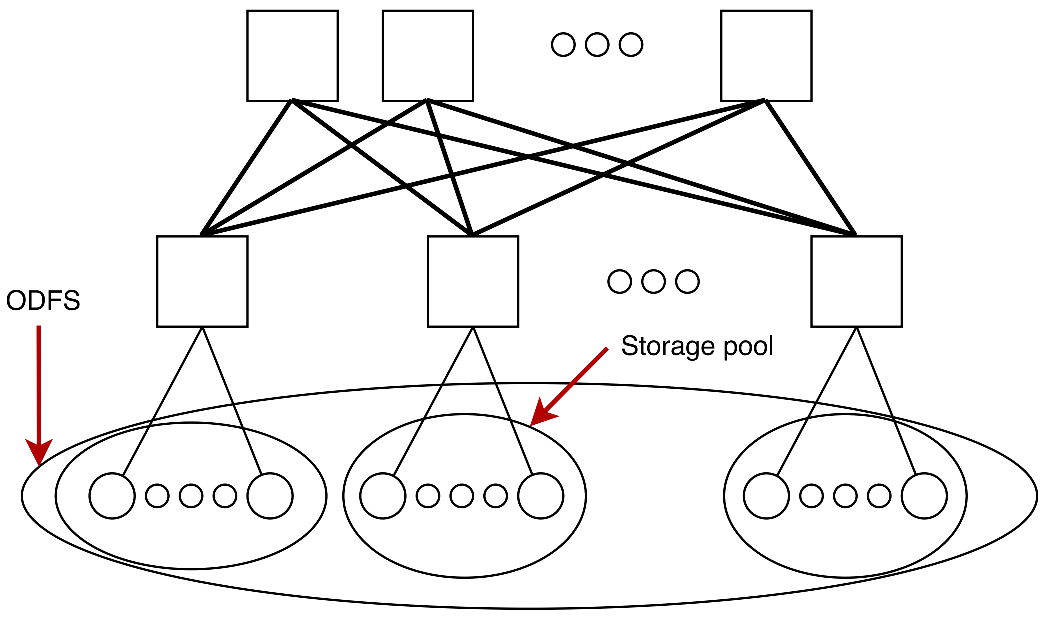


Figure 4.4: On-demand file system with a storage pool per leaf switch.

does not increase with the number of nodes. Comparing the values with Figure 4.3, it can be concluded that the number of nodes and thus storage targets can increase the throughput, but the performance of the metadata server remains constant. These values compared to the values provided by the dedicated global file system clearly show, that the global file system provides more metadata performance (cmp. Tables 3.1 and 4.2). However, it must be taken into account that the global PFS has dedicated servers for the metadata services and these are configured for this purpose. An on-demand file system uses an ordinary computing node. But using an on-demand file system, this metadata performance is exclusively available, while the metadata performance of the global file system is distributed to all users and jobs.

In a further test, the storage pooling feature of BeeGFS [166] has been investigated. The question here is, whether it is possible to deal with bottlenecks in topology with such a feature. For this purpose, an ODFS was initialized and so called storage pools were configured. According to the topology, an individual storage pool was configured for each leaf switch (see Figure 4.4). Directories can be associated with these storage pools. All data created within such a directory, will be stored on this specific storage pool. In other words, when writing to a storage pool, the data is distributed across the storage pool based on the stripe count and chunk sizes, but remains within the storage pool.

To create an artificial bottleneck within a cluster interconnect, available root switches were disabled. This reduces the available bandwidth in the network. To make the bottleneck more effective, the strip count was additionally set to maximum, which is done by setting the value to -1. This distributes a file across all storage targets and thus reduces the maximum bandwidth available. Figure 4.5 shows the write throughput for three scenarios. In the 3 different scenarios the maximum possible write throughput (red line)

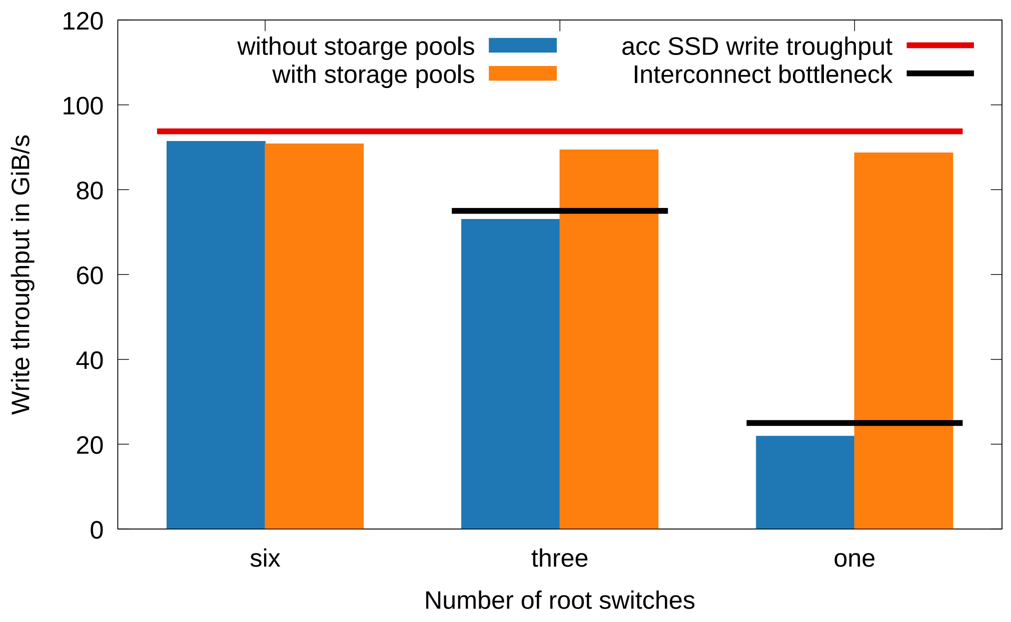


Figure 4.5: Measured write throughput with IoZone and reduced number of core switches on 240 nodes. Black line shows the accumulated SSD write performance. Black lines show the corresponding interconnect bottleneck.

Number of root switches	6	3	1
SSD write performance (GiB/s)	93.75	93.75	93.75
max. uplink bandwidth (GiB/s)	150.00	75.00	25.00
write w/o storage pool (GiB/s)	91.37	73.01	21.88
write with storage pools (GiB/s)	90.78	89.35	88.66

Table 4.3: Write throughput performance with reduced network capacity

remains the same. 240 SSD were used with 400 MiB/s per SDD resulting in 93.75 GiB/s available write throughput. The network capacity when using 6 root switches is around 150 GiB/s. If only 3 root switches are used, the network capacity is reduced to 75 GiB/s and with only one root switch 25 GiB/s (black line). In the first scenario, with six root switches there is a very small performance loss when using storage pools, which indicates a low overhead. With a reduced number of core switches, throughput decreases due to the lower network capacity without the use of storage pools. If the topology is taken into account and storage pools are created accordingly, it is possible to achieve the same performance as with full expansion (cmp. Table 4.3). Of course, it is important to keep in mind, that the network bottleneck only for I/O is overcome. If the application is communication intensive, then it will be slowed down by such a network bottleneck.

4.4 Scientific Use Cases

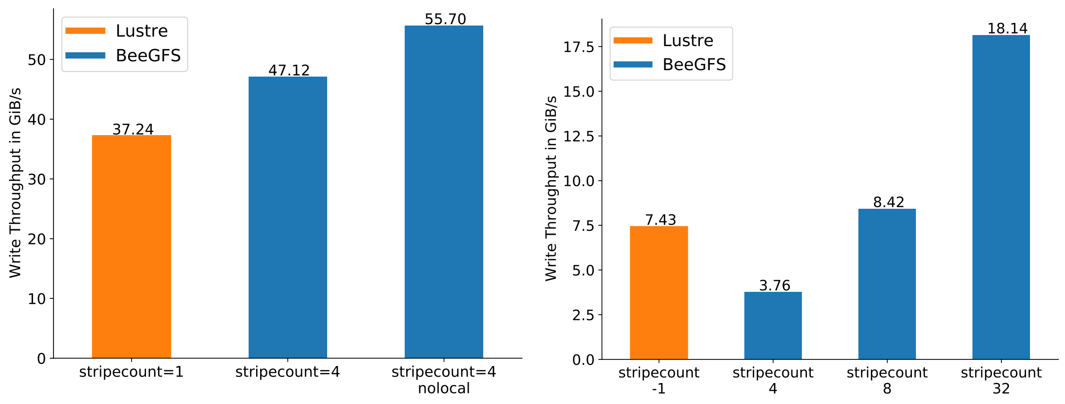
In this section, scientific use cases from SCC users are shown. The chosen applications and use cases in question have created enormous loads on the storage system and are therefore good candidates for evaluation. Details about the application are available in Section 3.3. The evaluated use-cases are relatively short regarding execution time, but it has been ensured that the cases are representative. The HPC system, where the testing has been done was in normal operation, so it had to be ensured that the impact to the operation and other running jobs is minimal.

4.4.1 super_SPH

The first application, which was investigated with an ODFS, was super_SPH [18]. The application is used with two workflows. One workflow is writing the results with a file per process method. The strategy that each process writes intermediate results to its own files is colloquially called file-per-process, even if one process writes multiple files. Afterwards, post-processing jobs are executed to aggregate the distributed files and write the results according to time steps. A new implementation is now writing directly the results into time steps format, using MPI-IO which makes the data post-processing unnecessary. Unfortunately the new implementation, using MPI-IO to write the results, was multiple times slower than using file-per-process. Both implementations were evaluated to check if better performance, with an ODFS, can be achieved. For both scenarios, 256 nodes were used, which serve as computing resources as well as storage for the on-demand file system.

Observation has shown, that file-per-process method is causing heavy load on the PFS and using MPI-IO is slower but has less impact on global PFS. Figure 4.6(a) and 4.6(b) show the results of super_sph when writing directly to the global parallel file system (Lustre) and to an on-demand file system. Both figures show already the optimal parameter set, for the global PFS (Lustre), using a stripecount = 1 for the FPO implementation and - 1 when using MPI-IO. While using the simple file-per-process method, small performance increase is gained. The application is already pretty close to the maximum throughput of the global file system (50 GiB/s). Interesting here is the performance leap, that can be achieved if using one extra node, illustrated by the bar with the option “nolocal” (Figure 4.6(a)). On this extra node, only the metadata service is started, but not the scientific application. With an extra node for metadata services, a higher throughput of 8.5 GiB/s can be achieved,

When using MPI-IO it is important to choose the right parameters for the on-demand file system. With the default values (stripcount = 1) of the on-demand file system even a deterioration can be seen, although SSDs are used here. Here the throughput increases from 3.76 GiB/s to 18.14 GiB/s, when changing the default value for the stripecount from 4 to 32.



(a) Write troughput with file-per-process method. **(b)** Write troughput with MPI-IO method.

Figure 4.6: Different write throughput, based on implementation on file system configuration.

File system used	Lustre	BeeGFS
fastest run time in (s)	304	319
slowest run time in (s)	435	326
average run time in (s)	335	323

Table 4.4: NAStJA total run time on on-demand and global file system. Average of 5 runs.

This application was noticed by the operator, as it generated high loads on the file system for a short time. By using an ODFS the high loads are completely shifted to the ODFS and by choosing the right parameters, the throughput can be increased significantly.

4.4.2 NAStJA

The second investigated application was NAStJA[151]. The application was selected, because it caused significant loads on the storage system. In addition to the increased I/O load, the user has noted, that the application has large variations in its run time. Here, an investigation how the application and the corresponding use cases perform with an ODFS was conducted. The application was run 5 times each on the global PFS and on an ODFS. For the evaluation NAStJA was run with 4800 parallel processes which are started on 240 compute nodes with 20 processes each node. An extra node is added and reserved for the metadata service, totaling 241 nodes. 4 800 blocks were chosen, i.e., one block per core. Also each block has a size of 1 MB each and a write is performed every 20th time-step for a total of 100 000 time-steps. Table 4.4 shows the run times of the NAStJA runs. One of the runs on the global file system was the fastest, but the average run time was faster with an ODFS. The variance in run time differs significant. On the global PFS, the time varied from 304 s to 435 s. When using an ODFS, the time varied only from 319 seconds to 326 seconds. It is remarkable, that a smaller variance occurs using ODFS. The compute nodes have to manage an ODFS while running the application. This would usually cause an interference.

During the simulation the execution time of each time-step was recorded. Figure 4.7 shows the average execution time (blue line) per time-step. The black bars represent the fastest and slowest execution time for the corresponding time-step. Figure 4.7 a) shows the average time for the time-steps when using the on-demand file system. Figure 4.7 b) shows the same simulation on the global PFS. Using the global PFS the variation of the time-steps is significant, here the fastest time steps are around 0.02 seconds and slowest over 0.1 seconds. By using an on-demand file system, there is almost no deviation in the execution time visible.

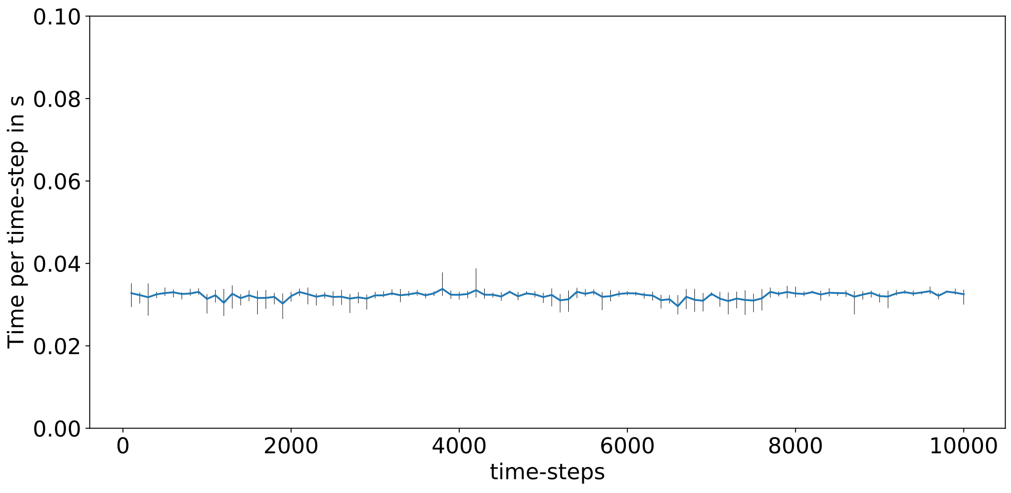
This application generated high loads on the file system during the whole simulation. The user had to make sure, that only one of these jobs was running at a time. By using an ODFS the high I/O loads are completely shifted to the ODFS and the execution time is more stable with a more predictable execution time.

4.4.3 OpenFOAM

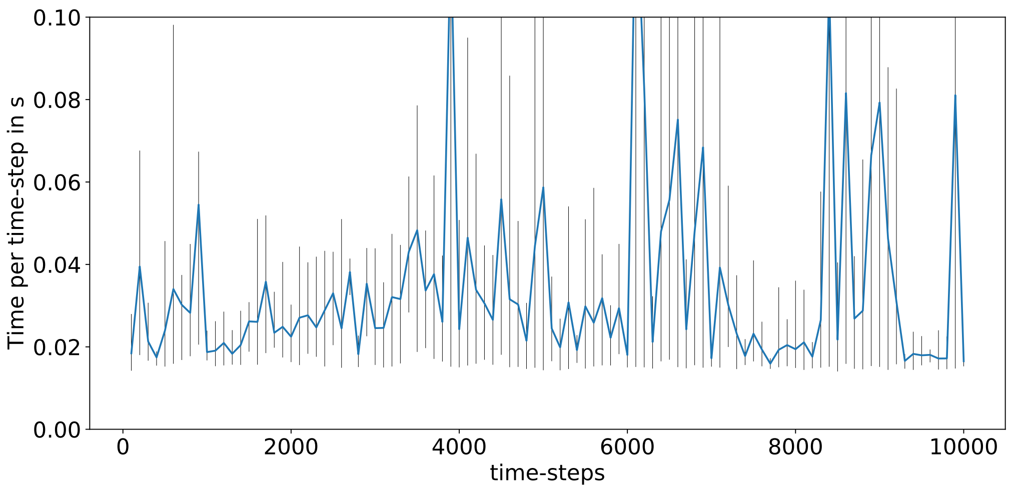
Another evaluated application was OpenFOAM [147, 148] and here two different use cases were evaluated. The respective jobs have usually a run time of about 24 hours. These use cases were selected because they have created a continuously high load on the file system. Figure 4.8 shows the load caused on the storage servers. Blue lines show the UNIX load generated on the metadata server. The average load of all object storage servers is shown by the orange line.

Use case 1: A simulation of a burner flame of laboratory scale [183]. The user runs the simulation on 5 000–28 000 CPU cores [184]. On the ForHLR II, the simulations take place on a few thousand cores. The challenge in terms of I/O comes from the most efficient I/O strategy OpenFOAM offers for check-pointing: Each time the application generates intermediate results, each process writes one file per solution variable, which can be concentration, pressure or temperature. Due to the large scale of this simulation, a large number of files is generated. In this thesis, the case was run on 240 nodes or 4800 processes, resulting in the creation of 95 files per process or half a million files in total, with about 0.2 MB per file or 25 MB per process or 120 GB in total. It should be noted that half a million files are created at each intermediate result. In other simulations with 28 800 CPU cores the number of files increases to 2.7 million.

Use case 2: In this use case, a mixture of methane and air in a pipe leading to the burner from use case 1 is simulated. The calculation mesh consists of 150 million cells. The case was executed and evaluated on 4 800 processes (240 nodes). The number of solution variables and thus the I/O requirements for check-pointing are much lower



(a) NAStJA using on-demand file system



(b) NAStJA using global file system

Figure 4.7: Average execution of time-steps for five NAStJA runs. Black bars showst^{ms}he minimum and maximum values of each process.

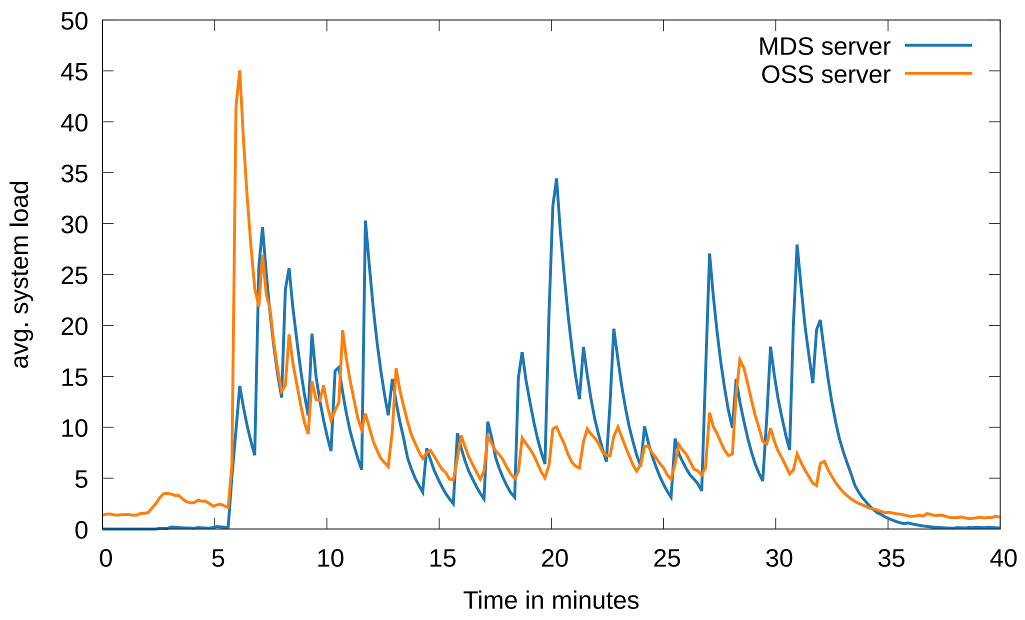
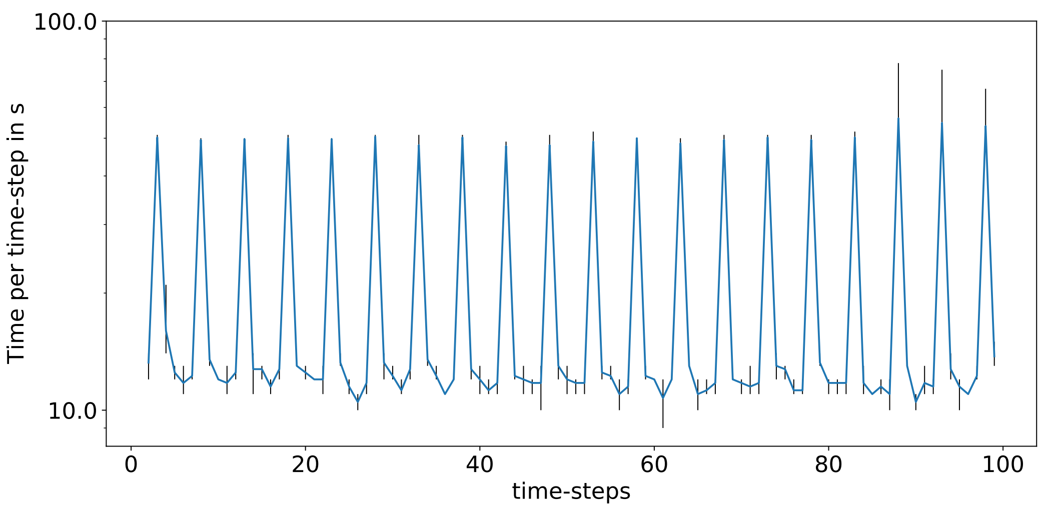


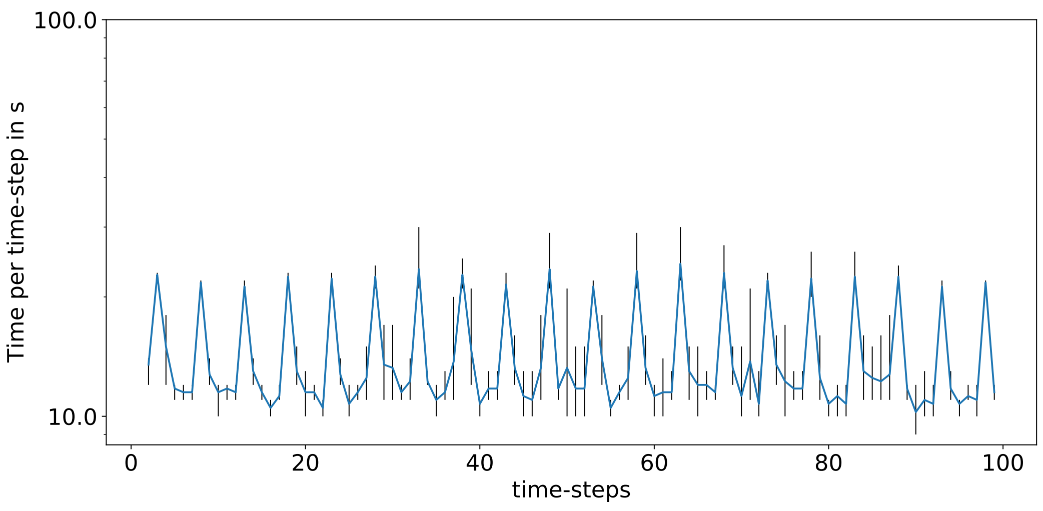
Figure 4.8: Load generated by the OpenFOAM use case. Minutes 0–5 and 35–40 show usual load during normal operation.

compared to use case 1, because there is no flame in this setup and therefore much less equations have to be solved. In this case, the I/O challenge results from the run time post-processing. This use case is the precursor simulation to use case 1. After each time step the solution variables are written, which then serve as inlet conditions for the simulation of use case 1. The post-processing data at run time, which consists of about 20 small files per time step, is written at high frequency during the entire simulation time. Figure 4.8 shows the load generated of a 30 min run.

Figure 4.9 shows the average execution time for each time step for use case 1. The use case was started 5 times on 240 nodes, both on the global PFS and the ODFS. The blue line is the average execution time of the time steps. The black bars represent the min/max values for the corresponding time steps. The high spikes in both cases occur when the application is writing intermediate results. Figure 4.9(a) shows the results when OpenFOAM is writing to the ODFS and 4.9(b) when using the global file system. The spikes are higher when using the on demand file system. These higher spikes are caused, by the fact that the on-demand file system has to share resources with the application. But the deviation (black bars) are higher when using the global file system. Here the advantage of the dedicated bandwidth and meta data performance results in less deviation, when using on demand file systems The second use case shows a slightly different behavior (see Figure 4.10). Using the ODFS leads to less variance in the execution of the time steps. But in the time steps where intermediate results are written, the time increases significantly. With the on-demand file system these steps need about 50 seconds, while using the PFS the same time-steps need on average under 25 seconds (compare Figures 4.10(a) and 4.10(b)).

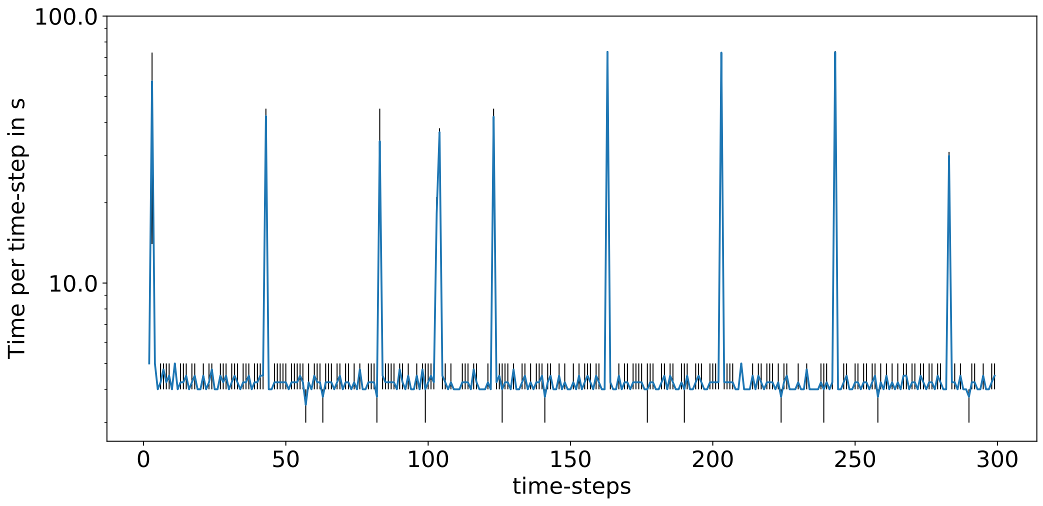


(a) OpenFOAM use case 1 on-demand file system.

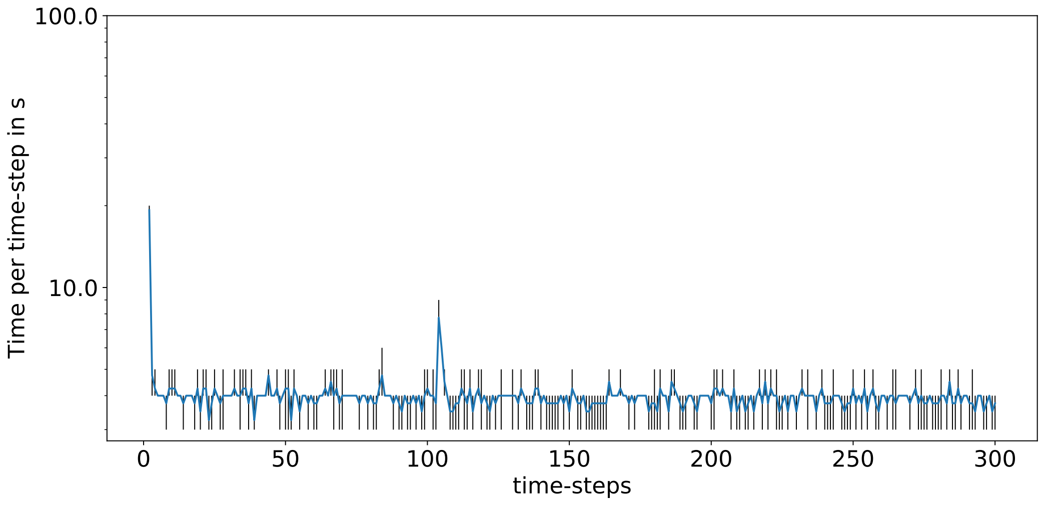


(b) OpenFOAM use case 1 global file system.

Figure 4.9: Average execution of time-steps for OpenFOAM use case 1.Black bars shows the minimum and maximum values of each process.



(a) OpenFOAM use case 2 on-demand file system.



(b) OpenFOAM use case 2 global file system.

Figure 4.10: Average execution of time-steps for OpenFOAM use case 2.Black bars shows the minimum and maximum values of each process.

	Use case 1		Use case 2	
File system used	Lustre	BeeGFS	Lustre	BeeGFS
Initial time-step in (s)	41	340	12	111
total computation time (s)	1331	1481	1111	1340
Clocktime in ()s)	1416	2335	1195	1906

Table 4.5: Initial time-step for OpenFOAM and total execution time.

Table 4.5 shows the average execution time for both OpenFOAM use cases. Here a clear slowdown of the two use cases can be seen. The total clock time of both cases has increased significantly when using the ODFS, for use case 1 from 1416 to 2335 seconds (approx. 65 %) and for the second case from 1195 to 1906 (approx. 60 %). The total computation time has increased by only 11% for the first case and 20 % for the second use case. Here the application was slowed down considerably by the I/O to the ODFS. The first time-step is also interesting. In this step the simulation data is being read. Here in both cases a multiple of the needed time can be seen (over eight-fold for use case 1 and nine-fold for use case 2).

Although the total run time of the two OpenFOAM use cases is much longer, the use of an ODFS is reasonable. Otherwise, the load which would be generated on the global storage system would be problematic for the whole system and affect all users.

4.4.4 Swarm-based Meta-heuristic

The last application is a swarm-based meta-heuristic search [152]. It was chosen because it creates an enormous I/O load on the storage systems. The use case was started with 50 nodes and regularly generated so-called Lustre Client evictions on the compute nodes. This caused the node to fail and the job to be aborted. The failed node had to be rebooted to resume normal operation. The application simulation is using so-called generations and after each generation, the intermediate results are evaluated and thus the next generation is initiated. Usually such a job iterates about 100 generations, which takes a little over 24 hours. Figure 4.11 shows the total I/O operations that are performed for the simulation of only one generation. In about 20 minutes (one generation), over 30 000 000 metadata operations are generated. The Figure 4.12 shows the average run times of jobs, both on the global PFS and the ODFS. Here again two decisive results can be seen. On the one hand the runs on an ODFS (26:55 min) are slower than on the global PFS (20:52 min), on the other hand there is less variance in the run time when using the ODFS (50 seconds). Figure 4.12 lists the file operations over time. The figure shows a processing of one generation on the ODFS, which is divided into 3 phases. At the beginning the simulation including processing the intermediate results are performed (about 800 seconds), with up to 60 000 metadata operations per second are generated.

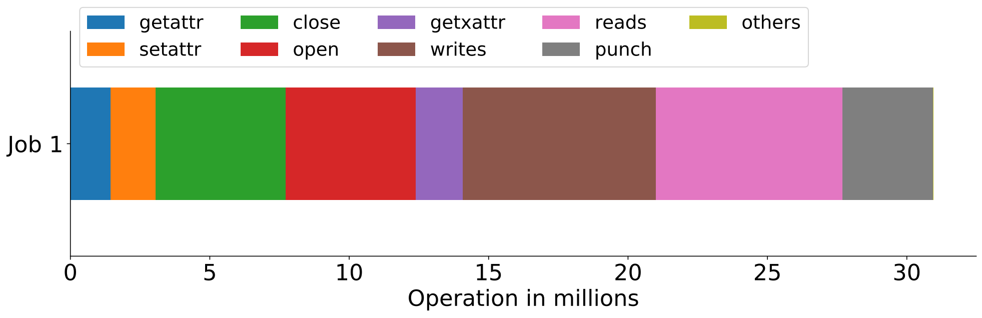


Figure 4.11: Total I/O operation created for one generation within approx. 20 min.

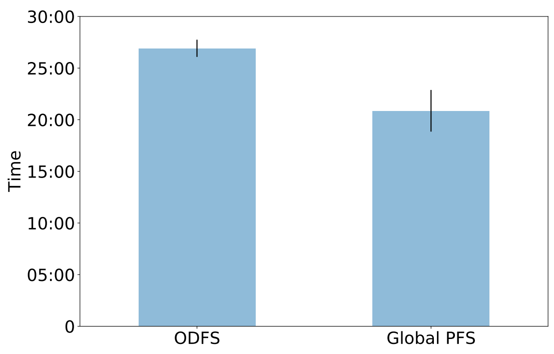


Figure 4.12: Average run time for one generation.

After the peak of IOPS there is a phase in which almost no metadata operations are performed, but are used for computing (800–1700 seconds). At the end (1700 seconds) there is a very small bump, these are operations for the stage-out to the global PFS. A example for executing two generations can be found in Appendix A.1.

Apart from the results, there is no other option for this use case, than to use the ODFS. The operational problems caused by the vast number of metadata operations make it almost impossible to use the global file system of an HPC system. Like mentioned above, due to heavy usage of the global PFS (Lustre), the compute nodes got evicted from the lustre file system. These compute nodes have to be rebooted.

4.5 Data Staging

Also the case of copying data back to the PFS while the application is running, has to be considered. For this purpose, a dedicated use case with different NAStJA simulations has been used. With NAStJA it is very easy to create different setups in order to use the requested cores to their optimal utilization. In the configured setups NAStJA has been executed on 23 nodes, which kept the application inside a leaf switch and allowed to eliminate any interference with other applications.

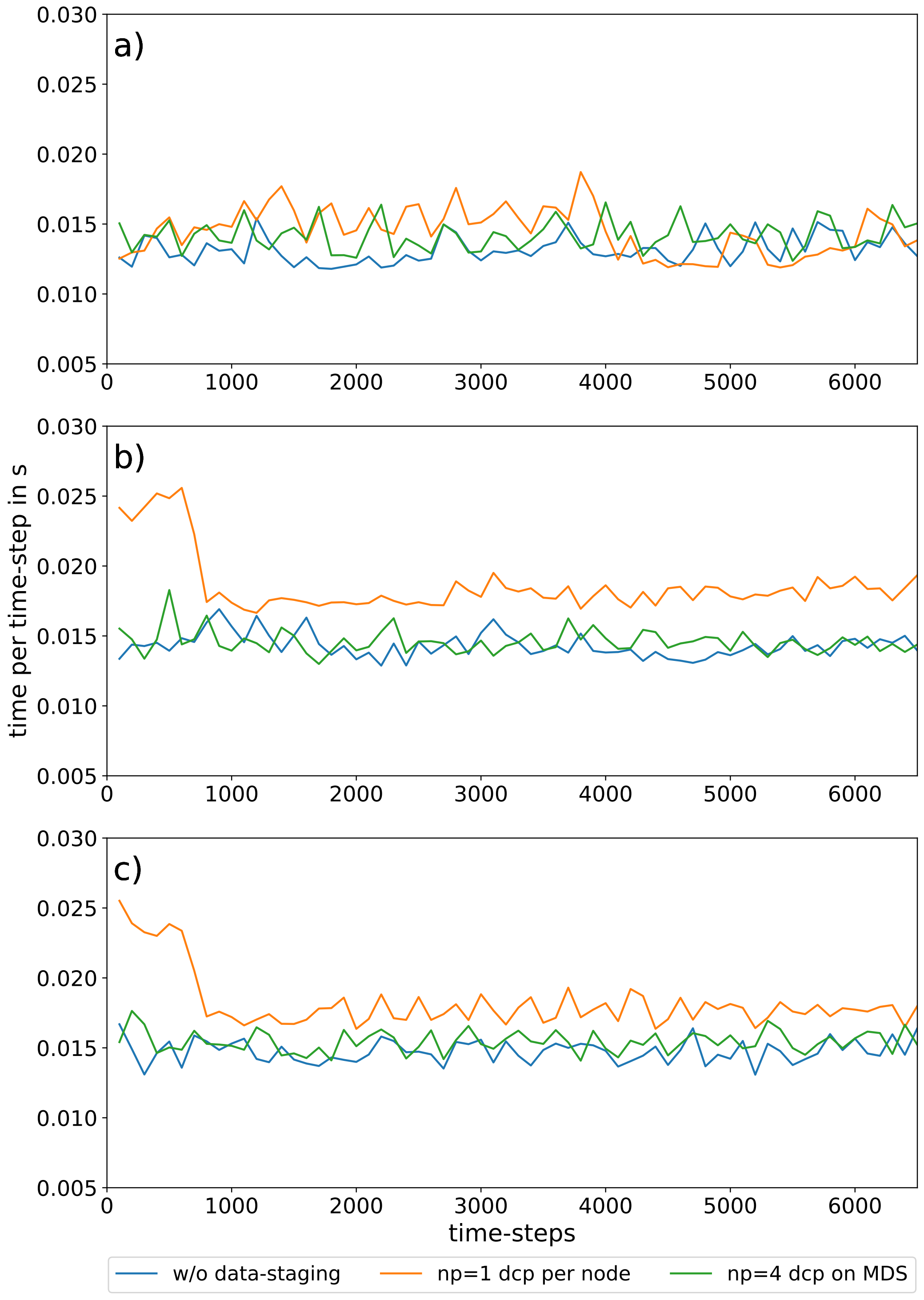


Figure 4.13: Total execution time of NAStJA for (a) 16 cores per node (b) 19 cores per node and (c) 20 cores per node.

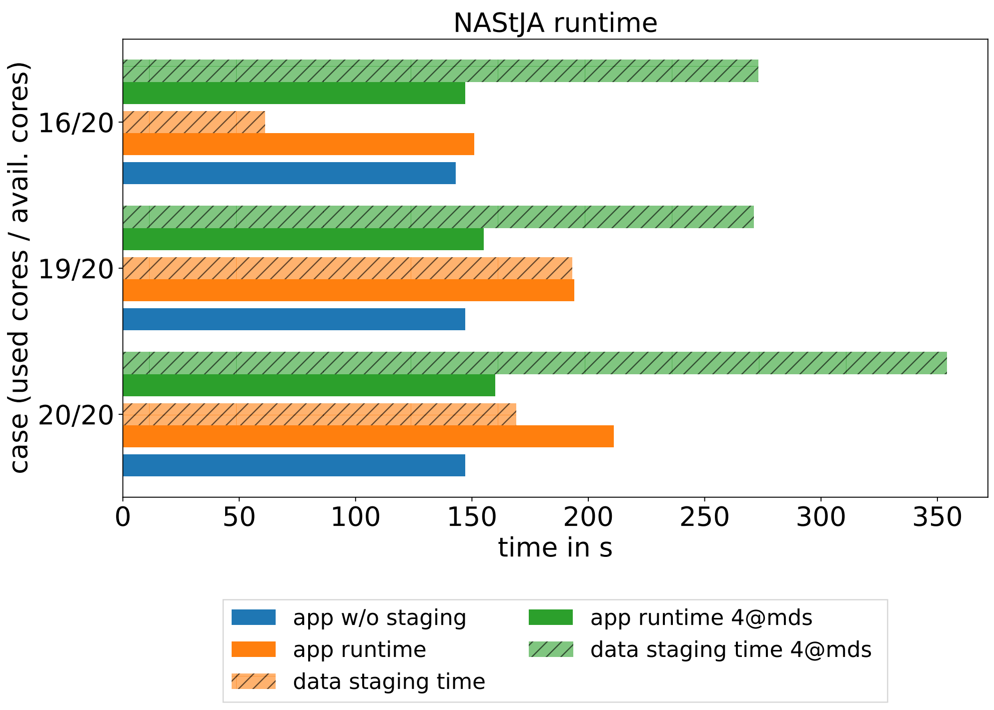


Figure 4.14: Total execution time (NAStJA) for application w/o data staging.

Another node has been added to the 23 nodes, which was used as additional storage node for the ODFS. On this node the scientific application was not running, but the MDS service of the ODFS was started. NAStJA was started in 3 different setups with 20, 19, 16 cores per node (max 20 available). With these different setups, it can be examined if free cores reduce the interference of the ODFS.

On each of these setups, three different scenarios were examined.

1. Data stage-out using all nodes with one copy process per node.
2. Data stage-out using the additional MDS node using 4 copy processes.
3. No data stage-out as comparison.

This allows to evaluate the impact of free resources for concurrent data staging. The remaining resources (cores) on the compute nodes are then available for the on-demand file system and data staging. To stage the data, during the NAStJA execution, the parallel copy tool dcp [185] was used. Figure 4.13 shows the average execution time per time-step of five runs in the different scenarios. With 16 cores for the application, from available 20 cores, the average execution times are similar whether the run was executed with or without data staging. When using 19 or 20 cores per node, the application is slowed down when the copy is executed with one process per node. At the beginning, the slowdown is significant (orange line) due to the high amount of metadata operations. In this case, a portion of the data is indexed on every node and this is causing interference with the application. When using only the MDS-node to copy the generated data (green line) the indexing is done only on the node with the MDS-service. Here, only a small influence can be observed, such that the interference to the application is negligible. Figure 4.14 shows the total time for the application run and data staging experiment. Using only the MDS node, to copy the data, needs more time than the execution of the simulation. If there are enough free resources on the compute nodes, the data can be staged-out without slowing down the application.

4.6 Summary

In this chapter, it has been shown how the node local SSDs can be used to provide a high performance parallel file system. Different concepts have been presented, which allow a seamless integration into a HPC system. Depending on the available functionality of the batch system, different solutions are available. With consideration of the topology, bottlenecks in the network can be avoided. This allows to create a very high throughput ODFS with a weak interconnect. The time for starting and stopping the ODFS is a matter of judgment. A few minutes to start the ODFS is acceptable if the job runs for several hours, but not suitable if the job takes seconds or minutes. The scientific applications have all been characterized by the fact, that the use cases investigated have all caused significant problems for the entire system. Users were asked, to ensure that only one of their jobs was running at any given time. In some cases, individual nodes have frozen and could only be resumed by a reboot. The result for all these cases, when using an ODFS, is a reduced influence on the entire system and no limitation of the concurrent jobs. Finally, it was shown that even concurrent copying of the data is possible under certain conditions. However, it is important to consider whether this have to be done during the simulation.

Although, the presented applications are somewhat slower with an ODFS, it makes sense to offer an ODFS as an additional feature. Applications that generate significant I/O loads are given a chance to use the HPC system without interfering with other users and applications. A single user might be able to make sure that only one of their I/O-heavy jobs is running at a time, but they can hardly coordinate this with other users.

5 Data Scheduling

“Plans are worthless, but planning is essential.”
— Dwight D. Eisenhower [186]

In the previous chapter, deployment of the ODFS was shown. This chapter answers the question of how much time is available to stage data. Figure 5.1 illustrates the approach, to stage data to a temporary on-demand file system. Here the scheduler is of special interest. Start times of the job are planned in advance by the scheduler. As already mentioned in Section 2.4, scheduler plans are dependent on user given wall time estimates. These user-defined wall time estimates will be examined in more detail. Especially, if wall time estimates can be improved in an automated way. Also, an evaluation of how good wall time estimates have to be, will allow to make a statement of how much time is available, to stage data in advance.

This chapter first gives a short overview about related work to improve wall time predictions as well as an overview of scheduling simulators. Afterwards, metrics are briefly explained. First an introduction on metrics to measure machine models, and then metrics regarding resource allocation prediction are introduced. Next, a method is presented how to better predict the wall times. Whether the accuracy is sufficient for the intended purpose is shown afterwards.

The investigation and some of the results discussed in this Chapter have originally been published:

- M. Soysal, M. Berghoff, and A. Streit. “Analysis of Job Metadata for Enhanced Wall Time Prediction”. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2018, pp. 1–14 [187],

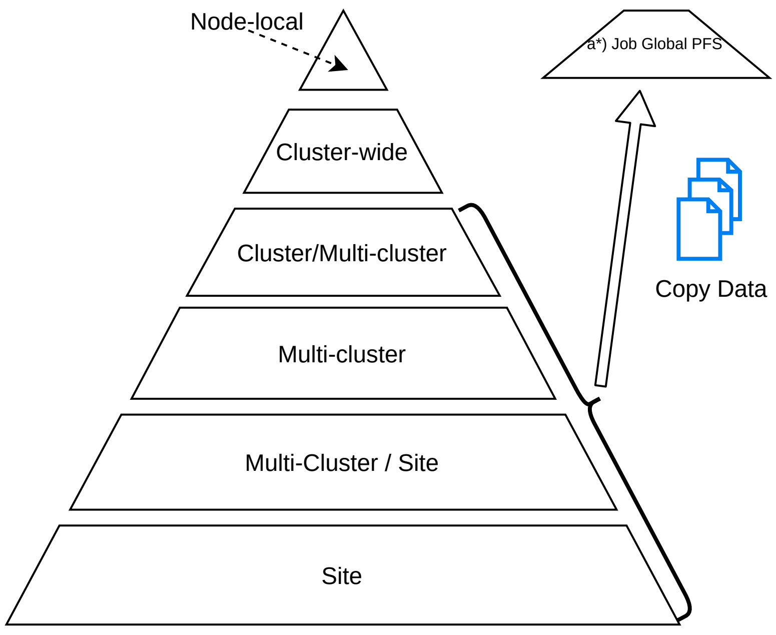


Figure 5.1: Data staging to an on-demand file system

- M. Soysal and A. Streit. “Ad-hoc file systems at extreme scales”. In: *High Performance Computing in Science and Engineering’19 (HLRS)* (10/2019). Accepted [156],
- M. Soysal, M. Berghoff, D. Klusáček, and A. Streit. “On the Quality of Wall Time Estimates for Resource Allocation Prediction”. In: *Proceedings of the 48th International Conference on Parallel Processing: Workshops*. ICPP 2019, Kyoto, Japan. New York, NY, USA: ACM, 2019, 23:1–23:8 [188].

5.1 Related Work

Related work in this chapter has to cover two areas. On one hand, the topic of wall time prediction and on the other hand, the topic of scheduling simulators are briefly introduced.

5.1.1 Background on Walltime Prediction

For efficient utilization of large-scale HPC systems, the task of resource management and job scheduling is of highest priority. Therefore, modern job scheduling systems require information about the estimated total wall time of the jobs already at submission time. Proper wall time estimates are a key for reliable scheduling decisions and can improve the scheduling performance [189]. However, the user requested wall time estimates are not close to the real used wall time. The approaches to improve wall time predictions are divided into two areas. The first, are classical mathematical models, sometimes in combination with rule-based methods. Second, approaches with methods from the field of machine learning have become popular. In the following a short introduction to both areas is given.

Traditional Methods Gibbons [189, 190] and Downey [191] used historical workloads to predict the wall times of parallel applications. They predict wall times based on templates. These templates are created by analyzing previously collected metadata and grouped according to similarities. However, both approaches are restricted to simple definitions. Other approaches try to predict the accuracy of user wall time estimates [192, 193]. Here, the deviation between the user estimated wall time and used wall time is determined and applied to new jobs from the user. An example, how this method is used, will be explained on the implementation within the Alea Scheduler [194]. Alea uses previously known run times, of already completed jobs, to establish a run time estimate for a newly arriving job. It is working on a per-user basis, i.e., a new run-time estimate for a given job is computed using information about previous jobs of that user. Each time a given user's job completes its execution, the predictor computes the ratio of the used wall time σ, the fraction of the job's actual run time and the user's estimate

$$\sigma(i) := \frac{T_{\text{Run}}(i)}{T_{\text{Req}}(i)}, \tag{5.1}$$

where i is the i-th job by the specific user, i.e., σ measures to what extent the estimated wall time was actually used. Since the user's estimate is the upper bound of job run time[1], σ is between 0.0 and 1.0 representing the relative usage of requested wall time. This method measures how much a user overestimates the job's run time. Once σ is computed, it is used to predict wall times. The predictor only uses the last five (most recently completed) jobs, when computing the prediction. First, it computes the σ of those five recently completed jobs and then chooses their maximum. Next, it multiplies the job's wall-time estimate $T_{\text{Req}}(i)$ by this maximum. $T_{\text{Pred}}(i)$ is defined as,

$$T_{\text{Pred}}(i) := T_{\text{Req}}(i) \max_{i-5 \leq k \leq i-1} \{\sigma(k)\}, \tag{5.2}$$

the predicted run time.

Machine learning methods In the recent years, machine learning algorithms were used to predict resource consumption in several studies [78, 190, 195–197]. As an example, Smith [198] is using greedy and genetic search techniques for estimating queue wait time and job run time. The method is based on machine learning and genetic algorithms are used to refine input parameters of the method. This system is used by XSEDE to predict queue wait time [199].

[1]The system is configured to kill a job if it exceeds user's wall-time estimate.

Other simple, but popular methods are using linear regression for predictions [200, 201]. Advanced methods such as Nearest Neighbors [202, 203] or Regression Trees [204], Instance Model Learning [198, 205, 206], or a combination of multiple machine learning methods as shown by Matsunga [207] which have been used for predicting the execution time of two bioinformatics applications: BLAST [208] and RAxML [209].

In contrast to these mentioned approaches, this thesis does not optimize machine models by hand. This would require an unfeasible effort to parameterize and optimize the models. Therefore, the use of AUTOML to perform this work is of great importance. This facilitates the work and allows an automated training of models.

5.1.2 Background on Scheduling Simulator

There are several scheduling simulators that imitate the behavior of a batch system. Using real or artificially generated workloads, it can be examined how, e.g., scheduling algorithms behaves. Following is a short overview of common scheduling simulators.

SimGrid [210] is a C-based scheduling simulator used for the simulation and development of distributed applications for heterogeneous systems. It uses a collection of core abstractions and functionalities to simulate specific application environments and topologies. Bricks [211] is a performance evaluation system, that allows comparison and analysis of different scheduling algorithms for HPC systems. Bricks simulates different patterns of computing systems, including network behavior and scheduling algorithms. It is designed for the simulation of client-server architectures in grid systems. GridSim [212] is a Java based flexible and modular grid simulator. It is platform independent and uses the simulation package SimJava2 to simulate discrete events. GridSim simulates a grid environment and its behavior. This is done by using simple implementations of common objects, like job or user. It also supports simulation of network topologies and data stores. Alea [193] is an extension of the GridSim toolkit to simulate various scheduling problems in grid systems. Alea simulates scheduling and execution of non-preemptive jobs. Also, Alea supports a queue-based and schedule-based scheduling and offers a variety of scheduling algorithms. Additional, it has a built-in wall time prediction features.

To investigate the resource allocation prediction, Alea was chosen as scheduling simulator and the feature set was enhanced. During simulation the time, when a node allocation is created, is now recorded. This enhanced version of the simulator that has been used in this work can be found at GitHub [194].

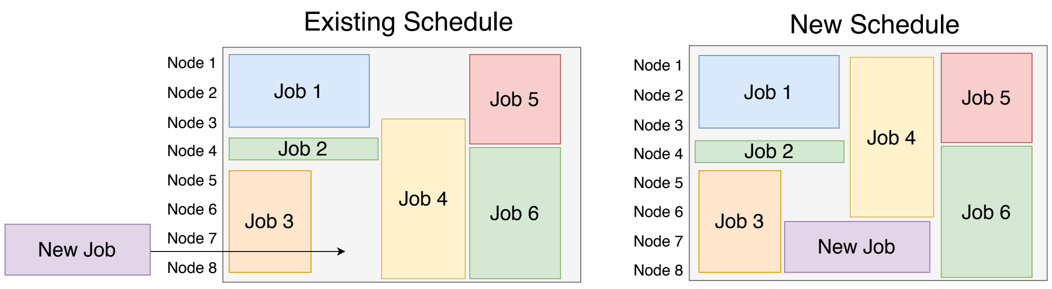


Figure 5.2: Re-scheduling of jobs to other nodes so that a new job can be back-filled.

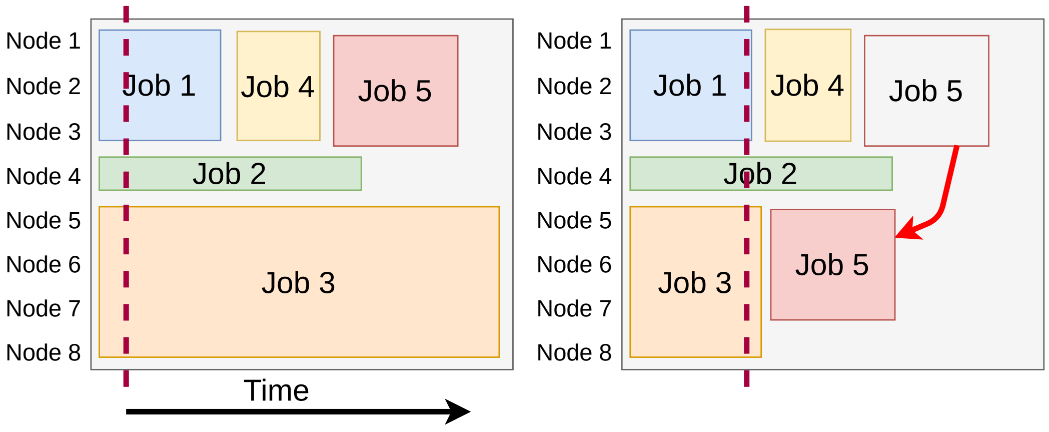


Figure 5.3: Job 3 finished early. Job 5 get rescheduled from node 1-3 to node 5-7.

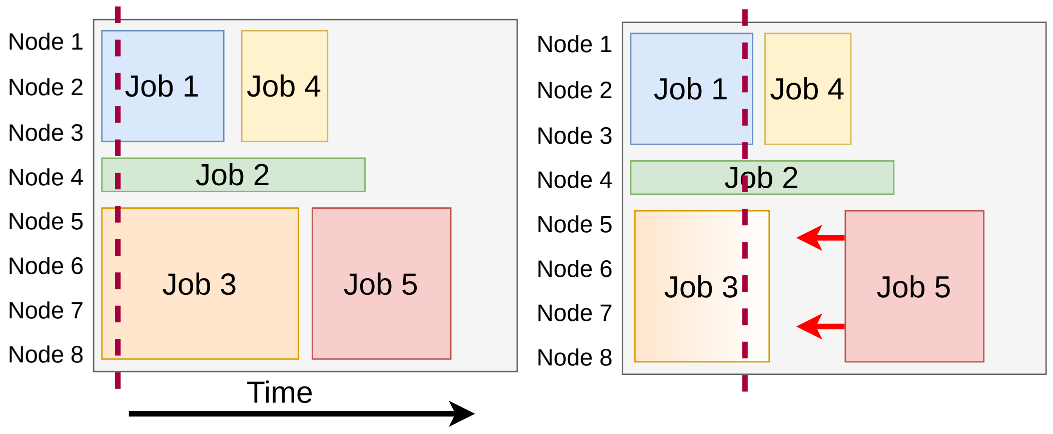


Figure 5.4: Job 3 finishes early. Job 5 jumps forward. Same nodes but earlier start time.

5.2 Real World Situation

One of the main tasks, of such schedulers is to optimize the utilization of resources. A few examples are illustrated in the Figures 5.2, 5.3, and 5.4. In the first case, Figure 5.2 shows how back-filling a new job, changes the schedule. To keep the workload high, all jobs are rescheduled. Job 4 is moved to a different set of nodes, but the start time remains the same. Rescheduling a job on a different set of nodes can also happen due to jobs finishing too early. An example is given by Figure 5.3. The red bar represents the time of the job scheduler, everything to the right of it would be the future schedule. Here Job 5 is shifted to a different set of nodes and the start time is also moved forward. The third case (Figure 5.4) shows what happens, when jobs finish earlier than scheduled. In this case, Job 3 ends earlier and therefore Job 5 can be started sooner.
In the first case, a premature ODFS creation would have happened on the wrong nodes and in the third case, a job might start too early before an ODFS has been created or data has been staged. The second case represents both problems, a too early job start and wrong node set. Besides the illustrated reasons, that make allocation predictions of the HPC scheduler unreliable, there are more events that cause similar situations. For example, technical failures could cause individual nodes to fail or short-term reservations set by admins would also cause a re-schedule. In this work such situations have been ignored. It is quite clear, that such situations occur on real systems. In Appendix A.2, a scheduling simulator is introduced, that is able to investigate such situations. This simulator is developed at SCC, and started as a student project. Nevertheless, the focus during the conducted evaluation, is unreliability of the planned run times.

5.3 Parallel Workloads Archive

The Parallel Workloads Archive (PWA) is a well established HPC workload trace repository. It contains detailed workload logs collected from large scale parallel systems. Each archived workload log is formatted in a uniform format – Standard Workload Format (SWF). The SWF was defined to ease the usage of workload logs and models. With this format, analyzing workloads or imitating HPC system scheduling need only to parse a single format [139, 213]. Each data set is represented by a sequence of lines (jobs) containing 18 columns (job metadata) per line.

1. Job Number – Unique job identifier, also called JobID.
2. Submit Time – seconds starting from workload log time.
3. Wait Time – difference between Submit Time and Start Time in seconds.
4. Run Time – the actual time in seconds the job was running
5. Number of Allocated Processors – integer value of allocated cores or CPU, depends on configuration
6. Average CPU Time Used – both user and system, in seconds.
7. Used Memory – average used memory per core in kilobytes.
8. Requested Number of Processors
9. Requested Time – Wall time requested for job
10. Requested Memory – requested memory per processor in kilobytes
11. Status – a number indicating the reason why job has finished. 1 = job was completed normal, 0 = failed, and 5 = canceled. If this field can not be provided it is -1.
12. User ID – a number identifying a user.
13. Group ID – a number identifying a group.
14. Executable (Application) Number – a number to identify the application. If not available then -1.
15. Queue Number – a number identifying configured queues. It is suggested to use 0 for interactive jobs.
16. Partition Number – a number identifying configured partitions.
17. Preceding Job Number – a previous Job Number (JobID) which is the job is waiting to finish. With this a dependency between jobs can be established.
18. Think Time from Preceding Job – a value indicating how long a job has to wait after a preceding job has finished before the job is started.

During the evaluation, workloads from the Parallel Workloads Archive were used. Additional, two workloads (ForHLR I+II) were provided by the Karlsruhe Institute of Technology [188][2]. These workloads have been converted into the SWF format and are available online at PWA. The other used workloads are SanDiego Supercomputer Center (SDSC) [214], Swedish Royal Institute of Technology (KTH) [215], and Cornell Theory Center (CTC) [216]. Details about the used workloads can be obtained from Tab. 5.1.

[2]The workload for ForHLR I is not yet available at the parallel workload archive. It is in the process of beeing published.

The workloads from KIT have been modified to fit into the node-level simulation of Alea:
The nodes have multi-core CPUs but due to the usage model of the system, the nodes are always scheduled and allocated as whole nodes, i.e. one node is never shared by two or more jobs and is always exclusively assigned to one job. Therefore, the nodes have been changed from 20 cores per node to one, and divided into the requested number of cores of all jobs accordingly. With this modification, a whole node scheduling is simulated. Also all jobs have been removed belonging to the partition of memory-heavy *fat* nodes, since these are not of interest for the evaluation and both systems (ForHLR I+II) only allow one fat node per job. This results in simulating approximately 249k jobs for ForHLR I and 112k jobs for ForHLR II, respectively.

Figure 5.5 shows the ratio between requested and used wall time of the evaluated workloads. ForHLR II users request about ~3.8 more time than they consume. The smallest difference is for the KTH workload with a factor of ~1.5, CTC with ~2.1 and SDSC with a factor of ~2.6. The highest factor is observed for the ForHLR I workload, where the user requests ~7.5 more than needed. The exact values, in core hours, can be found in the Table 5.2. These values alone indicate that no reliable schedules are possible. Users are requesting much more time than they need.

As a real world example for over-estimated wall times, two users from the ForHLR II workload are depicted. Their submitted jobs are shown in Figure 5.6. User A with 214 jobs has always requested the same 48 cores, but requested different wall time (20 and 24 hours). User B submitted 249 jobs with different core counts (100 and 200) and wall times (12, 15 and 24 hours). On the Y-axis the used wall time is shown. A very clear difference can be seen between both users. Jobs submitted by User A are spread over a wide range, from jobs that ran only for a short time and up to the requested 20 to 24 hours. User B shows a different usage. Although the jobs differ considerably from the time requested by the User B, the run time remains quite similar for different jobs. For example, User B submitted jobs with a requested 24 hours wall time (green), but all these jobs take less than 17 hours. Here a better estimation of the wall time

System	Months	Jobs	Users
CTC	11	77 222	679
KTH	11	28 476	214
SDSC	24	59 715	437
ForHLR I	24	249 000	135
ForHLR II	19	112 000	166

Table 5.1: Number of data points, or jobs in the used historic job workload prom the Parallel Workload Archive and SCC.

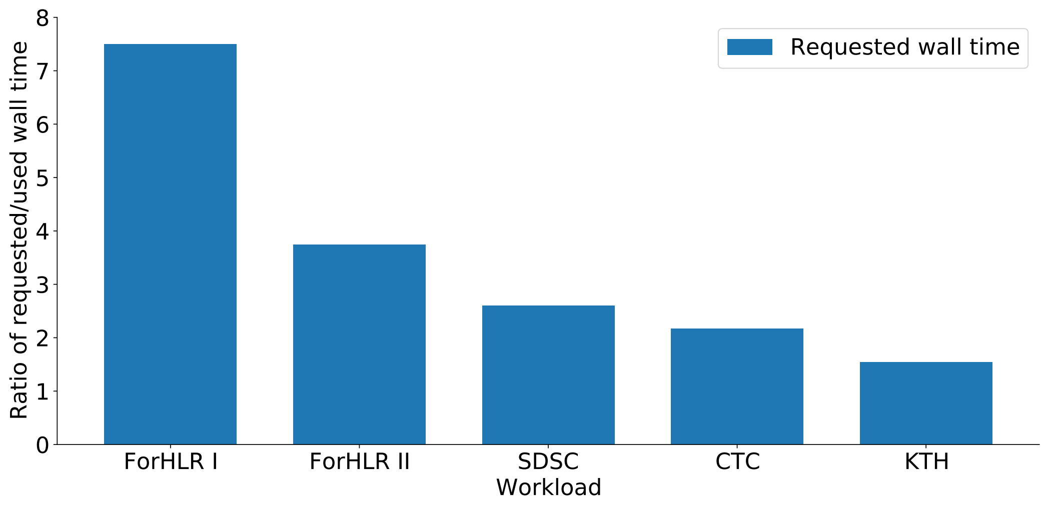


Figure 5.5: Ratio of requested and used wall time compared for different historic workloads..

could be easily done, by looking closely. For User A, such a simple estimation is not possible with the available information.

5.3.1 Additional Metadata

In addition to the available metadata from the PWA, a companion log has been provided for the workloads from KIT. From this additional log, two additional metadata have been examined in detail. One is the jobname and the other is the initial working directory (IWD). It should be mentioned, that this companion log contains privacy-sensitive information which cannot be published online. Both additional metadata entries are freely chosen strings from a user.

The usage of the jobname varies considerably, some users set a specific jobname for each job, others do not use a jobname at all, and some users use a specific jobname for specific parts of their workflow. To make the jobname usable for machine learning, a generic regular expression was used to split the string. The jobname was split by one of the following delimiters "[_|-|\s|.]". The split string is then converted to a matrix. An example how strings are converted is shown in Table 5.3

	ForHLR I	ForHLR II	CTC	SDSC	KTH
Requested time	10 249 993	2 041 322	523 321	288 455	108 073
Used time	1 355 633	555 903	241 900	110 926	70 176

Table 5.2: Total requested wall time by the users and real used wall-time (in hours).

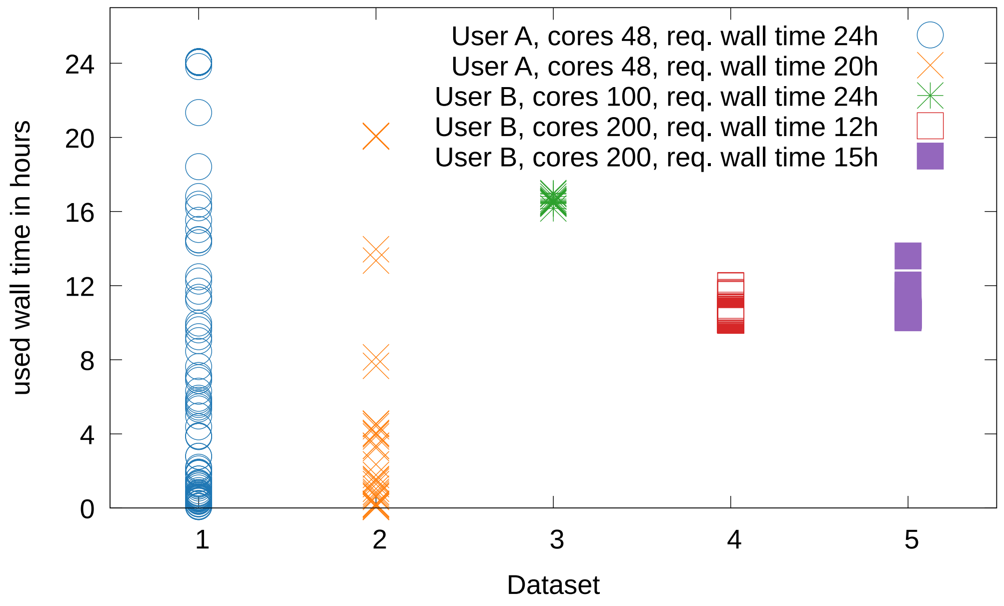


Figure 5.6: Real used wall time categorised by user, request wall time and core count. Category 1 and 2 shows a huge variance in the used wall time, while the other categories has a small variance.

jobname	job	jun	jul	10	16	18
job_jun.16-18	1	1	0	0	1	1
job_jul.10-18	1	0	1	1	0	1

Table 5.3: Example mapping from string to input matrix for two jobnames.

The IWD was split into three parts. The first part points to the parallel file systems. This separates jobs, using the regular home file system or the optimized and faster scratch file system. The second and third parts contain the directory, where the third part is the basename of the working directory. Table 5.4 shows a small example, how directories are converted into a matrix for the machine learning.

Detailed results how much every metadata contributes to improve the wall time prediction are shown in the paper [187] and are not further covered in this thesis. However, detailed results are available in the Appendix A.3.

5.4 Method and Metrics

In order to evaluate the machine learning models, metrics from the corresponding area have been used. These are explained here briefly.

IWD	/p1/joe	/p3/joe	sim	data	run_a
/p1/joe/sim/run_a	1	0	1	0	1
/p3/joe/data/run_a	0	1	0	1	1

Table 5.4: Example mapping from string to input matrix for directory names.

5.4.1 Metrics for Wall Time Prediction

In this section, the metrics are explained that have been used for the machine learning (ML) models. First the models are trained using samples (historical jobs). Here the real run time of a job is given as target value. After a training process, a trained model can be used for predictions. To predict run time of the jobs, a new sample (job) is passed to the model (without real run time) and a predicted target value (predicted run time) is returned. Since the actual used time of the new sample is known, the predicted time from the ML model and the real used time can be compared. With these two values, predicted wall time and real used time, it is possible to measure how good a model is predicting wall times. The built-in metrics from the scikit-learn library [101] are used to evaluate the trained models. Scikit-learn offers several metrics for the regression tasks, such as R^2, mean absolute error (MAE), and median absolute error (MedAE) [217] (cmp. Section 2.4.1).

The R^2 score (coefficient of determination) provides a metric of how well the trained model will predict new samples. It is defined by

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n_{\text{samples}}-1} (y_i - \bar{y})^2}, \tag{5.3}$$

where y_i is the real used walltime and $\hat{y}_i$ is the predicted value of the i-th sample, and

$$\bar{y} = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} y_i, \tag{5.4}$$

where $\bar{y}$ as the average of y_i. The best possible value is 1.0 which corresponds to a perfect prediction. The R^2 score can also be negative and indicates a badly trained model [218]. Other metrics are the mean absolute error (MAE) [219] and the median absolute error (MedAE) [220]. Both measure the difference between predicted and used wall time. MAE is the mean over all pairs of predicted and used wall time,

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|. \tag{5.5}$$

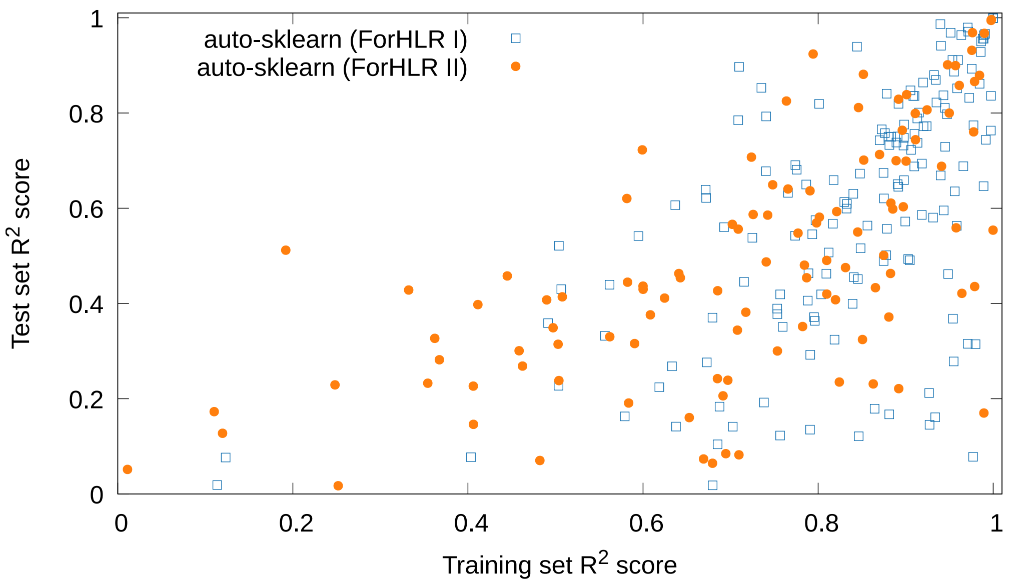


Figure 5.7: The x-axis shows the R^2 score on training samples and the y-axis shows R^2 score on test samples for the work load files from ForHLR I+II. Models trained with automatic Machine learning for 20 min each model.

MedAE is the median value of these pairs,

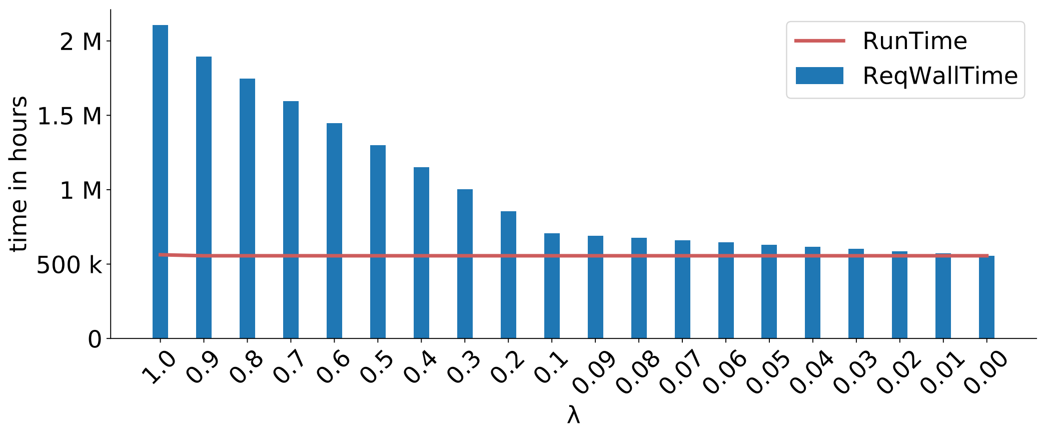
$$\text{MedAE}(y, \hat{y}) = \text{median}(|y_1 - \hat{y}_1|, \ldots, |y_n - \hat{y}_n|). \tag{5.6}$$

In contrast to MAE, MedAE is robust against outliers.

Figure 5.7 shows the results of AutoML trained models with all above mentioned metadata (see Section 5.3.1): requested wall time, task count, initial working dir, jobname, class, start time, and submit time. Each point represents a pair of the R^2 scores, from the training and test set for a specific user of the machines (ForHLR I+II). In Figure 5.7 an accumulation of the pairs in the right upper half can be seen, which indicate that a well-trained model for most of the users has been found.

5.4.2 Resource Allocation Metrics

To relate the wall times predicted by Machine Learning (ML), artificially improved wall times were used. This is used to investigate how scheduling behavior changes, when the requested wall times become more accurate. Especially interesting is, whether resource allocation remains stable and therefore can be predicted longer. During the investigation, job wall time estimates have been subsequently improved. Starting with very inaccurate wall time estimates, as provided by real users of the system, up to completely accurate job run time estimates. These refined wall times were used to investigate in detail the



.

Figure 5.8: ForHLR II workload with improved estimates. Y-axis shows the sum requested and used hours.

scheduler behavior regarding to the improved wall time estimates. For the purpose the "refined" requested wall time $\tilde{T}_{\mathrm{Req}}$ is introduced,

$$\tilde{T}_{\mathrm{Req}} = T_{\mathrm{Run}} + \lambda(T_{\mathrm{Req}} - T_{\mathrm{Run}}) \quad \text{with} \quad \lambda \in [0, 1], \tag{5.7}$$

where T_{Req} is the user-provided wall time and T_{Run} is the job run time. Each job in the workload is then adjusted by the same λ, effectively simulating different precision of provided wall time estimates. For example, if a user requests 10 hours but only needs 1, then for $\lambda = 1.0$ ($\lambda_{1.0}$) would be $\tilde{T}_{\mathrm{Req}} = 10$ hours and for $\lambda = 0.2$ ($\lambda_{0.2}$) the refined wall time $\tilde{T}_{\mathrm{Req}} = 1 + 0.2 \cdot (10 - 1) = 2.8$ hours. Clearly, for $\lambda = 0$ the $\tilde{T}_{\mathrm{Req}}$ becomes a perfect estimate, i.e., $\tilde{T}_{\mathrm{Req}} = T_{\mathrm{Run}}$. Of course, improving wall time estimates this way is rather unrealistic, as shown repeatedly by Tsafrir [221] and Feitelson [222]. It is obvious, that users cannot improve their wall time estimates in such fine-grained ways. But as mentioned before, only impact of accuracy on the node allocation prediction is important, thus such a refinement of wall times is sufficient for the investigation. In Figure 5.8 it is shown how with different lambda values, the requested time (blue bar) and the run time (red line) converge. $\lambda = 1$ reflects the user specified requested wall times and $\lambda = 0$ indicates that requested time and run time are identical.

To measure the accuracy of the node allocation prediction, T_{NAP} as the *Node Allocation Prediction Time* is introduced. This metric defines the time the allocation has not changed until the actual start of the job. In other words, T_{NAP} is the difference between the time the job started and when the last prediction was made. This time is used as a metric to evaluate the results. In this context, T_{NAP} is a *per job* metric, and for example, job parallelism does not affect the way it is calculated. For example, if $T_{\mathrm{NAP}} =$ 1 hour, then the allocated nodes were correctly predicted one hour before job execution was started. In the remainder of this section, results are divided into different groups according to the value of T_{NAP}. T_{NAP} below one second (< 1s), T_{NAP} over one second and below one minute (1–60s), T_{NAP} between 60–600s, and T_{NAP} over ten minutes (long-term predictions). A T_{NAP} below one second, indicates that a job, e.g.,

- started immediately after job submission due to free resources
- was picked up by a back-filling algorithm, or
- was rescheduled on free resources (previously unavailable).

The reason, why ten minutes were selected is based on previous work. It was shown that parallel file systems can be created in a very short time. BeeGFS in less than a minute on 256 nodes (cmp. Section 4.3) and GekkoFS in less than 30 seconds on 512 nodes [223]. In the remaining time, the required data could be copied an ODFS. The 10-minute limit should be interpreted as an arbitrary estimate. A meaningful value can only be determined by considering further factors, e.g., the maximum throughput of the global file system or the performance of the node local storage. In particular, the amount of data that can be copied is limited by the sum of all allocated node-local storage. Of course, this time depends on how much data needs to be copied, but 10 minutes seems to be a reasonable threshold, based on practical experience in a large HPC data center.

These workloads with varying degree of wall time accuracy, were then used as input for a scheduling simulator. The simulator uses the T_{NAP} metric to measure prediction accuracy of each simulation. All experiments have been conducted using the Alea job scheduling simulator [193]. To emulate the system-level job batch scheduler two widely adopted scheduling algorithms were used to create job schedules — a simple First-Come-First-Serve (FCFS) algorithm and the Conservative back-filling algorithm [78].

At the end, several approaches to predict wall times were compared to these *synthetic refinements*. With the artificial improvements, it is possible to make exact statements about how accurate wall time estimates have to be. By comparing the predictions with the previously achieved predictions, a final statement can be made, as to whether the desired advanced data staging is possible without intervention in the scheduling behavior.

5.5 Evaluation

In this section the predictions of the trained machine models and resource allocations are presented. First, it is shown how to create models using automated methods and how the predictions are improved depending on job metadata. Then it is shown which accuracies are needed to predict the computing resources planned by the scheduler and if the accuracies are sufficient to achieve data staging.

5.5.1 Wall Time Prediction using ML

First the accuracy using the R^2 value is shown. Afterwards, a comparison of AutoML trained models with the user estimates are shown using the MAE and MedAE metric.

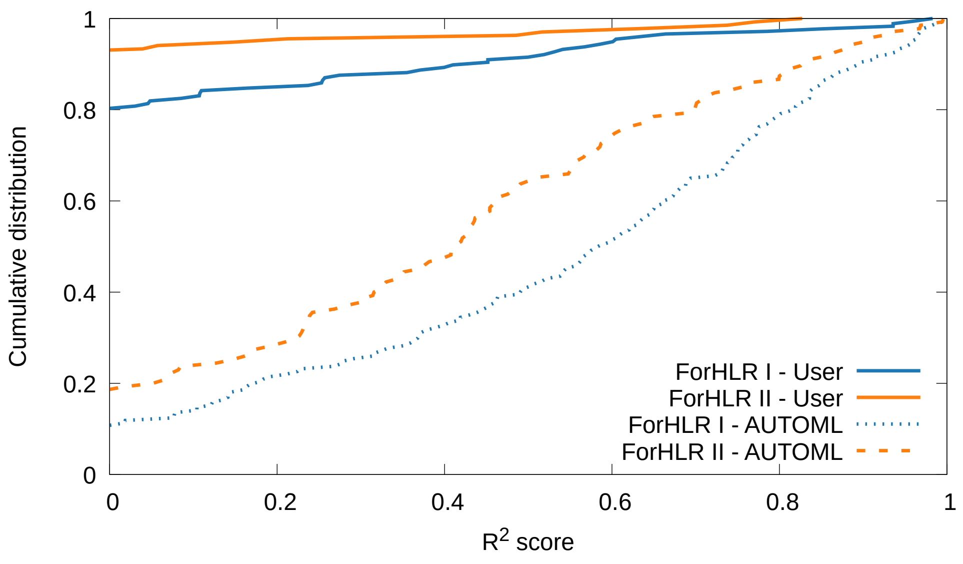


Figure 5.9: X-axis R^2 score on test samples, Y-axis cumulative distribution for ForHLR I+II. Solid line show R^2 for user estimates. Dashed lines show R^2 for AutoML trained models.

Figure 5.9 shows a cumulative distribution plot of the accuracy for trained models compared to user estimates. Dashed lines show R^2 score for the AutoML trained models. To train the models all metadata as described in Section 5.3.1 were used. As a comparison, the R^2 score was calculated by taking the user requested wall time as the user's prediction (solid lines). Ideally, a curve should be flat at the beginning and rise late (high R^2 prediction scores). In contrast, 80 % of users have a negative R^2 score based on user estimated wall times on ForHLR I (purple). Based on user estimated wall times on ForHLR II over 90 % of the users have a negative R^2 score. A more comprehensive table, with detailed results for each metadata is available in the Appendix A.3.

In Figures 5.10 and 5.11, a comparison of the user prediction and AutoML trained models can be seen. Figure 5.10 shows the mean absolute error (MAE) in hours. Figure 5.11 shows the cumulative distribution for the median absolute error (medAE) in hours. In both figures a horizontal line at 0.6 on the Y-Axis represents 60 % of the users. The user estimation of both clusters, has a medAE deviation of about 7.4 hours. A model trained with AutoML, shows for 60 % of the users a medAE of approximately 1 hour on the ForHLR I and 1.4 hours for the ForHLR II. Figure 5.10 shows the difference for 60 % of the users using MAE as a metric. Using AutoML improves the accuracy from 15.4 hours (user estimation) to 4.6 hours (AutoML). The accuracy of the predictions for the ForHLR II improves from almost 16 hours to only 3 hours by AutoML. Taking into account, that only job metadata is used without a knowledge of the payload, this is a good result. More detailed knowledge about the job could result in better predictions, but requires in-depth knowledge about the tools and applications of the users. This cannot be done in an automated way for the vast number of users.

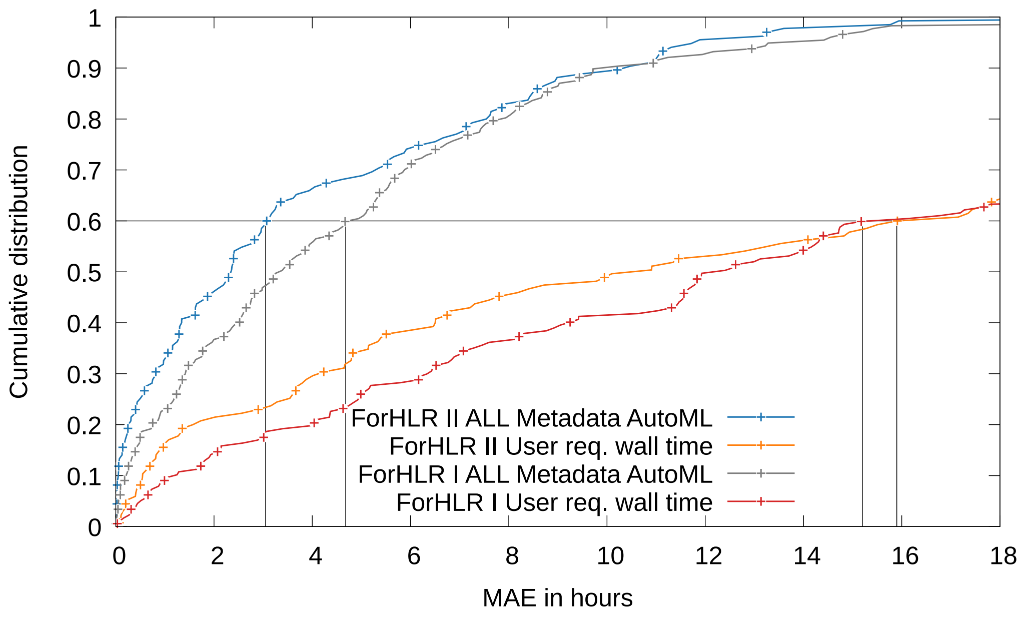


Figure 5.10: Comparison of MAE for ForHLR I+II. X-axis: mean absolute error in hours, Y-axis cumulative distribution.

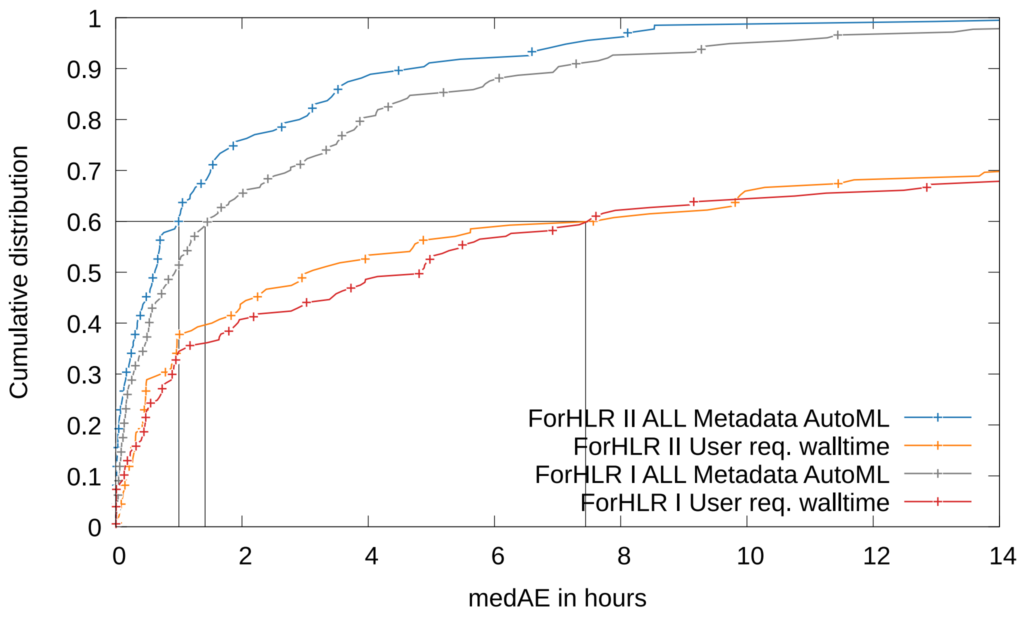


Figure 5.11: Comparison of medAE for ForHLR I+II. X-axis: Median absolute error in hours, Y-axis cumulative distribution.

	User	auto-sklearn	TPOT
ForHLR I	128 181	7 755	5 311
ForHLR II	47 744	10 471	7 362
CTC	13 151	5 572	4 658
KTH	4 817	3 812	3 307
SDSC	9 401	2 493	1 848

Table 5.5: MAE deviation of wall time estimates and predictions in seconds. User given wall time estimates are compared to auto-sklearn and TPOT predictions.

Table 5.5 shows the average MAE deviation for the evaluated workloads. Especially, the improvement for the two systems ForHLR I+II is noticeable. TPOT is able to predict 24 times better than the user for ForHLR I and 6.5 times better for ForHLR II. For the other workloads, the improvement is not as good. SDSC workload has a improvement above 2.8× CTC workload has almost a 4× improvement and KTH has only a 1.45× improvement. It can also be observed, that in this case TPOT shows better results. Both AutoML packages had 12 hour time to find the best model during their training..

5.5.2 Resource Allocation Prediction

The evaluation of node allocation predictions are shown in this section. This allows a statement if the node list is known in time.

The simulation results, with the refined wall times, are shown in Figure 5.12 and 5.13. Detailed results are available in the Appendix (see Tables A.6- A.9). Every workload was simulated using two different scheduling algorithms — FCFS (Figure 5.13) and conservative back-filling (Figure 5.12). The results are categorized within T_{NAP} classes (see Section 5.4.2). Each bar represents a simulation with a different λ value. The last bar, labeled “ALEA”, is a simulation using Alea’s built-in walltime predictor. The bars are categorized into the following groups based on the T_{NAP}:

- Blue: jobs that are started immediately after the job is submitted (instant)
- Orange: jobs with a T_{NAP} between 0 and 1 seconds.
- Green: jobs with a T_{NAP} larger than 1 and up to 60 seconds.
- Red: jobs with T_{NAP} larger than 60 seconds and up to 10 minutes
- Purple: jobs with T_{NAP} larger than 10 minutes (long term predictions)

The long term predictions (purple bars) are in the focus. These predictions increase significant only at very small $\lambda \leq 0.1$, which already proves that very good run time estimates are needed.

Using back-filling, the class of long term predictions (purple) for CTC (Figure 5.12a) workload increases from ~10 % to ~26 %, for $\lambda_{1.0}$ to $\lambda_{0.1}$. From $\lambda_{0.1}$ to $\lambda_{0.01}$ this class improves from ~26 % to ~50 %. Even with perfect wall time predictions $\lambda_{0.0}$, long-term predictions are not possible for half of all jobs. The results for the CTC workload using FCFS algorithm 5.13a show a continuous prediction improvement from $\lambda_{1.0}$ to $\lambda_{0.01}$. On all FCFS results (Figures 5.13a – 5.13e) a large movement, for long term predictions, can be seen when using perfect wall time estimates ($\lambda_{0.0}$). Almost only instantly starting jobs (blue) or with long term predictions (purple) remain when using $\lambda = 0$. Since operators like to keep the utilization of their system high, back-filling is often used, which makes node predictions more difficult. It improves utilization, but during this process it reshuffles node allocations of previously planned job, as was shown in Figure 5.2. This rearrangement then impedes all efforts to create an ODFS on the right set of nodes. From this point of view, a simple FCFS scheduling algorithm is a safer choice.

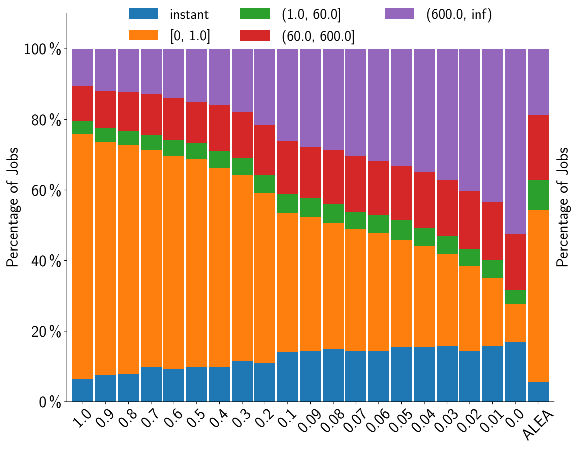
The result when using Alea's run time estimate predictor are varying. For the ForHLR I+II (Figure 5.12d – 5.12e and 5.13d – 5.13e) workloads the built-in predictor shows pretty impressive results. This is caused by the ratio between the requested and used walltime (see Figure 5.5). With such a high ratio between the requested and used run time, the effect of predictions on scheduling and thus on the node prediction is significant. For the remaining workloads, with better user-provided estimates the bar related to Alea's predictions is located, at least, behind the $\lambda_{0.3}$ simulations. Clearly, the simple Alea's predictor does not improve the accuracy very much in such cases.

During the simulations small details were noticed, which should be mentioned. In Figure 5.12 and 5.13 it can be observed that when using the back-filling algorithm with improved λ values, the number of jobs which start immediately (blue) slightly increases. This has two reasons. First, job throughput improves with more accurate wall time estimates as more waiting jobs are started by back-filling. Second, newly submitted jobs may fit better into available free slots. On the other hand, these values hardly change for FCFS algorithm, since it does not use such back-filling optimization. Clearly, jobs that are started instantly cannot profit from any advance data stage-in method.

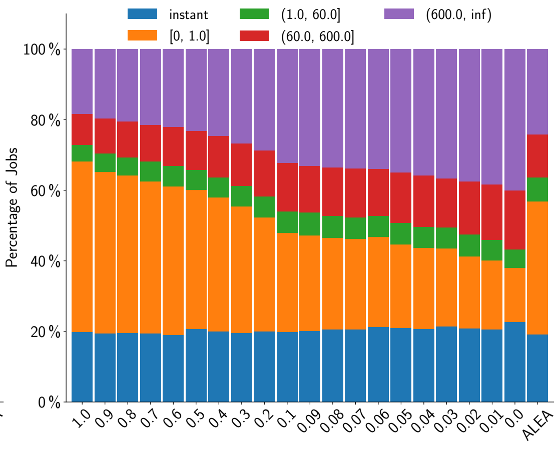
The results in Figure 5.12d – 5.12e and 5.13d – 5.13e show a high rate of jobs that are started immediately after the job submission. There are several reasons for this. First, the workload's time-frame is approximately 1.5 years and various maintenance slots have not been simulated, i.e., the queue is processed normally during maintenance. Second, a part of the nodes has been separated for a limited time for various campaigns whose consumption is not included in the workload.

5.6 Implications for Real Scenarios

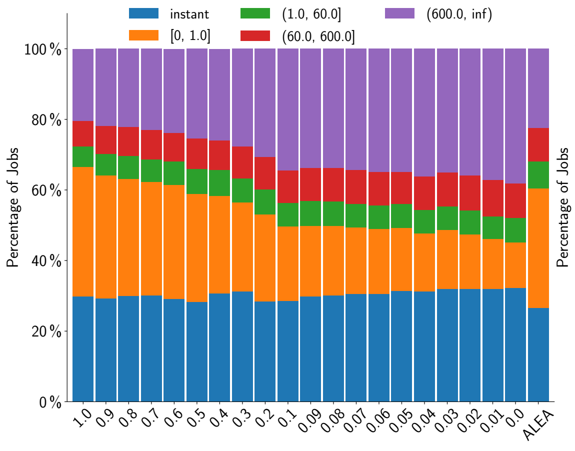
The results above, with artificial wall time refinements, have shown that very accurate predictions are needed.



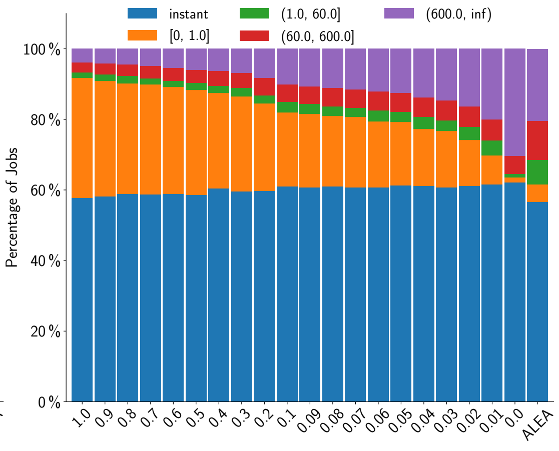
(a) CTC



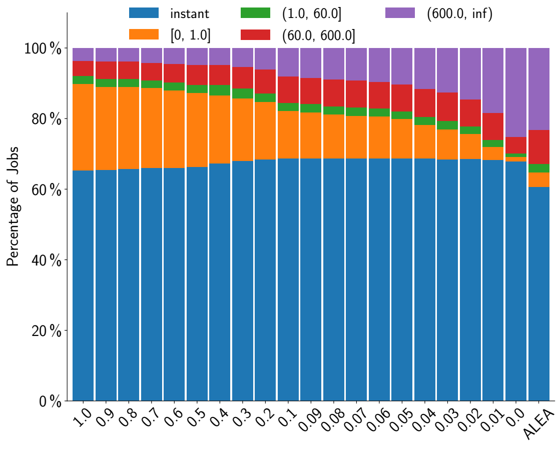
(b) SDSC



(c) KTH

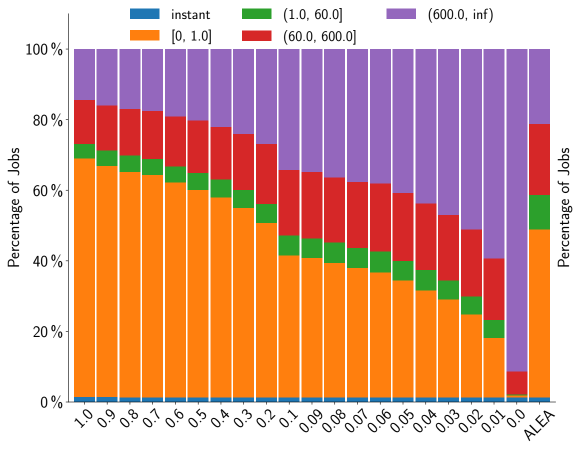


(d) ForHLR I

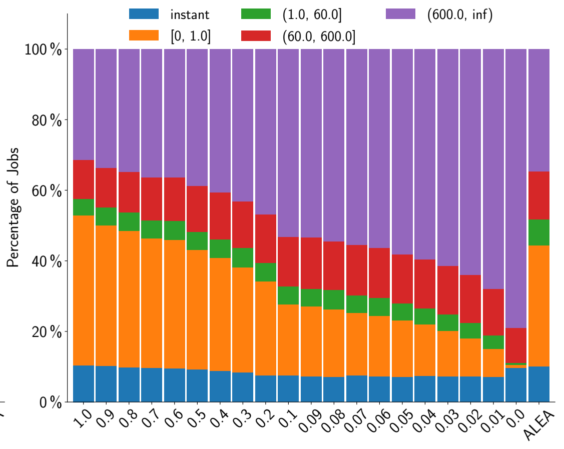


(e) ForHLR II

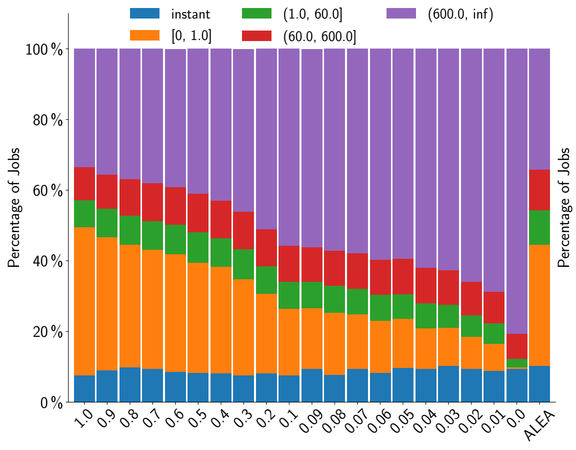
Figure 5.12: Conservative + Back-fill algorithm. Job distributions according to the T_{NAP} class for all used workloads. X-axis: shows different λ values. Blue color denotes instant jobs, orange means job having $T_{\mathsf{NAP}} \leq 1$ s, green denotes jobs with 1 s $< T_{\mathsf{NAP}} \leq 60$ s, red jobs with 60 s$<$ $T_{\mathsf{NAP}} \leq 600$ s, and purple denotes long-term predictions (600 s $< T_{\mathsf{NAP}}$).



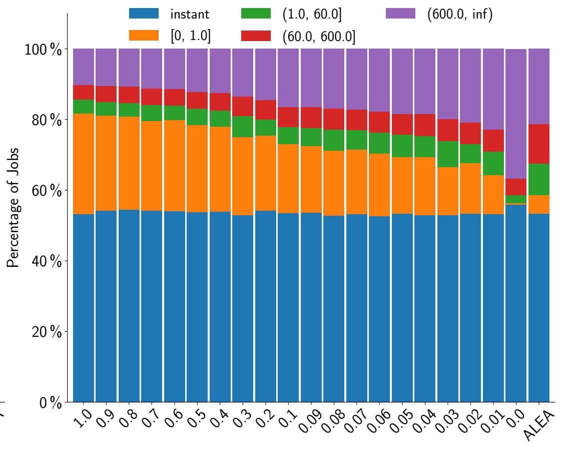
(a) CTC



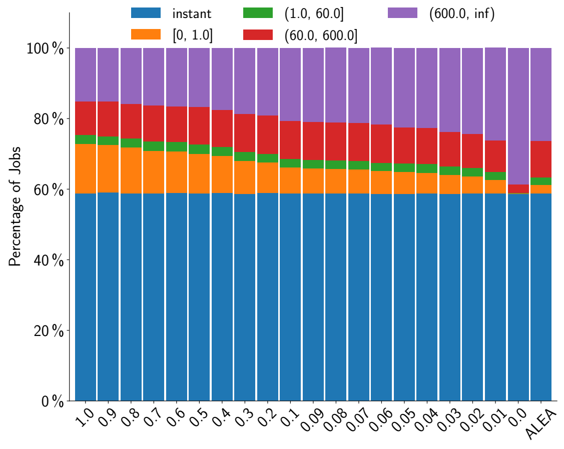
(b) SDSC



(c) KTH



(d) ForHLR I



(e) ForHLR II

Figure 5.13: First Come First Serve (FCFS) algorithm. Job distributions according to the T_{NAP} class for all used workloads. X-axis: shows different λ values. Blue color denotes instant jobs, orange means job having $T_{\text{NAP}} \leq 1$ s, green denotes jobs with $1\,\text{s} < T_{\text{NAP}} \leq 60\,\text{s}$, red jobs with $60\,\text{s} < T_{\text{NAP}} \leq 600\,\text{s}$, and purple denotes long-term predictions ($600\,\text{s} < T_{\text{NAP}}$).

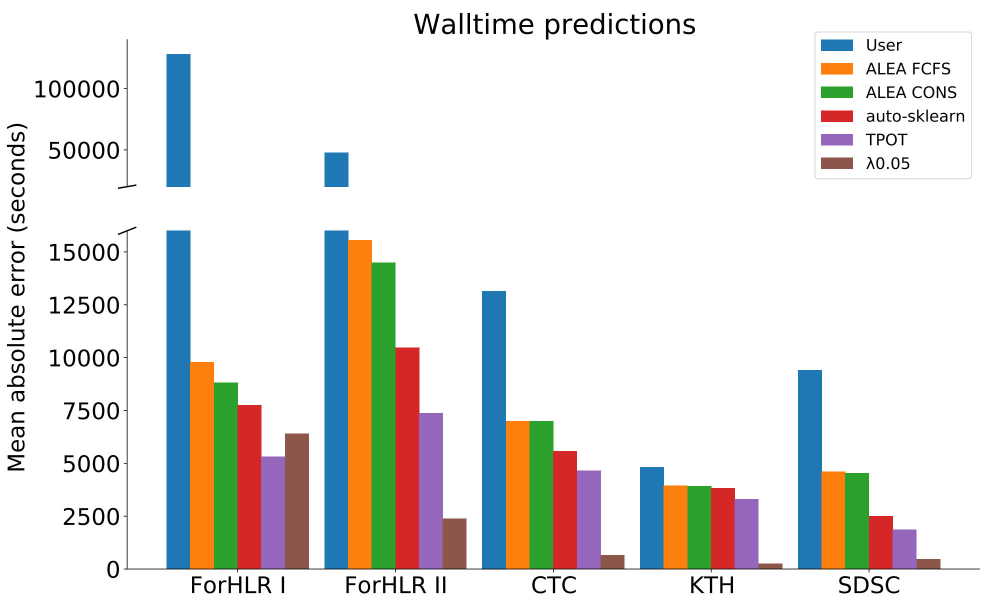


Figure 5.14: The mean absolute error (MAE) for different predictions on different workloads.

Thus, the next important investigation is whether the required accuracy can be achieved, through the use of modern machine learning methods. For this, models with the two AutoML tools were trained for every workload. These AutoML tools were introduced in Section 2.4.1. The models were trained on a 20 core (2× E5-2660) with 64 GiB RAM system. The limitation was a 12 hour training time. The result is shown in Figure 5.14. The mean absolute error (MAE) value is used as metric in the figure. To compare to synthetically refined wall times, the wall times for $\lambda_{0.05}$ are also depicted. With the exception of ForHLR I, the bar for $\lambda_{0.05}$ is far away from the attainable accuracy. The results for ForHLR I are promising at first glance. But it should not forgotten, that users heavily over-estimating the run time of their jobs. Here the TPOT model still shows a mean absolute error of over 5300 seconds, although this is the best model for predicting wall times for ForHLR I. This would mean, that the job start can only be predicted with an average deviation of about 1.5 hours. This makes the approach of creating an ODFS and copying data shortly before the job start practically impossible.

5.7 Summary

In this chapter, it was examined whether an ODFS can be created on the nodes prematurely and whether data can be staged to the nodes in advance. To achieve this, the scheduler must know the exact run times of the jobs. First it was shown how easy and automated improved wall times can be predicted. It was also shown, how well a scheduler can make its plans depending on accurate wall times. Here the recorded jobs were simulated with artificially improved requested wall times. The results have shown, that very good wall times have to be predicted, if not perfect times. Since this is hardly achievable in reality, it makes the approach of copying data to an ODFS in time rather pointless. Here it is necessary to choose a different approach and to interfere with the behavior of the scheduler. Conceptual prototypes have been presented in Section 4.2.

6 Automatic Data Management

"You cannot mandate productivity, you must provide the tools to let people become their best."
— Steve Jobs [224]

The huge increase in scientific data obtained from large-scale simulations or processed during data-intensive computing is raising a problem in data management. Data volumes are constantly growing and must be managed in complex environments by users. Additionally, in the previous chapter, an approach of using an on-demand file system in HPC environments was shown. An on-demand file system adds a temporary new storage layer to the environment. Simply because it is only temporarily, the data must be copied back and forth, which adds more complexity to the workflows. With growing volumes and more data movements, it is important to maintain control.

When managing large amounts of data in HPC environments, it is important to distinguish whether the data is actively used or not. With this information, it is possible to make a decision whether the data should remain on the expensive parallel file system. The primary goal of data management in HPC environments is to support and simplify the familiar processes of data management and to offer additional options to managing data. In addition, existing storage systems and infrastructures must be used efficiently.

In this chapter, a short overview of comparable approaches is given. Afterwards, reasons for data management within HPC environments are shown. Here, the SCC is used as an example, to show the real conditions that exist in HPC centers. The usage of a HPC system is shown, including the usage of the storage systems. The requirements to be met by a data management system are explained. With the shown examples, requirements for a data management tool are collected. Then, a concept for managing data in HPC environments is introduced and demonstrated how the framework facilitates work for operators and users.

This chapter is based on the submitted publication:

- M. Soysal and A. Streit. “JAWS: Just Another Workspace Suite”. In: *International Conference on High Performance Computing and Simulation (HPCS)* (12/2020). Submitted [225].

6.1 Related Work

In the past, there have been many developments and innovations to manage large scale scientific data. Although it is difficult to strictly separate the solutions, three categories can be distinguished: scientific community repositories, storage solutions, and scientific frameworks. All approaches have in common, that these want to manage or organize large amounts of data. Some approaches have a very specific focus and aim at these problems.

Scientific Community Repositories Scientific communities build their own solutions, either as a storage for results or as an interface to data sources. These community solutions are often used across institutes, therefore these repositories are also located outside of actual HPC centers. Such solutions often offer an interface or other possibilities, to provide access to the data. As an example, the Open Source Data Hub System (OS DHuS) is such a scientific repository solution [226]. OS DHuS is a service developed with the purpose to support access to the ESA Copernicus data. The OS DHuS provides two ways of accessing data. An Application Programming Interface (API) and a simple web interface [227]. There are similar scientific repositories for a wide range of domains available [228]. These solutions are tailored for a scientific community. This makes it hard to use such solutions as a generic data management solution. Nevertheless, such scientific community repositories should not be ignored.

Storage Solutions Storage solutions denote approaches, that are integrated into a storage system. These can be implemented nativly in file systems or represent an appliance. However, they have in common the idea to combine different storage technologies, e.g. disks and tapes. With this, either capacity is expanded or top performance is improved. The file system namespace is preserved and therefore, it remains transparent for the user.

An approach for such solutions are so-called Hierarchical Storage Management (HSM) systems. Here, often disk-based solutions are combined with tapes. This allows unused data to be swapped out to tapes. When this data is accessed, the data from the tapes is restored to disk. An example for such a HSM solution is the High Performance Storage System (HPSS). HPSS is a flexible, scalable, and policy-based Hierarchical Storage Management product developed by the HPSS Collaboration [229]. Other HSM appliances are IBM Spectrum Protect (formerly IBM Tivoli Storage Manager) or SAM-QFS [230, 231]. These solutions are very closely tied to the hardware and software of the vendor. Therefore they are not easily adaptable to the conditions in HPC environments. But like the community solution, a HSM itself can be used as data storage.

The above mentioned HSM solutions, aim to increase the usable capacity when combining disks with tape storages. The other direction would be to combine disks with SSDs to increase performance. BeeGFS recommends to use the functionality of their storage pools for such a scenario [166]. SSDs can be linked into a pool. This pool can then be made available to data-intensive applications.

One example of a hybrid solution that supports both directions is Lustre HSM. Lustre provides the capability to have multiple storage tiers within a single file system namespace and can offer capacity with slower tiers as well as peak performance with faster tiers. As storage tier it can use tape based systems, to offer a traditional HSM functionality and move files off the disk file system. If the files are moved successfully, it can be deleted from the disk file system, leaving only a link that references the copy on tape. In addition OSTs can be configured with SSDs, while still having the same namespace. Using the policy engine it can be defined which data should be stored on the SSDs [232, 233].

All these storage solutions are only available within a file system or an appliance. Although some connectors are available, e.g., Lustre HSM can connect to different storage systems like HPSS and TSM [234, 235], they are not suitable for higher-level data management within HPC centers. The features are coupled within the namespace of a file system. This makes it difficult to integrate third-party storage systems such as the scientific repositories.

Scientific Frameworks and Tools To solve the lack of freely available and generic HPC environments, frameworks and tools have been developed to address the management problem. Many of these tools also aggregate individual files and directories into logical units, sometimes called dataset or workspace. Rucio is a project that provides services and associated libraries for allowing scientific collaborations to manage large volumes of data spread across facilities at multiple institutions and organizations. Rucio has been developed by the ATLAS experiment [236]. Rucio is a are very complex tool and need a tight integration, where a very lightweight framework is preferred. Pegasus [237] is a framework for mapping complex scientific workflows onto distributed systems. It uses catalogs to map files to their physical locations but does not allow any advanced data placement. Pwrake [238] and ADIOS [169] are used for efficiently transferring data over wide area networks.

Data Jockey [239] is a tool, which aggregates files and directories to a logic unit, so called datasets. Data Jockey was developed to automate the task of bulk data movement and placements of scientific workflows within HPC centers. A disadvantage of Data Jockey is that it lacks policies on how long data should be stored somewhere or a role based administration.

The HPC-Workspace tools [240] also use the basic idea of workspaces, but have no data movement implemented. They offer a lifetime for workspaces, which helps to prevent that users do not store their data forever on the storage. While HPC-Workspace offers very limited metadata it is easily usable in HPC environments[1]. For efficiently using the infrastructure a queue-based data movement is missing, also it does not offer roles.

6.2 Reasons for Data Management within HPC Environments

The reasons why an advanced data management tool is needed within HPC environments, is explained in the following. The storage solutions, that have been discussed in the related work before, are not supposed to be replaced. It is desired to use the existing storage solutions efficiently and to be able to make new storage types available, in a simple manner. Also the already mentioned scientific community repositories should be accessible or easily adaptable by a data management tool. From this, the first requirements for such a data management tool can already be derived. On the one hand, already existing storage technologies and solutions should be used efficiently and on the other hand, new storage types must be easily adaptable.

6.2.1 Storage Capacity

Key numbers regarding capacity and price are important to understand why different strategies exist. For this, the development of the capacity of parallel storage systems as well as the prices paid at that time are shown. The numbers are derived from the purchases made at SCC. Due to the mechanism of public tendering, the prices are only a snapshot and should not be understood as world market prices.

Growth of PFS Storage First the growth of PFS capacity is shown in Figure 6.1. Only the PFS storage systems, that were directly purchased for the supercomputers, at SCC, are depicted. The net capacity has grown from 11.4 TiB in the year 2005 up to 5 427 TiB in 2019. During the 5 years from 2015 to 2019, usable PFS capacity increased from 2 118 TiB to 5 427 TiB, more than twice as much. It is expected, that the growth of these storage systems will increase at the same rate in the future.

[1]It should be mentioned that almost all HPC systems in the federal state Baden-Württemberg are using this tool [241].

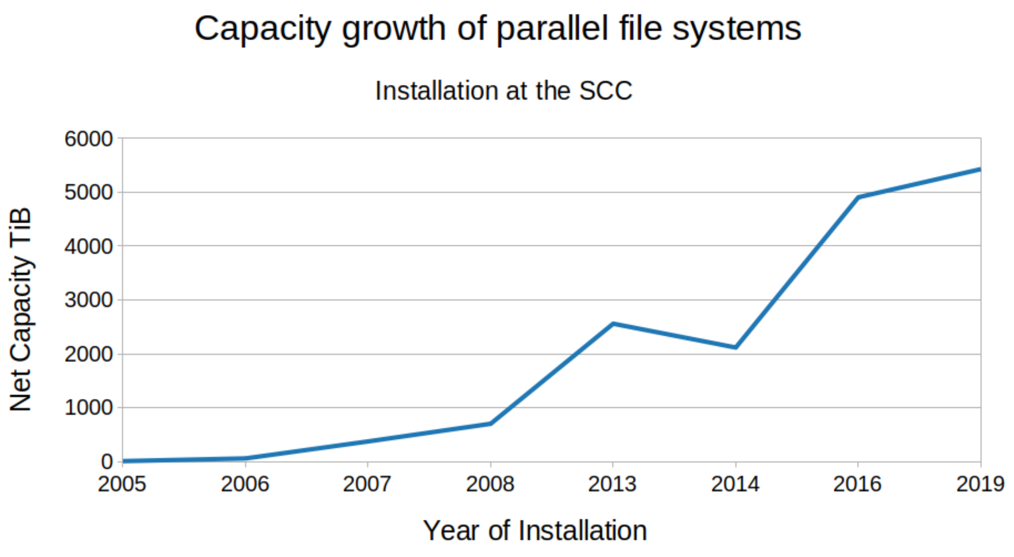


Figure 6.1: Net capacity growth (TiB) of parallel file system for HPC at SCC.

Price per Capacity Figure 6.2(b) shows the price development of procurements at SCC. The two lines show the price / GiB capacity for the purchase of tapes (blue line) and PFS storage (orange line). Before comparing the numbers, some aspects must be considered:

- The prices are converted with the Euro / USD exchange rate at that time
- With PFS procurement, the InfiniBand hardware for the connection to the corresponding cluster is also purchased and included in the price.
- The tape purchases are only the tapes themselves, since the tape library is still in use and is upgraded independently.
- Over the years, tapes have evolved in technology.
- For PFS, only the net capacity that is made available to the user was taken into account.
- For Tape, only uncompressed capacity is considered.

From 2007 up to the year 2014 only LTO Tapes [242] were purchased. In the year 2014 LTO price per GB was 40 times cheaper than the PFS per GB. Afterwards the used technology for tapes switched to Jaguar Tapes [243] with a small price increase. Tapes are still more than 10 times cheaper than PFS storage. Of course the comparison between tape and PFS is not fair. With the given values for PFS the complete installation including server, network, support contracts etc... is included in the price.

Figure 6.2(a) show the price per gigabyte for spinning hard disks. These prices are provided by the company Blackblaze, a large cloud storage provider [244]. According to backblaze, the prices are the average quarterly cost per gigabyte which has been paid [245]. Compared to these values, tape prices are up to 4 times cheaper than hard drives.

Similar conditions can also be found in the commercial sector for hard disks and tapes [122]. Again, the comparison is not quite fair. In this case, only the purchase prices for tapes and disks are quoted, but not the operating costs. The mentioned sums for PFS are complete prices including maintenance and server etc. ... For a fair comparison the TCO should be calculated. Such an calculation was made by Microsoft for their Azure archive [122]. Here a 10-year total cost of operation calculation for 1 PB storage has been made. Hard drives would cost about $5.5 million and for tape-based storage about $810,000 for a 10 year period. So the total cost of ownership for tapes is almost 7 times cheaper.

Considering the prices and the trends in capacity, the data management tool must be able to handle the capacity growth and therefore expected data volumes. Data management should also “encourage” the use of slower (cheaper) storage systems.

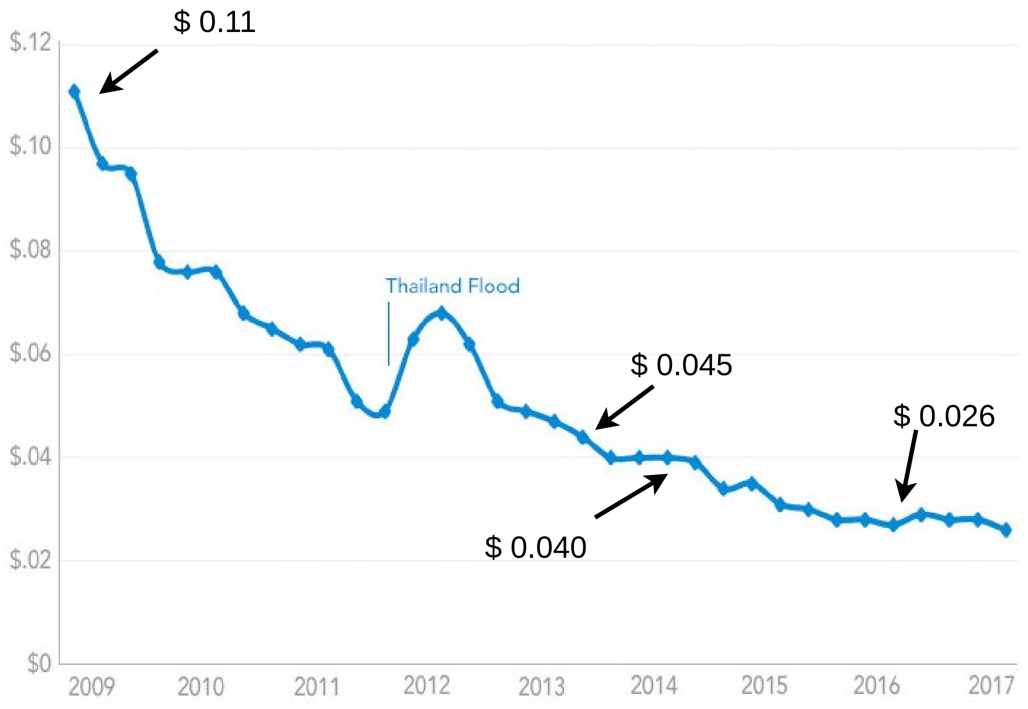
6.2.2 File System Statistics

It is also important, to take a close look at the use of parallel storage systems. The focus here is on the distribution of the file sizes and how much storage space is occupied by corresponding file sizes. Additionally, it is of interest how long the data is kept on the available storage systems.

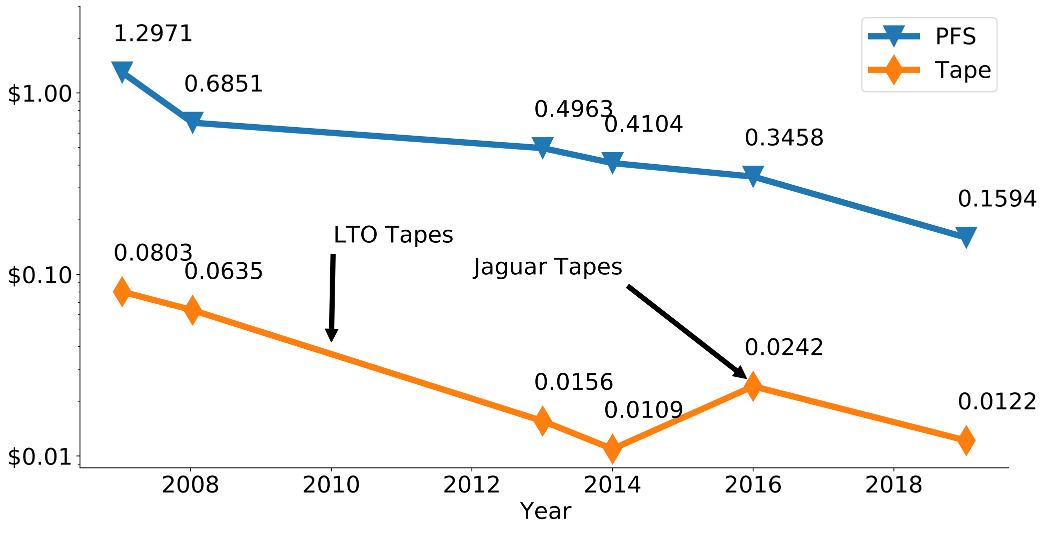
There are a few points that should be taken into account when looking at the presented figures. Only hot storages on the ForHLR II are considered (see Section 3.1.2). The storages have different retention policies and the IMK file system can only be used by users of the respective institute. The Figures 6.3 show the file distribution for all PFSs attached to ForHLR II. Here it can be observed, that most of the files on all PFS are rather small, as shown in Figure 6.3(a). Most files are smaller then 2 MiB, but capacity is mostly occupied by larger files. Table 6.1 shows the relationship between the stored files and the capacity usage, based on the file split of 2 MiB. It is visible that a few large files occupy the capacity but the majority of the files are rather small.

Figure 6.4 shows the ages of the stored files. HOME and IMK file system do not have a retention policy and a vast amount of files older than 64 days. On the IMK file system more than 50 % of the files are older than 512 days, which is not surprising because it was installed before the ForHLR II and is used as a source storage for climate data. The IMK file system was also attached to the predecessor system ForHLR I. Interesting are the values of the two storage systems *scratch* and *workspaces*. On both systems, retention policies are used to prevent data from being stored in storage forever[2]. Especially, the workspace file system is interesting. Although a maximum of 240 days is possible here, hardly any files are older than 64 days. Here users must regularly extend the lifetime of workspaces. If the user does not extend lifetime, the workspaces are deleted by the admins when needed. The reason why there are no files older than 64 days, is due to a

[2]It should be mentioned that the retention policy is only a minimum guarantee. Data is not deleted immediately



(a) Blackbaze average price per GiB paid. Source: `backblaze.com`



(b) Average price perGiB. Source: SCC

Figure 6.2: Price comparison of PFS storage (orange) and Tape storage (blue) per GiB.

file size	HOME	scratch	workspaces	IMK
count				
< 2MB	79.71 %	93.34 %	96.46 %	70.20 %
> 2MB	20.29 %	6.66 %	3.54 %	29.80 %
capcity used				
< 2MB	0.57 %	0.42 %	5.86 %	0.80 %
> 2MB	99.43 %	99.58 %	94.14 %	99.20 %

Table 6.1: Storage usage by file size.

cleanup of that file-system, shortly before the file system statistics where taken. The statistics were created in autumn 2016, after the system was in production for a half year.

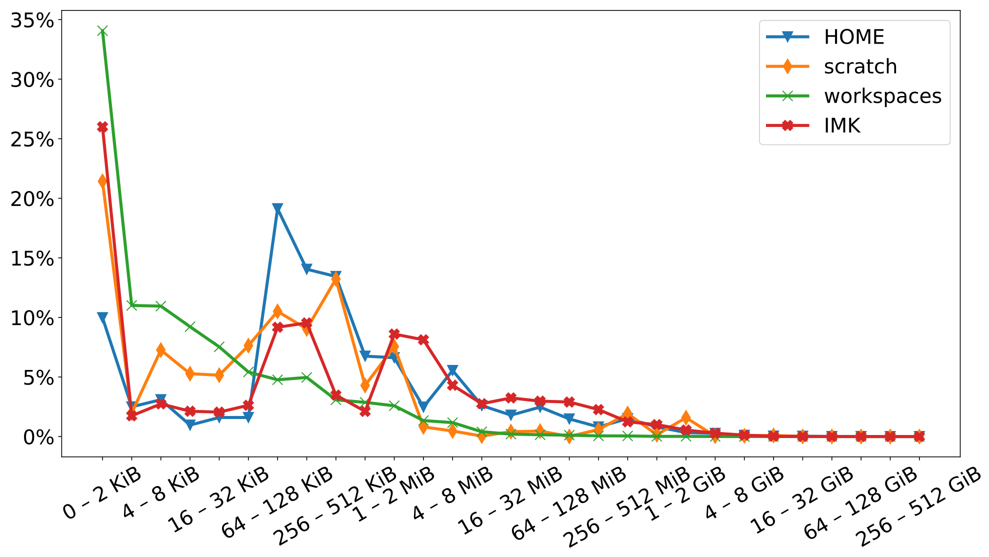
Similar observations can be obtained from other HPC sites [246]. With these file system statistics, additional requirements can be identified for a data management framework. It needs to be able to handle many small files as they make up the largest part of the files and large files as they consume most of the capacity. The data management framework should offer methods to identify unused data or users should indicate somehow, if they still need that data.

6.2.3 Example User Behaviors

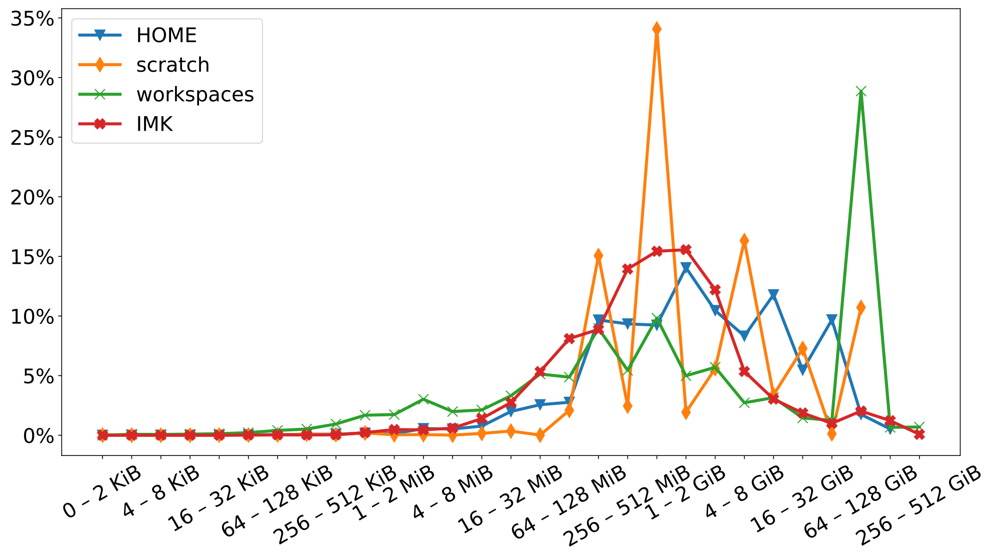
Without detailed key information on the scientific usage of the systems, it is often difficult to identify the problem precisely. It is also not possible to collect and store arbitrary data about users due to data protection guidelines. But with the existing log files and after personal contact with two users, it was possible, to reconstruct their usage on HPC systems. The submitted jobs have been categorized, by the users, into test jobs, pre- and post-processing, and payload jobs where real simulations were executed. The temporary data consumption for both users could be well investigated, for the given time period.

User 1 was a PhD student from Q1/2016 to Q4/2019 and mainly used the ForHLR II. In autumn 2019, he successfully finished his PhD in natural sciences at the KIT Faculty of Physics. The research area for his PhD thesis [247] was from the discipline of atmospheric sciences. His topic was polar stratospheric clouds and ozone in the polar stratosphere, which he simulated and investigated with the ICON-ART software [19].

User 2 has been a PhD student since Q3/2016 and will soon finish his doctorate. His research area for his PhD thesis [248] is from the discipline of turbulent combustion, which he simulates and investigates with OpenFOAM. He uses besides the ForHLR I+II, also other systems outside KIT. The use of the HPC systems outside of the KIT cannot be investigated in detail. However, the user stores the scientific data on archive systems at SCC, including data generated outside the SCC.

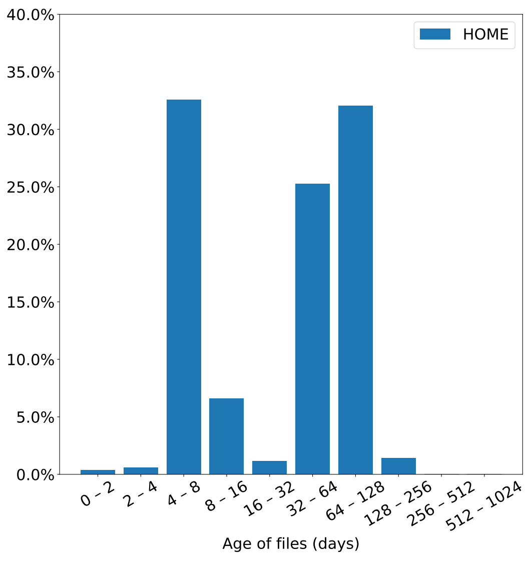


(a) Percentage of files regarding to filesize.

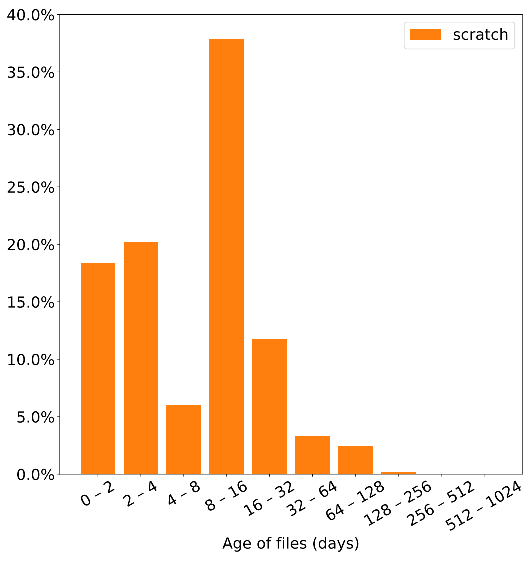


(b) Disk capacity used by filesize.

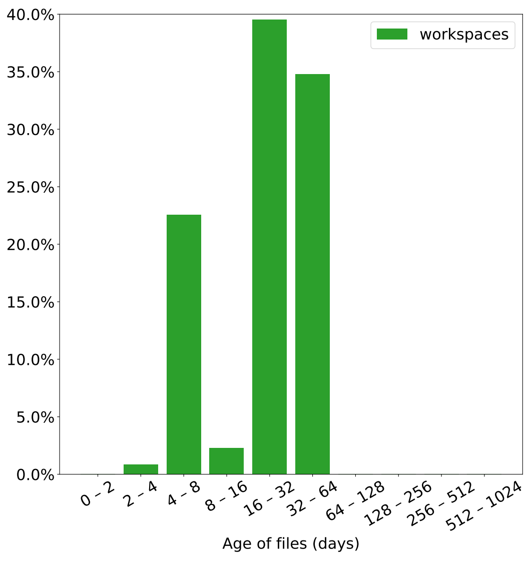
Figure 6.3: Distribution of file sizes and file system capacity used by large files.



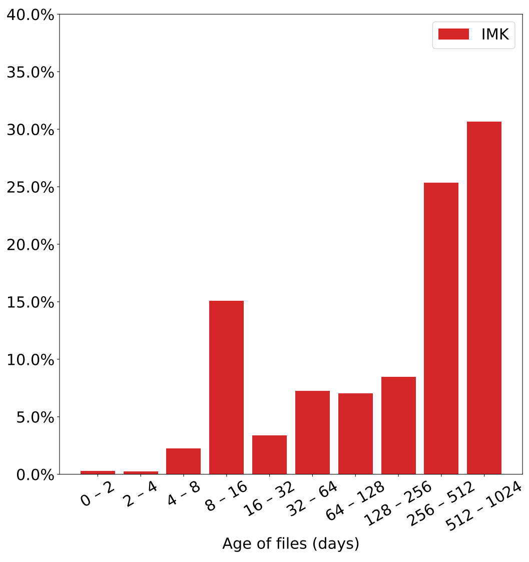
(a) HOME file system.



(b) Scratch file system.



(c) Workspaces file system.



(d) IMK file system.

Figure 6.4: Age of files since creation on different file systems.

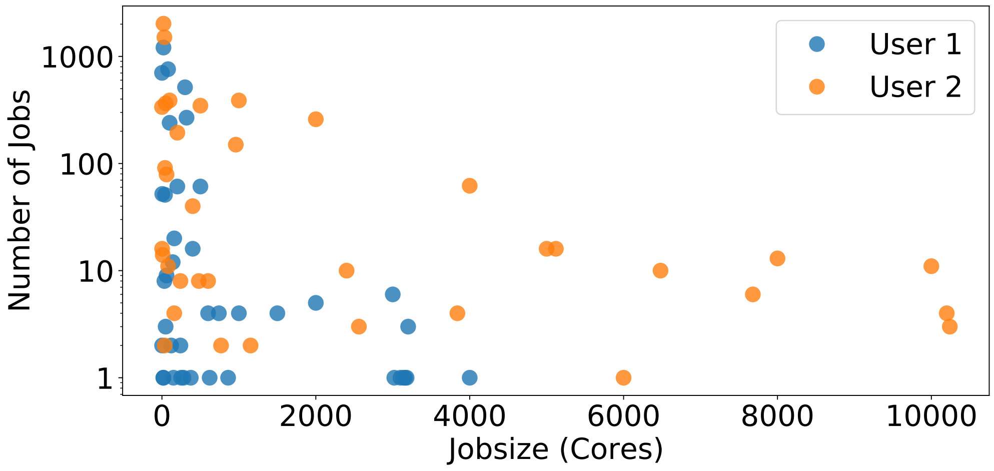


Figure 6.5: Distribution of jobsizes for two scientists based on their HPC usage at SCC. Details can be taken from Table A.1.

Table 6.2 shows the number of jobs and generated amount data. Figure 6.5 shows the distribution by CPU core number for both users. User 2 clearly submitted much larger jobs more often. User 1 submitted a total of 4038 jobs on the ForHLR II, of which 2715 were short jobs that were either used as a test or aborted too early. The rest can be seen as real payload and of these 252 were climate simulations and 1071 were pre- and post-processing. So the scientist needed about four times more jobs to pre- and post-processing, analyses, and copying his data than real simulations. In total User 1 generated about 113 TiB of data in the recorded period (about 1.5 years). His doctorate took approximately twice as long (3 years) and according to his own statements the total amount of data generated will correspond to twice as much, i.e., roughly 225 TiB. Approximately 14.2 TiB are stored in the archive for long-term storage.
User 2 submitted a total of 6402 jobs on the ForHLR II, of which 4044 were short jobs, that were either used as a test or aborted too early. The rest can be considered as real payload (~850) or pre- and post-processing jobs (~1500). Altogether, User 2 has generated in the recorded period (approx. 1.5 years) around 70 TiB of data. According to his own statements the total amount of data generated up today is roughly estimated at 140 TiB. Until the writing of this thesis about 20 TiB are stored in the archive for long-term storage. What is also interesting, is that around 5 million files are stored in the archive.

Both users are from completely different fields of science, climate research and combustion technology, but some conclusions can be drawn from these numbers. Both users submit much more pre- and post-processing, than actual simulation (payload) jobs. After personal contact with the operators of the HPC systems, this ratio is not unusual and applies to the majority of users. Both users are top users of ForHLR II systems and use rather classical HPC applications. The applications are large scale and highly parallel. Both have done code development and therefore submitted many jobs

	User 1	User 2
Total Jobs	4038	6402
Test Jobs	2715	4044
Payload Jobs	1323	2358
↳ Simulation Jobs	252	~850
↳ Pre/Post-processing	1071	~1500
Generated Amount of Data (TiB from 06/2016–01/2018)	~113	~70
Total Amount of Data (TiB)	~226	~140
Archived (TiB)	~14.2	~20
Archived (# Files)	~48K	>5M

Table 6.2: Historical usage of two scientist. User 1 used ForHLR II, active between 06/2016-12/2019. User 2 used ForHLR 1+II, active between 06/2016–10/2020.

for testing. The ratio between test and payload jobs is also not unusual, but the absolute numbers are much lower for other users. Also, only a small part of the total generated data volume remains for archiving, approximately for User 1 $\sim \frac{1}{16}$ and for User 2 it is $\sim \frac{1}{7}$. As mentioned before, both users also developed and tested code. For other users, which only use HPC systems for simulations, higher data volumes are expected.

Taking this small example for two users, additional requirements can be identified for a data management framework. Users submitting a large number of jobs. Test jobs and payload jobs are generating data, even if they are submitted for testing purposes. The large amount of data generated by the simulation needs to be organized. Users need more than just directories to organize their data. Users are submitting jobs for data movement, as data transfer can take some time due to the volume of data. Data movement should be back-grounded, without the need for an interactive session. Users are submitting vast numbers of test jobs, an ease of removing the generated results would be desired. As shown, users do a lot of pre- and post-processing. The data could be pushed to cheaper storage in the meantime and staged back to PFS when needed. A method to simplify this workflow can help reducing the usage on the expensive PFS storages.

6.2.4 Requirements for Data Staging

With the reasons described above for data management in HPC environments, the following requirements are imposed on a data management tool. It is important to use existing storage systems efficiently and new storage solutions should be easily adaptable. The data management tool must be able to handle the capacity growth and therefore expected data volumes. PFS storages are a very expensive storage and users should be "encouraged" to use slower (cheaper) storage systems, such as tape archives. Users are submitting vast number of jobs with pre- and post-processing. Here tools are required to manage the data and, e.g., if data is not immediately post-processed move it to a slower (cheaper) storage. As mentioned above, users are submitting jobs for data movement, because data transfer can take some time due to the data volume. Here a non-interactive data transfer would be preferred. Data management should help to identify unused data. A way where users could indicate, if data is important or not, is desired. Most files are very small, but capacity is used by large files. This ratio of small and large files should not be a problem for the data management framework.

6.3 Prototype Data Management Tool – JAWS

From the previous sections, various requirements for an automated data management tool have now emerged. To meet these requirements the concept of JAWS was developed. JAWS is designed to automate the data management process within HPC environments. The key design aspects are (i) simplify the usage of the different storage systems, (ii) efficient use of infrastructure and storage systems, (iii) a simple way to identify unused data, (iv) a role based system, and (v) an easy integration into HPC environments. To archive this, first some elements of JAWS are introduced.

6.3.1 JAWS Elements

Before the JAWS framework is explained in detail, the basic concepts are introduced here. Configuration options for the introduced elements can be found in Appendix A.3

Workspace The concept of a "workspace" is the central element. Files and folders are linked to a logical unit. A workspace is defined as a set of directories and files. This can be the result of a single simulation or a whole campaign. However, everything that belongs to a workspace depends on the user's workflow, but all files and folders within a workspace can be treated in the same way. The workspaces have a status through their lifetime, e.g., active or expired. Depending on the policy, this status can be extended, but the user has to be active to extend lifetime. If the user does not care about this, this is an indication that the data can be deleted or moved to a slower storage system.

Besides these mandatory metadata (see Appendix A.3), arbitrary entries can be configured. Thus workspaces can be assigned via any metadata. For example, an additional field can be used to specify the funding for this work, e.g. whether it is a DFG or Helmholtz funded project, or a DOI identifier can be used to indicate whether the data belongs to a publication. This option should help the operator to manage appropriate workspaces depending on the general environment and usage scenario.

Target A target describes a storage type, storage location, and properties. In addition, the way the storage is accessed is predefined, so a user does not need to know how to access the storage efficiently. Targets have also configured properties, such as a maximum lifetime for workspaces, and number of allowed extensions of this lifetime. A special property classifies the target as hot or cold. As already explained in Section 2.5, a hot storage indicates a suitable storage for HPC jobs.

Queuing System A simple queuing system is implemented to efficiently use the infrastructure and storage systems [249]. The queuing system consists of a master and distributed workers. So-called data jobs are generated, by the JAWS daemon and submitted to the master. The master acts as scheduler and controls the queue and the workers. It allocates data jobs to free workers and keeps track of the progress. The workers act like a resource manager and execute data jobs. Every worker is configured with targets, which they can serve. Multiple workers can serve the same targets. When starting a worker, it registers itself at the master and indicates which targets it can serve.

Data Job For physical operations on the workspaces, like moving and deleting a workspace, a data job is created and submitted to the above mentioned queuing system.

Roles A role defines the privileges of different user groups. Beside a user and admin role, a group manager role is provided. A group manager is associated with users and has configurable privileges over these users' workspaces.

As mentioned at the beginning of this section, the key design aspects (i-v) can be designed with the few elements presented. The simplified use of storage systems (i) is achieved by abstracting them with so-called (storage) targets. The target configuration also determines the tools and method how data should be stored. With a queue based system and distributed workers the infrastructure can be used efficiently (ii). For example, limits for maximum data jobs can be set, to prevent congestion of networks and storage systems. Whether data is still being used can be easily checked by its lifetime (iii). The workspaces are given a lifetime that a user has to take care of. They can extend it or move the data to other storage systems where a longer lifetime is possible. With the role based system, delegation of work to group managers is possible (iv). This is helpful, if a scientist leaves the institution and someone can take care of the digital legacy, such as a designated person at the institute. JAWS is not introducing a new storage type or systems, it orchestraes data within HPC environments. The workspaces are a logical view on data, the data itself is stored as usual on the storage systems, for example as normal files and directories on a POSIX based storage system (v). JAWS daemons do not need special hardware or resources, they use the existing nodes (e.g., service nodes) and infrastructure.

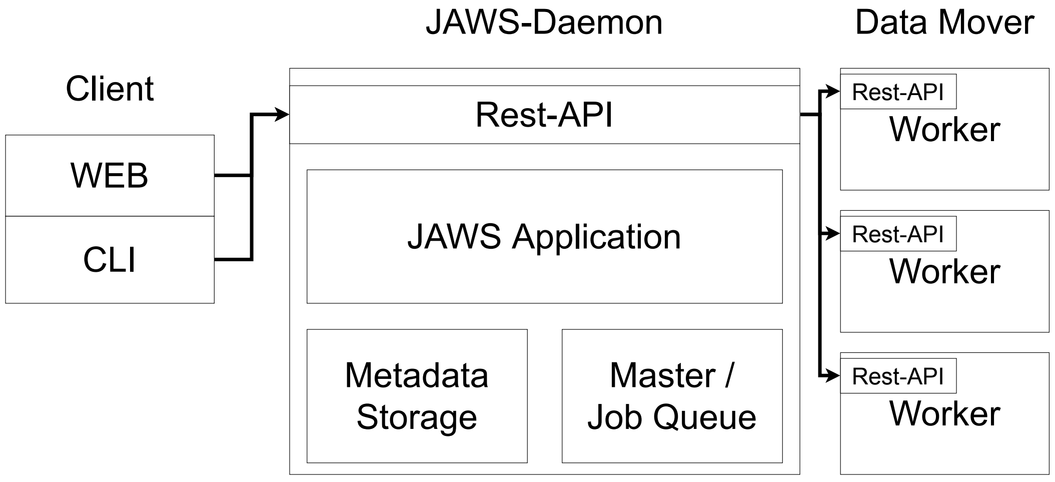


Figure 6.6: Design of the JAWS framework. Clients (Web/CLI) access the main daemon via REST API. JAWS Daemon communicates via REST API with distributed worker.

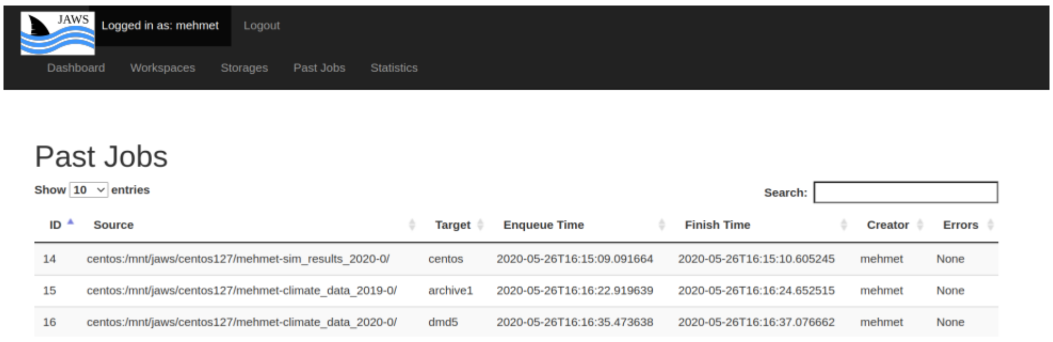
6.3.2 Design and Architecture

JAWS is developed using PYTHON 3 [250] and the Flask framework [251]. Flask is a web micro-framework, which helps developing web-based application with a template engine.

Figure 6.6 illustrates the components of the JAWS framework. These consist of a main daemon, workers for data moving and clients. The main daemon is accessible by a REST API [252] and communicates with the workers using REST. Two different clients, a web-interface and a Command Line Interface (CLI), are provided and both use the same API calls of the JAWS daemon..

JAWS Daemon The main daemon is composed of several components. The JAWS Application components contain the complete logic for managing and controlling the data management. Another important component is the queuing system consisting of the job queue and the corresponding master. The master acts as a scheduler and communicates with the workers. All jobs that make any physical changes on workspaces are placed in the job queue. A central storage is available for storage metadata such as lifetime or aliases of storages. During development a SQLite database as the metadata storage has been used. While using SQLAlchemy [253] as an abstraction layer the database is easily exchangeable with other SQL based databases. Modifications to the metadata of workspaces are done within the main daemon and no data job has to be created.

Workers Workers are responsible for physical actions on workspaces, like moving, compression or deleting. Any number of workers can be deployed. Every worker can be configured for multiple targets. At startup, workers register themselves to the master and wait until a job is allocated. During registration, workers indicate which targets they serve.



Past Jobs

Show 10 entries

Search:

ID	Source	Target	Enqueue Time	Finish Time	Creator	Errors
14	centos:/mnt/jaws/centos127/mehmet-sim_results_2020-0/	centos	2020-05-26T16:15:09.091664	2020-05-26T16:15:10.605245	mehmet	None
15	centos:/mnt/jaws/centos127/mehmet-climate_data_2019-0/	archive1	2020-05-26T16:16:22.919639	2020-05-26T16:16:24.652515	mehmet	None
16	centos:/mnt/jaws/centos127/mehmet-climate_data_2020-0/	dmd5	2020-05-26T16:16:35.473638	2020-05-26T16:16:37.076662	mehmet	None

Figure 6.7: Screenshot of web-frontend shows past data movement jobs.

Authentication JAWS offers different ways to authenticate users with the main daemon, based on the access methods. The command line interface offers two optional methods. (i) A random string is created and stored as a file in the user home and in the JAWS database. This string then can be used as a credential to authenticate the user by appending it as a key to the REST API calls. (ii) A more secured method uses the authentication service MUNGE [254] (MUNGE Uid 'N' Gid Emporium). It is highly scalable and often used in HPC cluster environments, since e.g., schedulers also use MUNGE. When enabling MUNGE, the whole communication between the CLI-clients, the daemon, and the workers are encrypted by MUNGE. Furthermore, additional safety features offered by MUNGE are also used, e.g., replay protection or time-to-live.

To login to the web interface a username and password is needed. After successful authentication a session key is created and stored in the browser. For the user password two optional methods were implemented. (iii) The user password from the system. (iv) The string from (i).

Please note that the solution (i) and (iv) are fallback methods. The generation of the string could be insecure if the JAWS daemon is running on a different host. Therefore, this method should be only used in testing environments. For (iii) the JAWS daemon requires extended privileges. This might be too insecure for some HPC centres, but adding additional methods for authentication for a web-based login are easy to implement.

Client JAWS provides two different clients. The web client is derived from the Flask daemon, which is the base for the JAWS daemon. Therefore, technically speaking, the web-frontend is part of the JAWS daemon. Figure 6.7 is showing a screenshot of the web-interface, showing the last workspace movements. Here users can also view the available storages as well as a listing of the users' workspaces. The statistics section shows collected stats about moved workspaces. As already mentioned both, web and CLI, use the same REST API. Figure 6.8 shows the CLI with all available commands. The explanation for the CLI options are available in Appendix A.3.

```
~$ jaws --help
usage: jaws <command> [<args>]

The JAWS - Command Line Tool

----- Currently available Command -----
move, activate, active, cancel, configuration, deactivate,
queue, job, log, resume, rights, resume, setPriority,
workers, share, shareInfo, workspaces, storages

positional arguments:
command     Subcommand to run

optional arguments:
-h, --help  show this help message and exit
```

Figure 6.8: JAWS CLI-frontend listing the available command.

6.4 JAWS usage on HPC Systems

As described above the REST API that can be accessed via a browser and CLI. Use scenarios of general processes are explained below. The detailed commands and all options can be found in the documentation (`https://github.com/mehsoy/jaws`).

Metadata & ACL As mentioned above there are different meta data for workspaces and storages. According to the requirements, new metadata can be added and users as well admins can be given appropriate rights for it. Figure 6.9 shows such an example configuration. Here an administrator is only allowed to set the specified values for storages. For workspaces there are additional to the above mentioned mandatory entries. A user can set a value for DOI and could indicate, that the workspace content belongs to a publication. Also the workspace contains an entry (student) which only can be set by a admin. This could be used to indicate if the workspace is created by a student and different polices should be applied.

Creation and use of workspaces First the user creates a workspace on a target. Therefore, the user specifies an alias that is unique for him/her. The JAWS daemon then creates a directory for the user on a default target. Based on the target configuration, the workspace is assigned with a lifetime. After the workspace is created, the directory can be used as usual. To make sure the workspace name is unique it is assembled of username, alias, and a counter. A counter is needed when a user chooses multiple times

```
[storages]
# storages are never allowed to be modified by simple users
Administrator=['alias', 'accept_jobs', 'max_extensions']

[workspaces]
User=['DOI']
Administrator=['student']
```

Figure 6.9: Example metadata ACL configuration for storages and workspaces. Only admins are allowed to change the keys for storages. For workspaces different keys are available for user and admin.

```
~$ jaws workspace create mysimdatat-2020
Created new Workspace joe-mysimdatat-2020-0
~$ jaws workspace locate  joe-mysimdatat-2020-0
/mnt/jaws/acratch/joe-mysimdatat-2020-0/
~$ cd /mnt/jaws/home/mehmet-mysimdatat-2020-0/
~$ touch test
```

Figure 6.10: Creation of workspace and using after it.

the same alias. The JAWS daemon can be queried to obtain the absolute path. This absolute path only changes when the workspace is moved to another target.

Archive and move workspaces After simulation and evaluations are completed, access to the workspace is at first no longer necessary. To keep the workspace for future use, it can be moved to storages which offer a longer lifetime. Therefore, the user selects the workspace and chooses the destination target. JAWS now creates a data movement job and appends it to the queue. In the background, the JAWS deamon processes the queue. It assigns the job to a free worker that is connected to both (source and destination) targets. Afterwards the worker checks the data on consistency and then deletes the source. The user can check the state of the data movement job. Statistics on the number of files and data volume are collected.

Lifetime of workspaces A user can list an overview of all workspaces and check lifetimes. Before the end of the lifetime, the user can extend the lifetime to indicate that the data is currently in use. If the user does nothing the workspace will be marked as expired. A user can delete a workspaces at any time if it is not longer needed. The process of workspace deletion is also handled by the workers.

Administrative tasks The admin can display a filtered list of all workspaces. Expired workspaces can be deleted immediately or moved to a background storage. If the workspace is moved to a background storage, users could just restore it by themselves. This is a very common scenario, as users sometimes forget to extend the lifetime in advance. This can quickly free up capacity on expensive storage.

Beside the control of the workspaces there are many other functions available to the admin. For example, if a storage is in maintenance, the relevant targets can be marked as offline. This will block all data jobs that have an offline storage as source or destination target.

Role of Group Manager The group managers are admins for a specific user group. They can view a list of workspaces and delete or move a workspace to another storage. However, the view is restricted to certain users in an associated group. This can be useful to delegate work to the group managers, e.g., to manage specific institutes or projects storages.

Additional metadata Besides the mandatory metadata, such as lifetime, any other metadata can be configured. These are not predefined and can be customized by the HPC center. It is possible to link metadata to role privileges. For example, the faculty affiliation can only be set by admins. Other metadata can be configured to be modifiable by the user or group managers. This way workspaces can be better organized. The additional metadata can be set by a command with key–value pair.

Deployment The deployment is easy to realize in existing environments. The daemon, CLI-clients and workers can be distributed to corresponding login/service nodes. The storage systems in use can also be configured as targets. For this purpose, the storage systems in the configuration should point to different root directories to avoid collision with the previous usage. The web interface can be considered optional. JAWS can be limited to the use of the CLI tools.

Adding new Storages Making a new storage available to jaws is quite simple. First is must be configured as a new target, with the corresponding config file. In addition, the workers who will serve this target must be configured. After that the target is available for the users.

Replace a storage If a storage system needs to be replaced, the JAWS framework makes it relatively seamless. The first thing to do is to make the new storage available to JAWS. Now there are two options. Either users move their workspaces from the old storage system to the new target, or the administrator moves all workspaces for the users. If a user logs in and can not find the data in the usual place, they can locate the workspace using the JAWS CLI.

6.5 Summary

In this chapter, an overview of different forms of data management was given. Different approaches have been shown, which all manage a large amount of data, each with different objectives. Using detailed data from the SCC, it was shown where a data management tool can be helpful and what requirements it has to meet. The data presented includes detailed statistics on storage systems and the work of 2 scientists from different disciplines. Also the historical development was presented, regarding the development of capacity and the prices for procurements. The trend to more and more data volume has been visible for a long time. In order to keep the costs for data management within reasonable limits, a special data management tool is needed, which fits seamlessly into these general conditions. The development of concepts for data management led to the prototype JAWS.

JAWS is a simple framework that provides end users with a simple abstraction layer for managing their data, and provides operators with a practical tool for managing their users' data sets and efficiently exploiting the infrastructure. Large storage will become more heterogeneous and complex. To reduce costs, cheaper back-end storage must also be used in HPC environments. With JAWS, the use of these different types of storage can be abstracted and will help users and operators. The complexity of a storage system can be hidden by a framework like JAWS.

The benefits for users and operators were demonstrated using a framework like JAWS. The operators now have the ability to identify unused records. They do not have to go recursively through directories and check the age of a file as described in Section 3.1.2, admins can now check the lifetime of the workspaces. The operator can transparently make new storage systems available to the users and the users do not need to be trained how to access the storage. Through arbitrary workspace metadata, the user is given a way to organize his data. This metadata can also serve as an indicator of how to handle the data in case of expiration.

7 Conclusion and Outlook

“I see little commercial potential for the Internet for at least 10 years.”
— Bill Gates 1994 [255]

The trend in scientific computing is clearly visible. On the one hand, new data-driven sciences are emerging and on the other hand, traditional communities need more and more data storage. To support data-driven science a very powerful data storage system has to be provided. To satisfy the needs for increasing data volumes much more storage capacity has to be offered. Due to this situation three corresponding topics have been identified. First, the feasibility of an on-demand file system was demonstrated, which can provide peak performance for applications. Second, a seamless integration into the batch system is needed to achieve an advanced data staging to the temporary file systems. Third, a concept needs to be developed to manage and move data in HPC environments to satisfy the need for ever-increasing data volumes.

It was shown that on-demand file systems can be easily integrated into an HPC system. An ODFS can immediately reduce the load on the global file systems. Start and stop times are rather not suitable for very short jobs, but experience shows that large jobs usually have longer wall times. The trend in the HPC world clearly shows that ever faster solid state disks are entering the compute nodes. This should further demonstrate the benefits of on-demand file systems on the compute nodes. Despite the additional processes on the computing nodes, the tested applications ran more stable and with less variance in run times. In addition, the applications are less affected by side effects and generate less load on the global PFS. In the applications tested, it happened that some use cases were slower with the use of an on-demand file system, but the advantages for the overall system outweighed the disadvantages of slower execution time. At the same time, the use of on-demand file systems is quite straightforward and changes to the system configuration are minimal. The advantage of this approach over others is that the application code does not need to be changed. The most important effect that occurs immediately is the reduction of the load on the PFS.

Current batch systems can create an ODFS but cannot do an advanced data staging. For this purpose, the resource allocation behavior of schedulers were investigated. Focus here was to determine the time available to stage data. The work shows that all the chosen job metadata contains information which improves the wall time predictions. In particular, the previously unnoticed metadata provide an additional source of information. Good prediction models can be trained with very simple job metadata without having precise knowledge of the user's work. Further questions are, if the prediction models accurate enough to achieve advanced data staging into a temporary on-demand file system. Detailed simulations using existing HPC workloads were conducted. The influence of (in)accurate wall time estimates on the prediction's reliability were investigated. It was shown that ML and the use of multiple job metadata can improve the prediction. Further investigations have shown that, very accurate wall time estimates are needed to deliver precise node allocation predictions. These are hardly reachable and some alternatives have to be found. Different concepts were developed to accomplish these challenges.

Besides the data management between a PFS and an ODFS, the data orchestration within the whole storage pyramid is important. The development of a data management system is triggered by the trend towards data-intensive computing and ever-increasing data volumes in HPC environments. In addition there are more and more heterogeneous storage systems and a confusing jungle of newly installed storage systems, tools, protocols, policies and best practice guides that overwhelm the user. For this purpose, the behavior of users was examined and a detailed look at the file system usage was made. Thus the requirements for a data management tool could be determined. From this requirement the concept for the lightweight framework JAWS was developed. JAWS empowers the users to manage their scientific data across various storage architectures within a HPC center. It transparently provides access to these storages, reducing the need to get familiar with the heterogeneous storage types. Large scale storages are going to get more heterogeneous and more complex. This tool will help users and operators to use these different storages by abstracting the usage of these different storages. The complexity of a storage system can be hidden by frameworks like JAWS.

With the designed concepts for providing a very powerful on-demand file system and for data management of large-volume data, this dissertation contributes to efficient resource and data management within large HPC centers. These concepts are kept open to technology and are therefore generally applicable. New technological developments can be used with minimal effort.

7.1 Outlook

The subject of wall-time predictions is an area that is constantly evolving with new methods and approaches. Collaboration have developed during this thesis in which we continue to pursue the field of scheduler predictions. A repository is in the process of being created, which collects different methods for run time predictions. This will allow to compare new methods in the future. `https://github.com/mehsoy/walltime-prediction-tools`.

Also during this thesis the idea arose to do research and development of a scheduling simulator which can simulate interruptions, such as node failures, software errors or any events that could occur on real systems. A simple scheduling simulator is already developed and is able to simulate node failures and reservations. A first version is planned to be published soon.

The functionality of an on-demand file system is already in general productive use. Depending on the application, the positive influence is enormous and facilitates the operation of the system. Also users have to worry lessa^{ms}bout I/O and can continue with scientific work. As an example, climate researchers have already started using ODFS in their scientific workflow and gain faster executions times for their data-intensive workflows [256]. In the future, new hardware will provide further advantages, which can be evaluated in appropriate cases. An investigation of new applications allows a better adaptation of the on-demand file system and should be done for new applications.

At last the JAWS data management framework should be put into production operation. During this work a prototype was developed. This prototype should be designed to be robust enough to be deployed accordingly. For the prototype a simple queue manager is implemented. This queue manager is only equipped with the most necessary functions. For the future it is planned to use an established HPC batch system which is usually used within supercomputers. This would reduce complexity and would be a more convenient way to use existing software.

A

Appendix

A.1 Metadata of Swarm-based Meta-heuristic Job

Figure A.1 show the metadata operation created by a swarm-based meta-heuristic search. Figure show the execution of two generations. Each generation is causing more than 30 million metadata operations. Every generation is creating a vast number of metadata operation at the beginning and then following by a computation phase.

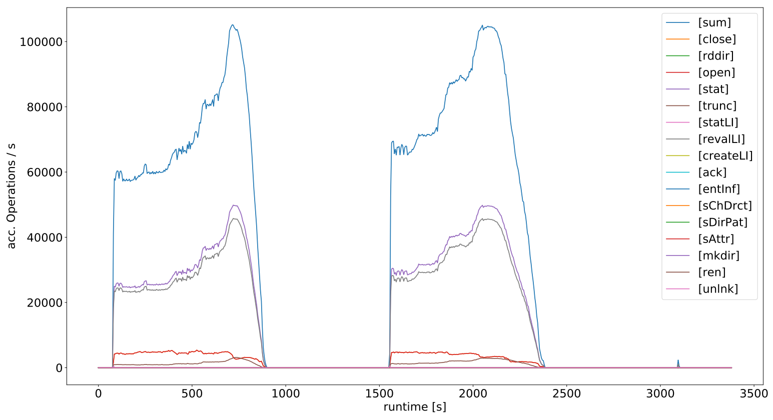


Figure A.1: IOPS over time for two generation. (Swarm-based meta-heuristic)

A.2 Scheduling Simulator (BOB)

BOB (Bob Offers Benchmarks) is a new scheduling simulator, started as a student project at SCC in 2019 [257]. It was developed by a student team during the Lecture “Practice of Software Engineering” (PSE). The aim was to create a rudimentary scheduler, as a basis for further research and development. As initial algorithms it offers a simple First Come First Serve (FCFS) and FCFS with back-filling on configurable HPC systems. The main advantage over other comparable simulators is the information-rich and comprehensive visualization of the simulation results. Is uses modern tools and frameworks such as Python, SimPy, Flask, VueJS. As a bachelor thesis the Scheduling Simulator was enhanced with the feature of simulating node failures and reservations. With this the effects of failures and reservations on the scheduling in HPC systems can be evaluated [258]. Figures A.2 A.3 A.4 show screenshot of the scheduling simulator.

A.3 JAWS

Here are examples for JAWS options and configurations.

A workspace is a logical instance and has the following mandatory metadata.

- alias: a meaningful alias
- username: owner of this workspace
- mountpoint: absolute location
- storage: the target location of the workspace
- max_extensions: how many extensions are allowed
- times_extended: integer value indicating how many time this workspace has been extended on this storage target
- time created: Time when this workspace has been created
- expiration date: when this workspace is going to expire
- max_extension_period: integer value of maximum allowed days to extend the workspace
- status: Possible values are “expired” or “active”

Following configuration options are available:

- alias: a string to use as meaningful name, e.g. “long-term archive” or “scratch space”
- mountpoint: point the location of the target
- is_archive: boolean value to indicate if data has to be packed and compressed (can be configured)
- type: describes the storage type, at the moment only POSIX and tar is possible
- description: a longer string to explain the purpose of this storage

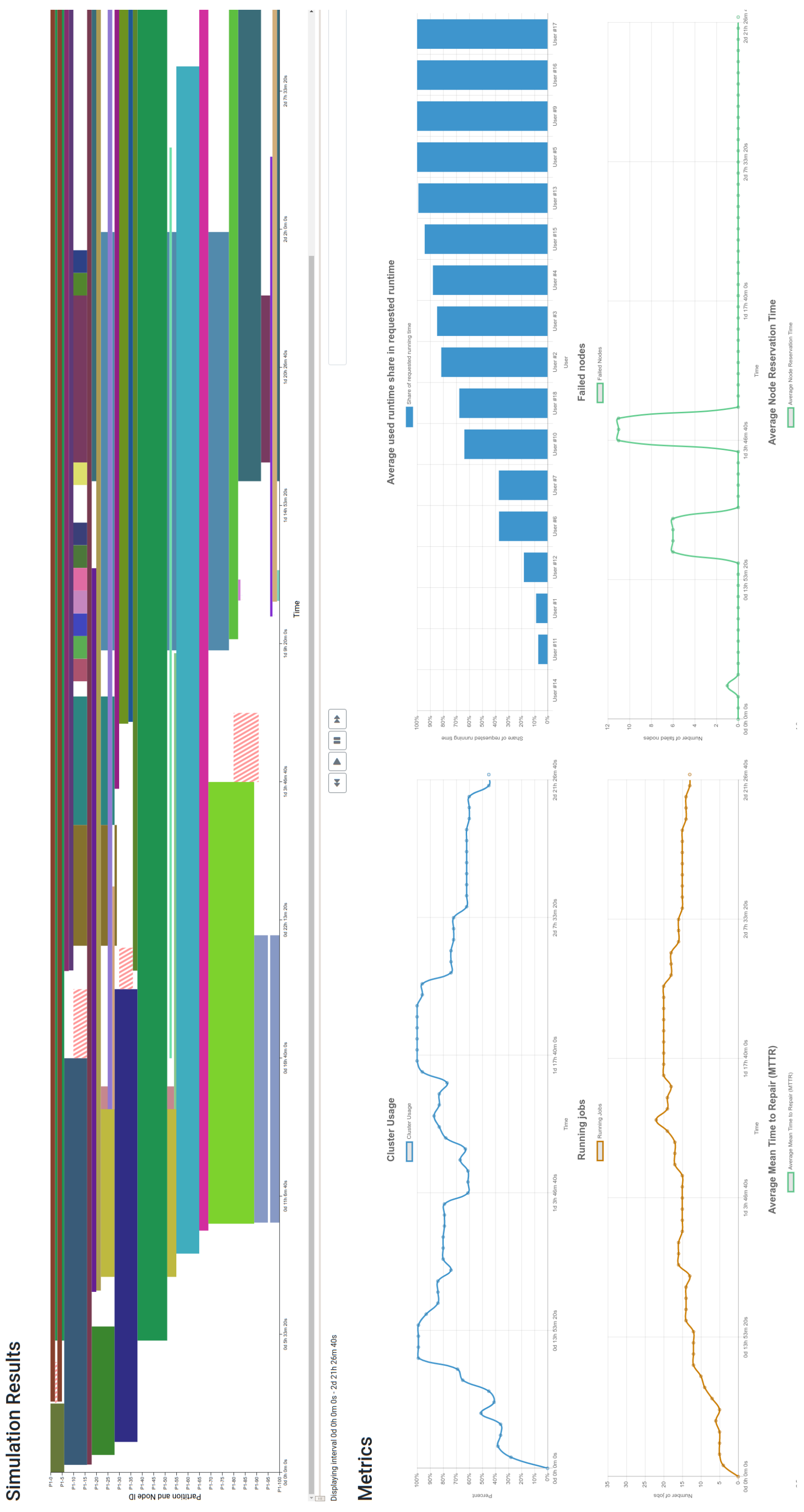
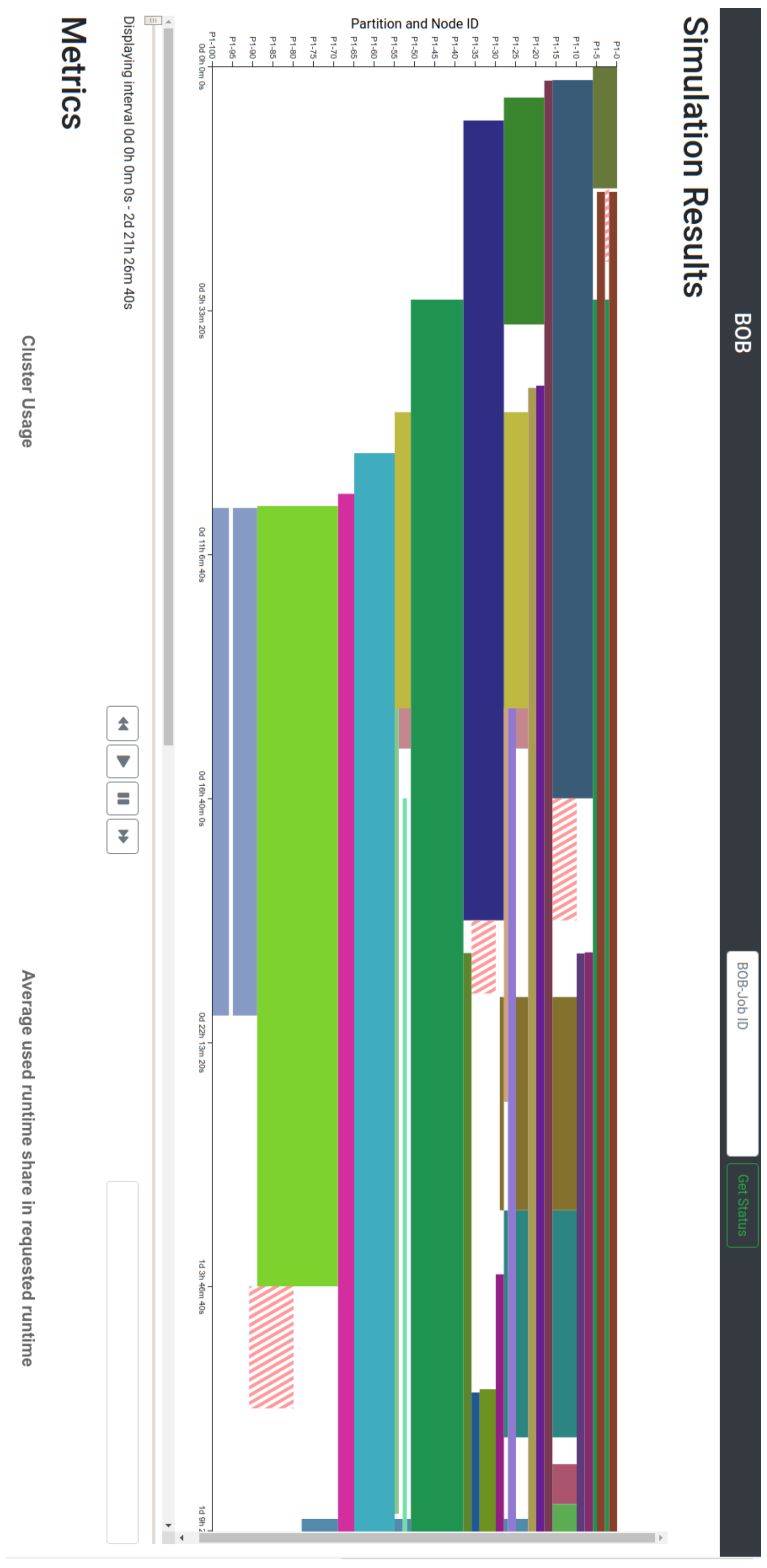
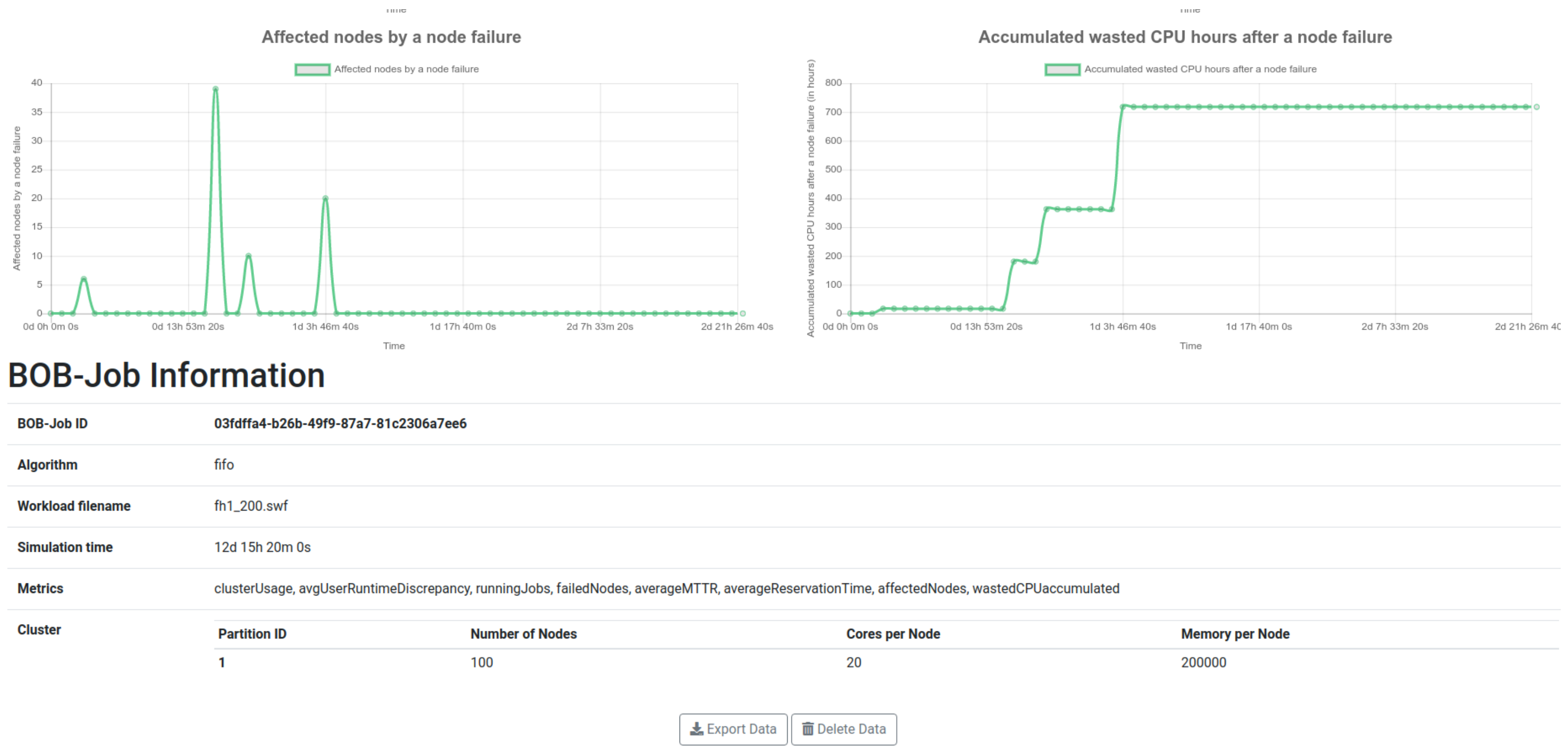

Simulation Results
Partition and Node ID
Time
Displaying interval 0d 0h 0m 0s - 2d 21h 26m 40s
Metrics
Cluster Usage
Percent
Running jobs
Number of jobs
Average used runtime share in requested runtime
Share of requested running time
User
Failed nodes
Number of failed nodes
Average Mean Time to Repair (MTTR)
Average Node Reservation Time

Figure A.2: BOB Showing job “tetris” and life metrics.



Displaying interval 0d 0h 0m 0s - 2d 21h 26m 40s

Figure A.3: BOB screenshot showing the job “tetris”. Red hatching are node reservations.


Affected nodes by a node failure
Affected nodes by a node failure
Affected nodes by a node failure
40
35
30
25
20
15
10
5
0
0d 0h 0m 0s
0d 13h 53m 20s
1d 3h 46m 40s
1d 17h 40m 0s
2d 7h 33m 20s
2d 21h 26m 40s
Time
Accumulated wasted CPU hours after a node failure
Accumulated wasted CPU hours after a node failure
Accumulated wasted CPU hours after a node failure (in hours)
800
700
600
500
400
300
200
100
0
Export Data
Delete Data

BOB-Job Information

BOB-Job ID	**03fdffa4-b26b-49f9-87a7-81c2306a7ee6**
Algorithm	fifo
Workload filename	fh1_200.swf
Simulation time	12d 15h 20m 0s
Metrics	clusterUsage, avgUserRuntimeDiscrepancy, runningJobs, failedNodes, averageMTTR, averageReservationTime, affectedNodes, wastedCPUaccumulated
Cluster	

Partition ID	Number of Nodes	Cores per Node	Memory per Node
1	100	20	200000

Figure A.4: BOB Screenshot showing life metrics and option to export simulated data.

- max_extensions: integer value of maximum allowed lifetime extension for workspaces located on this storage
- max_extension_period: integer value of maximum allowed days to extend the workspace
- accept_jobs: bool value if the storage targets are available. If set to 0 no data jobs executed, that require this storage.
- is_hot: bool value to indicate if this value storage is a hot storage.

Workers have following options for configuration:

- targets: specifies the targets which the worker serves
- token: A token to authenticate with the master
- copytools: configuration which binary to use for specific transfer method.

The options for the JAWS CLI are:

- move: moving a workspace to new target
- activate: command to activate a paused Worker/Master (admin only)
- active: shows active jobs
- cancel: cancels a job
- configuration: shows the configuration of daemon
- deactivate: Administrative command to disable the data job queue or a worker (admin only)
- queue: show the data job queue
- job: get details of a job
- log: shows logs
- resume: pause a job
- rights: set roles (admin only)
- setPriority: set priority for a data job (admin only)
- workers: show workers and status (admin only)
- share: Command to set access rights for workspaces to other users (only supported on POSIX)
- shareInfo: Displayed information on access rights for workspaces
- workspaces: command for workspace
  - create: create a new workspace
  - locate: locate a given workspace
  - list: list all workspaces
  - extend: extend lifetime
  - remove: remove workspace
  - set: set metadata with key=value pair
- storages:
  - list: show all available storages
  - set: set metadata for storage with key=value pair (admin only)

When a data job is created it has the following metadata:

- user_name: who initiated this data job
- action: what physical action should be done, e.g., move or delete
- workspace_identifier: which workspace
- target: optional if workspace has to be moved.

A.4 Tables

Cores	#Jobs User 1	#Jobs User 2	Cores	#Jobs User 1	#Jobs User 2
1	702	337	620	1	0
2	2	16	740	4	0
4	52	0	768	0	2
10	0	14	860	1	0
18	1	0	960	0	150
19	1	0	1000	4	388
20	1210	2021	1152	0	2
32	8	1510	1500	4	0
36	0	2	2000	5	259
40	51	91	2400	0	10
48	3	364	2560	0	3
60	9	79	3000	6	0
80	760	11	3020	1	0
100	240	388	3100	1	0
120	2	0	3140	1	0
140	12	0	3160	1	0
150	1	0	3180	1	0
160	20	4	3200	3	0
200	61	194	3840	0	4
240	2	8	4000	1	62
250	1	0	5000	0	16
280	1	0	5120	0	16
300	515	0	6000	0	1
320	268	0	6480	0	10
375	1	0	7680	0	6
400	16	40	8000	0	13
480	0	8	10000	0	11
500	61	347	10200	0	4
600	4	8	10240	0	3

Table A.1: Job distribution for both users based on core count.

Nodes	Directory creation	Directory stat	Directory removal	File creation	File stat	File read	File removal	Tree creation	Tree removal
4	6208	104989	6944	10384	69001	23542	17310	644	42
8	7649	129858	7496	19186	101042	36261	24587	344	20
16	7620	123337	7147	26761	113917	38302	29988	144	10
32	7447	115644	7363	27208	114139	38056	31813	51	5
64	7217	115369	7394	26799	103164	37305	31957	54	2
128	6702	113868	7082	26464	105713	34888	31401	25	1

Table A.2: ODFS metadata benchmark performed with mdtest (mdtest -d <odfs> -u -n 3600 -i 3)

	ForHLR I				ForHLR II			
MAE range	30min	30min-3h	3h-6h	6h-∞	30min	30min-3h	3h-6h	6h-∞
req. Walltime	22	42	29	84	28	40	25	42
+IWD	28	44	34	71	32	47	14	42
+StartTime	29	49	41	58	30	45	25	35
+SubmitTime	32	47	43	55	31	42	25	37
+TaskCount	28	39	29	81	32	42	18	43
+Jobname	24	45	33	75	30	40	21	44
+Class	26	41	32	78	32	39	19	45
ALL	31	52	42	52	33	46	21	35
User	8	22	20	127	10	21	21	83

Table A.3: Number of users categorized in mean absolute error (MAE) values for the ForHLR I+II.

Nodes	Directory creation	Directory stat	Directory removal	File creation	File stat	File read	File removal	Tree creation	Tree removal
HOME	13588	364989	73268	101641	183297	160128	50086	347	84
WORK	12591	352069	78649	73432	248275	163157	132318	305	78
Workspaces	31715	332744	88935	82120	243150	137580	103584	202	58

Table A.4: Metadata benchmark during the acceptance phase with mdtest (mdtest -d <dir> -i -n 3600 -i -p 10)

	SDSC							
	Conservative + Backfill				FCFS			
λ	instant	[0, 1.0]	(1.0, 600.0)	(600.0, ∞]	instant	[0, 1.0]	(1.0, 600.0]	(600.0, ∞)
1.0	19.91	48.21	13.48	18.40	10.34	42.52	15.75	31.39
0.9	19.43	45.76	15.18	19.64	10.26	39.79	16.36	33.60
0.8	19.63	44.55	15.34	20.48	9.90	38.57	16.81	34.72
0.7	19.51	42.96	16.02	21.51	9.72	36.62	17.38	36.29
0.6	19.01	42.05	16.92	22.02	9.56	36.40	17.67	36.37
0.5	20.69	39.44	16.66	23.21	9.30	33.78	18.24	38.68
0.4	19.96	38.01	17.40	24.63	8.82	31.95	18.69	40.54
0.3	19.61	35.82	17.88	26.69	8.41	29.65	18.79	43.14
0.2	20.04	32.26	18.95	28.75	7.59	26.51	19.15	46.75
0.1	19.92	28.05	19.77	32.26	7.55	20.18	19.04	53.23
0.09	20.13	27.13	19.67	33.07	7.30	19.72	19.59	53.38
0.08	20.62	25.88	19.95	33.55	7.15	19.15	19.26	54.44
0.07	20.60	25.65	20.02	33.73	7.47	17.81	19.23	55.48
0.06	21.21	25.48	19.41	33.90	7.26	17.13	19.26	56.35
0.05	20.95	23.63	20.57	34.85	7.17	15.92	18.69	58.22
0.04	20.65	23.05	20.49	35.81	7.36	14.59	18.51	59.54
0.03	21.48	21.98	19.97	36.57	7.22	12.97	18.47	61.34
0.02	20.93	20.31	21.24	37.52	7.21	10.85	17.97	63.97
0.01	20.60	19.50	21.55	38.35	7.16	7.86	17.09	67.89
0.0	22.73	15.26	21.91	40.09	9.73	0.72	10.59	78.96
ALEA	19.16	37.74	18.96	24.14	10.11	34.28	21.02	34.59

Table A.5: Percentage of jobs based on λ value. Categorization into 4 groups.
instant: Jobs that started immediately after submitting.
[0,1.0]: Jobs with a change of node allocation one second before job start.
(1,600]: Node allocation which did not change 1 and 600 seconds before job start.
(600,∞): Jobs with a valid prediction over 600 seconds.

	CTC							
	Conservative + Backfill				FCFS			
λ	instant	[0, 1.0]	(1.0, 600.0]	(600.0, ∞)	instant	[0, 1.0]	(1.0, 600.0]	(600.0, ∞)
1.0	6.58	69.47	13.48	10.48	1.40	67.62	16.56	14.42
0.9	7.51	66.13	14.41	11.95	1.37	65.55	17.09	15.99
0.8	7.83	64.81	15.04	12.32	1.35	63.88	17.88	16.89
0.7	9.81	61.59	15.77	12.83	1.34	62.99	18.20	17.47
0.6	9.27	60.52	16.26	13.95	1.38	60.77	18.75	19.10
0.5	9.96	58.91	16.26	14.88	1.34	58.80	19.62	20.24
0.4	9.78	56.55	17.66	16.01	1.36	56.57	20.05	22.01
0.3	11.65	52.75	17.81	17.80	1.33	53.74	20.92	24.01
0.2	10.94	48.26	19.15	21.65	1.35	49.33	22.52	26.80
0.1	14.20	39.32	20.30	26.19	1.35	40.26	24.23	34.16
0.09	14.48	37.93	19.89	27.7	1.35	39.44	24.36	34.84
0.08	14.87	35.87	20.65	28.62	1.32	38.21	24.15	36.33
0.07	14.51	34.37	20.84	30.28	1.31	36.68	24.36	37.64
0.06	14.51	33.22	20.51	31.77	1.32	35.36	25.23	38.09
0.05	15.65	30.36	20.91	33.08	1.32	33.14	24.75	40.79
0.04	15.60	28.42	21.14	34.84	1.31	30.28	24.65	43.76
0.03	15.78	26.08	21.00	37.14	1.30	27.69	24.02	46.98
0.02	14.51	23.88	21.41	40.20	1.32	23.55	24.12	51.01
0.01	15.75	19.37	21.54	43.34	1.32	16.78	22.64	59.25
0.0	17.04	10.76	19.71	52.49	1.34	0.52	6.81	91.32
ALEA	5.53	48.81	26.93	18.73	1.38	47.50	29.89	21.22

Table A.6: Percentage of jobs based on λ value. Categorization into 4 groups.
instant: Jobs that started immediately after submitting.
[0,1.0]: Jobs with a change of node allocation one second before job start.
(1,600]: Node allocation which did not change 1 and 600 seconds before job start.
(600,∞): Jobs with a valid prediction over 600 seconds.

	KTH							
	Conservative + Backfilling				FCFS			
λ	instant	[0, 1.0]	(1.0, 600.0]	(600.0, ∞)	instant2	[0, 1.0]	(1.0, 600.0]	(600.0, ∞)
1.0	29.82	36.77	13.02	20.39	7.58	41.94	16.97	33.51
0.9	29.27	34.82	14.05	21.86	9.05	37.63	17.71	35.61
0.8	30.02	33.17	14.60	22.22	9.80	34.85	18.40	36.96
0.7	30.09	32.21	14.66	23.03	9.46	33.72	18.73	38.09
0.6	29.14	32.27	14.80	23.79	8.59	33.34	18.97	39.09
0.5	28.27	30.63	15.71	25.39	8.28	31.22	19.46	41.04
0.4	30.73	27.65	15.71	25.91	8.14	30.14	18.74	42.98
0.3	31.28	25.20	15.79	27.73	7.61	27.13	19.22	46.03
0.2	28.46	24.66	16.20	30.68	8.14	22.62	18.19	51.04
0.1	28.52	21.16	15.86	34.46	7.61	18.88	17.79	55.72
0.09	29.86	20.02	16.38	33.75	9.43	17.13	17.35	56.09
0.08	30.21	19.61	16.42	33.77	7.70	17.61	17.56	57.14
0.07	30.51	18.89	16.26	34.34	9.41	15.46	17.33	57.80
0.06	30.53	18.44	16.09	34.95	8.33	14.7	17.27	59.70
0.05	31.36	17.89	15.87	34.89	9.79	13.81	17.04	59.36
0.04	31.26	16.38	16.21	36.14	9.44	11.43	17.12	62.01
0.03	31.91	16.79	16.24	35.06	10.27	10.79	16.28	62.66
0.02	32.01	15.45	16.60	35.95	9.38	9.19	15.47	65.96
0.01	31.93	14.24	16.59	37.24	8.83	7.71	14.68	68.78
0.0	32.15	13.02	16.66	38.16	9.49	0.43	9.46	80.62
ALEA	26.54	33.94	17.06	22.45	10.37	34.17	21.25	34.21

Table A.7: Percentage of jobs based on λ value. Categorization into 4 groups.
instant: Jobs that started immediately after submitting.
[0,1.0]: Jobs with a change of node allocation one second before job start.
(1,600]: Node allocation which did not change 1 and 600 seconds before job start.
(600,∞): Jobs with a valid prediction over 600 seconds.

	ForHLR I							
	Conservative + Backfilling				FCFS			
λ	instant	[0, 1.0]	(1.0, 600.0]	(600.0, ∞)	instant	[0, 1.0]	(1.0, 600.0]	(600.0, ∞)
1.0	57.74	34.04	4.31	3.91	53.22	28.56	7.93	10.28
0.9	58.25	32.66	4.93	4.16	54.19	26.91	8.37	10.52
0.8	58.85	31.34	5.39	4.42	54.52	26.30	8.50	10.68
0.7	58.69	31.21	5.27	4.83	54.15	25.43	9.17	11.25
0.6	58.92	30.25	5.41	5.42	54.06	25.74	8.75	11.45
0.5	58.59	29.84	5.61	5.96	53.83	24.65	9.26	12.25
0.4	60.49	27.02	6.23	6.26	53.95	24.04	9.46	12.56
0.3	59.59	26.98	6.52	6.92	52.96	22.07	11.41	13.57
0.2	59.71	24.75	7.21	8.33	54.14	21.33	10.06	14.47
0.1	60.94	21.09	7.80	10.17	53.56	19.46	10.51	16.47
0.09	60.66	20.90	7.80	10.64	53.63	18.85	11.02	16.51
0.08	61.09	19.84	7.98	11.09	52.82	18.40	11.82	16.96
0.07	60.68	19.93	7.89	11.51	53.19	18.32	11.25	17.23
0.06	60.77	18.71	8.48	12.04	52.70	17.62	11.92	17.76
0.05	61.35	17.86	8.19	12.59	53.33	16.11	12.09	18.48
0.04	61.21	16.10	8.91	13.78	52.89	16.46	12.20	18.45
0.03	60.70	15.98	8.70	14.63	52.92	13.59	13.62	19.86
0.02	61.16	12.98	9.46	16.39	53.35	14.31	11.41	20.93
0.01	61.58	8.20	10.13	20.09	53.28	10.98	12.87	22.87
0.0	62.08	1.41	6.15	30.36	53.24	2.62	8.47	35.66
ALEA	56.68	4.88	18.04	20.40	53.31	5.35	20.02	21.32

Table A.8: Percentage of jobs based on λ value. Categorization into 4 groups.
instant: Jobs that started immediately after submitting.
[0,1.0]: Jobs with a change of node allocation one second before job start.
(1,600]: Node allocation which did not change 1 and 600 seconds before job start.
(600,∞): Jobs with a valid prediction over 600 seconds.

	ForHLR II							
	Conservative + Backfilling				FCFS			
λ	instant	[0, 1.0]	(1.0, 600.0]	(600.0, ∞)	instant	[0, 1.0]	(1.0, 600.0]	(600.0, ∞)
1.0	65.28	24.59	6.49	3.63	58.89	13.90	12.17	15.05
0.9	65.47	23.52	7.22	3.79	59.02	13.43	12.43	15.12
0.8	65.72	23.32	7.24	3.72	58.81	13.04	12.36	15.79
0.7	66.09	22.60	7.15	4.16	58.85	12.06	12.89	16.20
0.6	66.09	21.98	7.42	4.51	58.94	11.79	12.79	16.48
0.5	66.39	20.85	7.96	4.81	58.89	11.07	13.39	16.65
0.4	67.31	19.32	8.59	4.78	58.98	10.44	13.09	17.48
0.3	67.99	17.79	8.83	5.38	58.71	9.30	13.31	18.68
0.2	68.47	16.28	9.25	6.01	58.88	8.71	13.32	19.09
0.1	68.75	13.47	9.80	7.97	58.77	7.39	13.21	20.63
0.09	68.71	13.05	9.85	8.38	58.82	7.03	13.22	20.93
0.08	68.79	12.40	9.89	8.92	58.78	6.92	13.19	21.11
0.07	68.68	12.09	10.11	9.12	58.79	6.79	13.18	21.24
0.06	68.68	11.93	9.78	9.61	58.72	6.38	13.21	21.68
0.05	68.75	11.17	9.74	10.33	58.65	6.34	12.57	22.45
0.04	68.72	9.52	10.18	11.58	58.77	5.94	12.73	22.55
0.03	68.39	8.63	10.39	12.59	58.63	5.51	12.20	23.67
0.02	68.66	6.95	9.92	14.48	58.75	4.95	12.02	24.27
0.01	68.31	3.66	9.68	18.35	58.76	3.93	11.10	26.21
0.0	67.84	1.29	5.68	25.18	58.71	0.03	2.61	38.65
ALEA	60.65	4.18	12.01	23.16	58.85	2.31	12.51	26.32

Table A.9: Percentage of jobs based on λ value. Categorization into 4 groups.
instant: Jobs that started immediately after submitting.
[0,1.0]: Jobs with a change of node allocation one second before job start.
(1,600]: Node allocation which did not change 1 and 600 seconds before job start.
(600,∞): Jobs with a valid prediction over 600 seconds.

Bibliography

[1] H.-J. Bungartz, W. E. Nagel, P. Neumann, S. Reiz, and B. Uekermann. “Software for Exascale Computing: Some Remarks on the Priority Program SPPEXA”. In: *Software for Exascale Computing-SPPEXA 2016-2019*. Springer, 2020.

[2] *International Electrotechnical Commission*. `https://www.iec.ch/`. 09/23/2020.

[3] *Wikipedia - Ken Batcher*. `https://en.wikipedia.org/wiki/Ken_Batcher`. 09/23/2020.

[4] E. L. Miller and R. H. Katz. “Input/output behavior of supercomputing applications”. In: *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*. 1991, pp. 567–576.

[5] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. Long, and T. T. McLarty. “File system workload analysis for large scale scientific computing applications”. In: *Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies*. 2004, pp. 139–152.

[6] G. C. Fox, S. Jha, J. Qiu, and A. Luckow. “Towards an understanding of facets and exemplars of big data applications”. In: *Proceedings of the 20 Years of Beowulf Workshop on Honor of Thomas Sterling’s 65th Birthday*. 2014, pp. 7–16.

[7] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. “Big data and its technical challenges”. In: *Communications of the ACM* 57.7 (2014), pp. 86–94.

[8] R. T. Kouzes, G. A. Anderson, S. T. Elbert, I. Gorton, and D. K. Gracio. “The changing paradigm of data-intensive computing”. In: *Computer* 42.1 (2009), pp. 26–34.

[9] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. Byers. “Big data: The next frontier for innovation, competition, and productivity”. In: *http://www.mckinsey. com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation* (2011).

[10] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. “Understanding and improving computational science storage access through continuous characterization”. In: *ACM Transactions on Storage (TOS)* 7.3 (2011), pp. 1–26.

[11] D. A. Reed and J. Dongarra. “Exascale computing and big data”. In: *Communications of the ACM* 58.7 (2015), pp. 56–68.

[12] L. Freeman. “What’s Old Is New Again-Storage Tiering”. In: *SNW Spring2012* (2012).

[13] *ForHLR II, KIT/SCC*. 06/19/2017. URL: `https://www.scc.kit.edu/dienste/forhlr2.php`.

[14] A. Brinkmann, K. Mohror, W. Yu, P. Carns, T. Cortes, S. A. Klasky, A. Miranda, F.-J. Pfreundt, R. B. Ross, and M.-A. Vef. “Ad Hoc File Systems for High-Performance Computing”. In: *Journal of Computer Science and Technology* 35.1 (2020), pp. 4–26.

[15] *About Seymour Cray*. 09/19/2020. URL: `https://www.computer.org/volunteering/awards/cray/about-cray`.

[16] A. Pillepich, V. Springel, D. Nelson, S. Genel, J. Naiman, R. Pakmor, L. Hernquist, P. Torrey, M. Vogelsberger, R. Weinberger, and et al. “Simulating galaxy formation with the IllustrisTNG model”. In: *Monthly Notices of the Royal Astronomical Society* 473.3 (10/2017), pp. 4077–4106.

[17] A. Abbas-Bayoumi and K. Becker. “An industrial view on numerical simulation for aircraft aerodynamic design”. In: *Journal of Mathematics in Industry* 1.1 (2011), p. 10.

[18] S. Braun, S. Holz, L. Wieth, T. F. Dauch, M. C. Keller, G. Chaussonnet, C. Schwitzke, R. Koch, and H.-J. Bauer. “HPC Predictions of Primary Atomization with SPH: Validation and Comparison to Experimental Results”. In: *12th International SPHERIC Workshop*. 2017, pp. 314–321.

[19] G. Zängl, D. Reinert, P. Rípodas, and M. Baldauf. “The ICON (ICOsahedral Nonhydrostatic) modelling framework of DWD and MPI-M: Description of the nonhydrostatic dynamical core”. In: *Quarterly Journal of the Royal Meteorological Society* 141 (01/2015).

[20] *Das aktuelle numerische Wettervorhersage- und Notfallsystem*. URL: `https://www.dwd.de/DE/forschung/wettervorhersage/num_modellierung/06_num_wettervorhersage_notfallsyste/num_wettervorhersage_notfallsystem_node.html`.

[21] L. Adhianto and B. Chapman. “Performance modeling of communication and computation in hybrid MPI and OpenMP applications”. In: *Simulation Modelling Practice and Theory* 15.4 (2007), pp. 481–491.

[22] W. Gropp, W. D. Gropp, E. Lusk, A. Skjellum, and A. D. F. E. E. Lusk. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press, 1999.

[23] W. E. Johnston, E. Dart, M. Ernst, and B. Tierney. “Enabling high throughput in widely distributed data management and analysis systems: Lessons from the LHC”. In: *TERENA Networking Conference (TNC)*. 2013.

[24] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, and K. A. Persson. “FireWorks: a dynamic workflow system designed for high-throughput applications”. In: *Concurrency and Computation: Practice and Experience* 27.17 (2015). CPE-14-0307.R2, pp. 5037–5059.

[25] R. F. da Silva, R. Filgueira, I. Pietri, M. Jiang, R. Sakellariou, and E. Deelman. “A characterization of workflow management systems for extreme-scale applications”. In: *Future Generation Computer Systems* 75 (2017), pp. 228–238.

[26] W. E. Johnston. “High-speed, wide area, data intensive computing: A ten year retrospective”. In: *Proceedings. The Seventh International Symposium on High Performance Distributed Computing (Cat. No. 98TB100244).* IEEE. 1998, pp. 280–291.

[27] J. Dongarra, T. Sterling, H. Simon, and E. Strohmaier. “High-performance computing: clusters, constellations, MPPs, and future directions”. In: *Computing in Science & Engineering* 7.2 (2005), pp. 51–59.

[28] TOP500. *The TOP500 List.* `https://www.top500.org/`.

[29] K. Ishii, H. Abe, S. Kawabe, and M. Hirai. “An overview of the hitachi S-3800 series supercomputer”. In: *Supercomputer’92.* Springer, 1992, pp. 65–81.

[30] M. J. Flynn. “Some computer organizations and their effectiveness”. In: *IEEE transactions on computers* 100.9 (1972), pp. 948–960.

[31] J. R. Nickolls. “The design of the MasPar MP-1: a cost effective massively parallel computer.” In: *Compcon.* 1990, pp. 25–28.

[32] H. Jeong, S. Kim, W. Lee, and S.-H. Myung. “Performance of SSE and AVX instruction sets”. In: *arXiv preprint arXiv:1211.0820* (2012).

[33] D. Culler, J. P. Singh, and A. Gupta. *Parallel computer architecture: a hardware/software approach.* Gulf Professional Publishing, 1999.

[34] *Sun Fire 6800/4810/4800/3800 Systems Overview.* URL: `https://docs.oracle.com/cd/E19095-01/sf4810.srvr/805-7362-13/805-7362-13.pdf`.

[35] *FIRE 15K/FIRE 6800/SUN FIRE LINK.* URL: `https://www.top500.org/system/167255/`.

[36] J. Dongarra, I. Foster, G. Fox, K. Kennedy, and L. Torczon. *Sourcebook of Parallel Computing.* The Morgan Kaufmann Series in Computer Architecture and Design Series. Morgan Kaufmann, 2003.

[37] A. Vrenios. *Linux cluster architecture.* Sams, 2002.

[38] T. L. Sterling. *Beowulf cluster computing with Linux.* MIT press, 2002.

[39] K. E. Batcher. “Design of a massively parallel processor”. In: *IEEE Transactions on Computers* 9 (1980), pp. 836–840.

[40] *TOP 500 list - June 2013*. https://top500.org/lists/top500/2013/06/. 2012.

[41] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, et al. “The Sunway TaihuLight supercomputer: system and applications”. In: *Science China Information Sciences* 59.7 (2016), p. 072001.

[42] Z. Xu, J. Lin, and S. Matsuoka. “Benchmarking sw26010 many-core processor”. In: *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2017, pp. 743–752.

[43] G. Gostin, J.-F. Collard, and K. Collins. “The Architecture of the HP Superdome Shared-Memory Multiprocessor”. In: *Proceedings of the 19th Annual International Conference on Supercomputing*. ICS ’05. Cambridge, Massachusetts: Association for Computing Machinery, 2005, 239–245.

[44] A. Grama, V. Kumar, A. Gupta, and G. Karypis. *Introduction to parallel computing*. Pearson Education, 2003.

[45] B. Sosinsky. *Networking bible*. Vol. 567. John Wiley & Sons, 2009.

[46] J. Kim, W. J. Dally, S. Scott, and D. Abts. “Technology-driven, highly-scalable dragonfly topology”. In: *2008 International Symposium on Computer Architecture*. IEEE. 2008, pp. 77–88.

[47] J. Peter and T. Perttunen. *Network topologies*. https://www.stl.tech/sterlite-live/application_notes/1/original/Network_Topologies.pdf.

[48] B. Schürmann. *Grundlagen der Rechnerkommunikation: technische Realisierung von Bussystemen und Rechnernetzen; für alle Studiengänge: Informatik, Elektrotechnik und Informationstechnik*. Springer-Verlag, 2004.

[49] B. Jia. *Contention free pipelined broadcasting within a constant bisection bandwidth network topology*. US Patent 8,274,987. 09/2012.

[50] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection networks*. Morgan Kaufmann, 2003.

[51] T. Lammle. *CompTIA Network+ Study Guide Authorized Courseware: Exam N10-005*. John Wiley & Sons, 2012.

[52] C. E. Leiserson. “Fat-trees: universal networks for hardware-efficient supercomputing”. In: *IEEE transactions on Computers* 100.10 (1985), pp. 892–901.

[53] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, et al. “Blue Gene/L torus interconnection network”. In: *IBM Journal of Research and Development* 49.2.3 (2005), pp. 265–276.

[54] N. R. Adiga, G. Almási, G. S. Almasi, Y. Aridor, R. Barik, D Beece, R. Bellofatto, G. Bhanot, R. Bickford, M Blumrich, et al. "An overview of the BlueGene/L supercomputer". In: *SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE. 2002, pp. 60–60.

[55] J. S. Vetter, S. R. Alam, T. Dunigan, M. R. Fahey, P. C. Roth, and P. H. Worley. "Early evaluation of the Cray XT3". In: *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE. 2006, 10–pp.

[56] Y. Ajima, T. Kawashima, T. Okamoto, N. Shida, K. Hirai, T. Shimizu, S. Hiramoto, Y. Ikeda, T. Yoshikawa, K. Uchida, et al. "The tofu interconnect D". In: *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2018, pp. 646–654.

[57] TOP500. *Supercompouter - Supercomputer FUGAKU TOP500 List*. `https://www.top500.org/system/179807/`.

[58] M. Berghoff. *3d Torus for 4 X 4 X 4 nodes*.

[59] Steinbuch Centre for Computing. *bwUniCluster 2.0+GFB-HPC*. `http://www.scc.kit.edu/en/services/bwUniCluster_2.0.php`.

[60] *Forschungshochleistungsrechner ForHLR 1*. 2018. URL: `www.scc.kit.edu/dienste/forhlr1.php`.

[61] *Forschungshochleistungsrechner ForHLR 2*. 2018. URL: `www.scc.kit.edu/dienste/forhlr2.php`.

[62] OAK RIDGE National Laboratory. *Summit Fact Sheet*. Accessed: August 23, 2016. 2014. URL: `https://www.olcf.ornl.gov/wp-content/uploads/2014/11/Summit\_FactSheet.pdf`.

[63] Lawrence Livermore National Laboratory. *CORAL - Sierra*. Accessed: August 23, 2016. URL: `https://asc.llnl.gov/coral-info`.

[64] S. R. Walli. "The POSIX family of standards". In: *StandardView* 3.1 (1995), pp. 11–17.

[65] P. J. Braam and P. Schwan. "Lustre: The intergalactic file system". In: *Ottawa Linux Symposium*. 2002, p. 50.

[66] J. Heichler. *An introduction to BeeGFS*. 2014. URL: `http://www.beegfs.com/docs/Introduction\%5Fto\%5FBeeGFS\%5Fby\%5FThinkParQ.pdf`.

[67] *LBNL Node Health Check (NHC)*. 10/04/2020. URL: `https://github.com/mej/nhc`.

[68] I. Altair Engineering. *PBS Professional Open Source Project*. `https://www.pbspro.org/`.

[69] *Adaptive Computing*. `http://www.adaptivecomputing.com`.

[70] *IBM - Platform computing*. URL: `http://www.ibm.com/systems/platformcomputing/products/lsf/`.

[71] *Slurm - SchedMD*. `http://www.schedmd.com`.

[72] *Torque Resource Manager*. 2016. URL: `http://www.adaptivecomputing.com/products/open-source/torque-resource-manager`.

[73] P. Krzyzanowski. *Process Scheduling*. 2015. URL: `http://www.cs.rutgers.edu/\~pxk/416/notes/07-schedulinghtml`.

[74] S. Intel. “Serial ATA Native Command Queuing: An Exciting New Performance Feature for Serial ATA”. In: *Intel and Seagate (July 2003)* (2003).

[75] M. Hovestadt, O. Kao, A. Keller, and A. Streit. “Scheduling in HPC Resource Management Systems: Queuing vs. Planning”. In: *Job Scheduling Strategies for Parallel Processing*. Vol. 2862. 06/2003, pp. 1–20.

[76] D. G. Feitelson and L. Rudolph. “Toward convergence in job schedulers for parallel supercomputers”. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 1996, pp. 1–26.

[77] J. Y. Leung. *Handbook of scheduling: algorithms, models, and performance analysis*. CRC press, 2004.

[78] A. W. Mu’alem and D. G. Feitelson. “Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling”. In: *IEEE Trans. Parallel Distrib. Syst.* 12.6 (06/2001), pp. 529–543.

[79] D. A. Lifka. “The ANL/IBM SP scheduling system”. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 1995, pp. 295–303.

[80] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. “Selective Reservation Strategies for Backfill Job Scheduling”. In: *Job Scheduling Strategies for Parallel Processing*. Vol. 2537. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 55–71.

[81] S. Hoopes. *Moab Reservations*. `https://cug.org/proceedings/cug2014_proceedings/includes/files/tut107.pdf`. 2014.

[82] B. G. Lawson and E. Smirni. “Multiple-queue backfilling scheduling with priorities and reservations for parallel systems”. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2002, pp. 72–87.

[83] *Slurm - Multifactor Priority Plugin*. `https://slurm.schedmd.com/priority_multifactor.html`. 2020.

[84] S. D. Pollard, N. Jain, S. Herbein, and A. Bhatele. “Evaluation of an Interference-free Node Allocation Policy on Fat-tree Clusters”. In: *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2018, pp. 333–345.

[85] W. Smith, I. Foster, and V. Taylor. “Predicting application run times using historical information”. In: *Job Scheduling Strategies for Parallel Processing*. Springer, 1998, pp. 122–142.

[86] E. Gaussier, D. Glesser, V. Reis, and D. Trystram. “Improving backfilling by using machine learning to predict running times”. In: *SC’15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2015, pp. 1–10.

[87] R. O. Duda, P. E. Hart, and D. G. Stork. “Unsupervised learning and clustering”. In: *Pattern classification* (2001), pp. 517–601.

[88] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[89] F. Alimoglu, E. Alpaydin, and Y. Denizhan. “Combining multiple classifiers for pen-based handwritten digit recognition”. In: (1996).

[90] E. Anderson. “The species problem in Iris”. In: *Annals of the Missouri Botanical Garden* 23.3 (1936), pp. 457–509.

[91] C. Viswesvaran. “Multiple Regression in Behavioral Research: Explanation and Prediction”. In: *Personnel Psychology* 51.1 (1998), p. 223.

[92] J. L. Barlow. “Numerical aspects of solving linear least squares problems”. In: *Computer Science Department, The Pennsylvania State University, University Park, PA, USA* (1999).

[93] R. B. Darlington and A. F. Hayes. *Regression analysis and linear models: Concepts, applications, and implementation*. Guilford Publications, 2016.

[94] *scikit - Generalized Linear Models*. `http://scikit-learn.org/stable/modules/linear_model.html`. 06/19/2017.

[95] *scikit - Decision Trees*. `http://scikit-learn.org/stable/modules/tree.html`. 06/19/2017.

[96] X. He, K. Zhao, and X. Chu. *AutoML: A Survey of the State-of-the-Art*. 2019. arXiv: `1908.00709 [cs.LG]`.

[97] P. Gijsbers, E. LeDell, J. Thomas, S. Poirier, B. Bischl, and J. Vanschoren. *An Open Source AutoML Benchmark*. 2019. arXiv: `1907.00909 [cs.LG]`.

[98] I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, et al. “Analysis of the AutoML Challenge Series”. In: *Automated Machine Learning* (2019), p. 177.

[99] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. “Efficient and Robust Automated Machine Learning”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., 2015, pp. 2962–2970.

[100] R. S. Olson and J. H. Moore. “TPOT: A tree-based pipeline optimization tool for automating machine learning”. In: *Workshop on automatic machine learning*. PMLR. 2016, pp. 66–74.

[101] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[102] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[103] D. Henseler, B. Landsteiner, D. Petesch, C. Wright, and N. J. Wright. "Architecture and design of cray Datawarp". In: *Cray User Group CUG* (2016).

[104] DataDirect Networks. *IME - Flash native cache*. 2018. URL: `https://www.ddn.com/products/ime-flash-native-data-cache/`.

[105] F. Schmuck and R. Haskin. "GPFS: A Shared-Disk File System for Large Computing Clusters". In: *Proceedings of the 1st USENIX Conference on File and Storage Technologies*. FAST '02. Berkeley, CA, USA: USENIX Association, 2002.

[106] F. J. Pfreundt and S. Breuner. *A parallel file system – made in Germany*. 09/27/2020. URL: `https://storageconference.us/2012/Presentations/M08.Pfreundt.pdf`.

[107] *ThinkparQ*. 09/27/2020. URL: `https://thinkparq.com/`.

[108] *BeeGFS Parallel File System Now Open Source*. 09/27/2020. URL: `https://www.hpcwire.com/off-the-wire/beegfs-parallel-file-system-now-open-source/`.

[109] W. Digital. *Ultrastar DC HC500 Series*. 09/14/2020. URL: `https://www.westerndigital.com/products/data-center-drives/ultrastar-dc-hc500-series-hdd`.

[110] Samsung. *Samsung SSD 850 PRO*. 09/14/2020. URL: `https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/850pro/`.

[111] Samsung. *NVMe SSD 960 PRO | EVO*. 09/14/2020. URL: `https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/ssd960/`.

[112] Samsung. *Samsung 980 PRO*. 09/23/2020. URL: `https://news.samsung.com/global/samsung-delivers-next-level-ssd-performance-with-980-pro-for-gaming-and-high-end-pc-applications`.

[113] Q. Xu, H. Siyamwala, M. Ghosh, T. Suri, M. Awasthi, Z. Guz, A. Shayesteh, and V. Balakrishnan. "Performance analysis of NVMe SSDs and their implication on real world databases". In: *Proceedings of the 8th ACM International Systems and Storage Conference*. ACM. 2015, p. 6.

[114] Lawrence Livermore National Laboratory. *ASC Sequoia.* 10/2012. URL: https://asc.llnl.gov/publications/Sequoia2012.pdf.

[115] B. Behlendorf. *Sequoia's 55PB Lustre+ZFS Filesystem.* 10/08/2020. URL: https://asc.llnl.gov/publications/Sequoia2012.pdf.

[116] Tom Papatheodore. *Summit System Overview.* Accessed: Oktober, 2018. 06/01/2018. URL: https://www.olcf.ornl.gov/wp-content/uploads/2018/05/Intro_Summit_System_Overview.pdf.

[117] *HoreKa - KIT Procures New Supercomputer.* https://www.scc.kit.edu/en/services/horeka.php. 09/22/2020.

[118] *Management of research data – a key strategic challenge for university management.* 09/14/2020. URL: https://www.hrk.de/resolutions-publications/resolutions/beschluss/detail/management-of-research-data-a-key-strategic-challenge-for-university-managemen.

[119] Deutsche Forschungsgemeinschaft. *Guidelines for Safeguarding Good Research Practice. Code of Conduct.* Available in German and in English. 09/2019. URL: https://doi.org/10.5281/zenodo.3923602.

[120] Helmholtz Open Science External Organizations and Library, Scientific Infrastructure and Platforms. "Recommendations for guidelines at the Helmholtz centres on handling research data. Approved in the 109th General Assembly of the Helmholtz Association on 13-14 September 2017". In: (2019).

[121] *Satzung zur Sicherung guter wissenschaftlicher Praxis am 160 Karlsruher Institut für Technologie (KIT).* https://www.sle.kit.edu/downloads/AmtlicheBekanntmachungen/2018_AB_032.pdf. 05/2018.

[122] J. Weingand. *IBM - Tape the future of tape.* 10/08/2020. URL: https://de.slideshare.net/JosefWeingand/ibm-tape-the-future-of-tape.

[123] F. J. Pfreundt and S. Breuner. *Designing a high performance tape archive.* 10/08/2020. URL: https://www.versity.com/blog/designing-a-high-performance-tape-archive.

[124] HP. *HP StoreEver Tape Library - Performance Problems.* 10/08/2020. URL: https://storageconference.us/2012/Presentations/M08.Pfreundt.pdf.

[125] B. Lee. *Bruce Lee Jeet Kune Do: Bruce Lee's Commentaries on the Martial Way.* Vol. 3. Tuttle Publishing, 2020.

[126] *ForHLR I, KIT/SCC.* 06/19/2017. URL: https://www.scc.kit.edu/dienste/forhlr1.php.

[127] *Intel Xeon Processor E5-2660 v3.* 06/19/2017. URL: https://ark.intel.com/content/www/us/en/ark/products/81706/intel-xeon-processor-e5-2660-v3-25m-cache-2-60-ghz.html.

[128] *Intel Xeon Processor E7-4830 v3*. 06/19/2017. URL: https://ark.intel.com/content/www/us/en/ark/products/84678/intel-xeon-processor-e7-4830-v3-30m-cache-2-10-ghz.html.

[129] *GEFORCE® GTX900 SERIES*. 06/19/2017. URL: https://www.nvidia.com/en-us/geforce/900-series/.

[130] Arden and H. Lee. "Analysis of Chordal Ring Network". In: *IEEE Transactions on Computers* C-30.4 (04/1981), pp. 291–295.

[131] *ForHLR - Hardware and Architecture*. 2020. URL: https://wiki.scc.kit.edu/hpc/index.php?title=ForHLR_-_Hardware_and_Architecture\#File_Systems.

[132] J. A. von der Gönna. *Introduction to the LRZ HPC Infrastructure*. 09/14/2020. URL: https://www.lrz.de/services/compute/courses/x_lecturenotes/191008_Intro-LRZ-HPC_web.pdf.

[133] *TTU HPC Center Data Policy*. 09/14/2020. URL: https://www.depts.ttu.edu/hpcc/operations/datapolicy.php.

[134] *NREL HPC Center Data Retention Policy*. 09/14/2020. URL: https://www.nrel.gov/hpc/data-retention-policy.html.

[135] *bwDataArchive - long-term archival storage of research data*. 06/19/2017. URL: http://www.scc.kit.edu/en/services/11228.php.

[136] *bwDataArchive - long-term archival storage of research data*. 06/19/2017. URL: https://www.rda.kit.edu/english/.

[137] A. Torrez, B. Kettering, W. Loewe, and R. Klundt. *MDtest metadata benchmark*. https://sourceforge.net/projects/mdtest/. Accessed: September 4, 2016. 2015.

[138] D. Klusáček and H. Rudová. "Alea 2: job scheduling simulator". In: *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. 2010, pp. 1–10.

[139] *Parallel Workloads Archive*. http://www.cs.huji.ac.il/labs/parallel/workload/. 02/12/2018.

[140] BeeGFS. *BeeOND™: BeeGFS On Demand*. Accessed: August 18 2018. URL: http://www.beegfs.io/wiki/BeeOND.

[141] BeeGFS. *BeeGFS*. Accessed: September 21 2020. URL: https://www.beegfs.io/docs/BeeGFS_Flyer.pdf.

[142] M.-A. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann. "GekkoFS-a temporary distributed file system for HPC applications". In: *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE. 2018, pp. 319–324.

[143] *losetup - set up and control loop devices*. https://www.linux.org/docs/man8/losetup.html. 09/22/2020.

[144] K. Yamamoto, F. Shoji, A. Uno, S. Matsui, K. Sakai, F. Sueyasu, S. Sumimoto, and F. Limited. *Analysis and Elimination of Client Evictions on a Large Scale Lustre Based File System*. en.

[145] D. Capps and W. Norcott. *IOzone filesystem benchmark*. 2008. URL: http://iozone.org/.

[146] *I/O 500*. 2020. URL: https://www.vi4io.org/start.

[147] OpenCFD. *OpenFOAM: The Open Source CFD Toolbox. User Guide Version 1.4, OpenCFD Limited.* Reading UK, 04/2007.

[148] *The OpenFOAM Foundation*. https://openfoam.org/. 2018.

[149] T. Zirwes, F. Zhang, J. Denev, P. Habisreuther, and H. Bockhorn. "Improved Vectorization for Efficient Chemistry Computations in OpenFOAM for Large Scale Combustion Simulations". In: *High Performance Computing in Science and Engineering '17*. Ed. by W. Nagel, D. Kröner, and M. Resch. Springer, 2018.

[150] T. Zirwes, F. Zhang, T. Häber, and H. Bockhorn. "Ignition of combustible mixtures by hot particles at varying relative speeds". In: *Combustion Science and Technology* 0.0 (2018), pp. 1–18. eprint: https://doi.org/10.1080/00102202.2018.1435530.

[151] M. Berghoff, I. Kondov, and J. Hötzer. "Massively Parallel Stencil Code Solver with Autonomous Adaptive Block Distribution". In: *IEEE Transactions on Parallel and Distributed Systems* 29.10 (2018), pp. 2282–2296.

[152] *Hybrid Data Analysis and Integration for Structural Biology*. 10/09/2020. URL: http://www.helmholtz-analytics.de/helmholtz_analytics/EN/UseCases/StructuralBiology/_node.html.

[153] C. DiBona and S. Ockman. *Open sources: Voices from the open source revolution*. Appendix A - The Tanenbaum-Torvalds Debate. O'Reilly Media, Inc., 1999.

[154] S. Maneas, K. Mahdaviani, T. Emami, and B. Schroeder. "A Study of {SSD} Reliability in Large Scale Enterprise Storage Deployments". In: *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*. 2020, pp. 137–149.

[155] M. Soysal, M. Berghoff, T. Zirwes, M.-A. Vef, S. Oeste, A. Brinkman, W. E. Nagel, and A. Streit. "Using On-demand File Systems in HPC Environments". In: *2019 International Conference on High Performance Computing and Simulation (HPCS)* (2019).

[156] M. Soysal and A. Streit. "Ad-hoc file systems at extreme scales". In: *High Performance Computing in Science and Engineering'19 (HLRS)* (10/2019). Accepted.

[157] J. Xing, J. Xiong, N. Sun, and J. Ma. "Adaptive and Scalable Metadata Management to Support a Trillion Files". In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. SC '09. New York, NY, USA: ACM, 2009, 26:1–26:11.

[158] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock. "I/O performance challenges at leadership scale". In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 11/2009, pp. 1–12.

[159] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. "On the role of burst buffers in leadership-class storage systems". In: *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. 04/2012, pp. 1–11.

[160] T. Wang, K. Mohror, A. Moody, K. Sato, and W. Yu. "An Ephemeral Burst-Buffer File System for Scientific Applications". In: *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 11/2016, pp. 807–818.

[161] CRAY. *Cray® DataWarp™ Applications I/O Accelerator*. 2018. URL: `https://www.cray.com/datawarp`.

[162] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir. "Scheduling the I/O of HPC applications under congestion". In: *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2015, pp. 1013–1022.

[163] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate. "PLFS: a checkpoint filesystem for parallel applications". In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 11/2009, pp. 1–12.

[164] IBM. *GPFS - Highly available write cache (HAWC)*. 2018. URL: `https://www.ibm.com/support/knowledgecenter/en/STXKQY\%5F5.0.0/com.ibm.spectrum.scale.v5r00.doc/bl1adv\%5Fhawc.htm`.

[165] R. Mohr, M. Brim, S. Oral, and A. Dilger. "Evaluating progressive file layouts for Lustre". In: *Cray User Group Conference (CUG 2016)*. 2016.

[166] BeeGFS. *BeeGFS Storage Pool*. Accessed: August 18 2018. 2018. URL: `https://www.beegfs.io/wiki/StoragePools`.

[167] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson. "An overview of the HDF5 technology suite and its applications". In: *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. ACM. 2011, pp. 36–47.

[168] R. Rew and G. Davis. "NetCDF: an interface for scientific data access". In: *IEEE computer graphics and applications* 10.4 (1990), pp. 76–82.

[169] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)". In: *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*. ACM. 2008, pp. 15–24.

[170] R. Thakur, W. Gropp, and E. Lusk. "On implementing MPI-IO portably and with high performance". In: *Proceedings of the sixth workshop on I/O in parallel and distributed systems*. ACM. 1999, pp. 23–32.

[171] R. Thakur, W. Gropp, and E. Lusk. “Data sieving and collective I/O in ROMIO”. In: *Frontiers of Massively Parallel Computation, 1999. Frontiers’ 99. The Seventh Symposium on the*. IEEE. 1999, pp. 182–189.

[172] W. Frings. *SIONlib*. 2009. URL: `https://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SIONlib/_node.html`.

[173] M. Neuer, J. Salk, H. Berger, E. Focht, C. Mosch, K. Siegmund, V. Kushnarenko, S. Kombrink, and S. Wesner. “Motivation and Implementation of a Dynamic Remote Storage System for I/O Demanding HPC Applications”. In: *International Conference on High Performance Computing*. Springer. 2016, pp. 616–626.

[174] D. Teigland and H. Mauelshagen. “Volume Managers in Linux.” In: *USENIX Annual Technical Conference, FREENIX Track*. 2001, pp. 185–197.

[175] TU Dresden. *HPC FOR DATA ANALYTICS*. `https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/HPCDA`.

[176] T. Wickberg and C. Carothers. “The RAMDISK Storage Accelerator: A Method of Accelerating I/O Performance on HPC Systems Using RAMDISKs”. In: *Proceedings of the 2Nd International Workshop on Runtime and Operating Systems for Supercomputers*. ROSS ’12. New York, NY, USA: ACM, 2012, 5:1–5:8.

[177] *Adaptive Computing - Data staging*. `http://www.adaptivecomputing.com/blog-hpc/data-staging/`.

[178] SchedMD. *Slurm Burst Buffer Guide*. [Online; last accessed am 2020-09-21]. URL: `https://slurm.schedmd.com/burst_buffer.html`.

[179] V. Voigt. “Entwicklung eines on-demand Burst-Buffer-Plugins für HPC-Batch-Systeme”. Bachelor. Steinbuch Centre for Computing (SCC), 04/10/2019.

[180] E. Kim. *SSD Performance-A Primer: An Introduction to Solid State Drive Perfromance, Evaluation and Test*. Tech. rep., Storage Networking Industry Association, 2013.

[181] J. Garlick. *PDSH*. `https://github.com/chaos/pdsh`. 2018.

[182] M. Soysal. *Speeding up BeeOND startup - Google Groups*. 2018. URL: `https://groups.google.com/g/fhgfs-user/c/g8ysFS35Ucs/m/E-RtxKyiCAAJ`.

[183] R. S. Barlow, S. Meares, G. Magnotti, H. Cutcher, and A. R. Masri. “Local extinction and near-field structure in piloted turbulent CH4/air jet flames with inhomogeneous inlets”. In: *Combust. Flame* 162.10 (2015), pp. 3516–3540.

[184] T. Zirwes, F. Zhang, J. Denev, P. Habisreuther, and H. Bockhorn. “Automated Code Generation for Maximizing Performance of Detailed Chemistry Calculations in OpenFOAM”. In: *High Performance Computing in Science and Engineering ’17*. Ed. by W. Nagel, D. Kröner, and M. Resch. Springer, 2017, pp. 189–204.

[185] D Sikich, G Di Natale, M LeGendre, and A Moody. *mpiFileUtils: A Parallel and Distributed Toolset for Managing Large Datasets*. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2017.

[186] L. Galambos. *The papers of Dwight David Eisenhower, Columbia University: Volume XI*. 1984.

[187] M. Soysal, M. Berghoff, and A. Streit. "Analysis of Job Metadata for Enhanced Wall Time Prediction". In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2018, pp. 1–14.

[188] M. Soysal, M. Berghoff, D. Klusáček, and A. Streit. "On the Quality of Wall Time Estimates for Resource Allocation Prediction". In: *Proceedings of the 48th International Conference on Parallel Processing: Workshops*. ICPP 2019, Kyoto, Japan. New York, NY, USA: ACM, 2019, 23:1–23:8.

[189] R. Gibbons. "A historical application profiler for use by parallel schedulers". In: *Job scheduling strategies for parallel processing*. Springer. 1997, pp. 58–77.

[190] R. Gibbons. "A historical profiler for use by parallel schedulers". In: *Master's thesis, University of Toronto* (1997).

[191] A. B. Downey. "Predicting queue times on space-sharing parallel computers". In: *Parallel Processing Symposium, 1997. Proceedings., 11th International*. IEEE. 1997, pp. 209–218.

[192] W. Tang, N. Desai, D. Buettner, and Z. Lan. "Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P". In: *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE, 2010, pp. 1–11.

[193] D. Klusáček, v. Tóth, and G. Podolníková. "Complex Job Scheduling Simulations with Alea 4". In: *Ninth EAI International Conference on Simulation Tools and Techniques (SimuTools 2016)*. ACM, 2016, pp. 124–129.

[194] *Alea 4: Job Scheduling Simulator*. https://github.com/aleasimulator. 02/2019.

[195] N. H. Kapadia and J. A. Fortes. "On the design of a demand-based network-computing system: The Purdue University Network-Computing Hubs". In: *The 7th International Symposium on High Performance Distributed Computing*. IEEE, 1998, pp. 71–80.

[196] F. Nadeem and T. Fahringer. "Using templates to predict execution time of scientific workflow applications in the grid". In: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2009, pp. 316–323.

[197] D. Tsafrir, Y. Etsion, and D. G. Feitelson. "Backfilling using system-generated predictions rather than user runtime estimates". In: *IEEE Transactions on Parallel and Distributed Systems* 18.6 (2007).

[198] W. Smith. “Prediction services for distributed computing”. In: *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE. 2007, pp. 1–10.

[199] *XSEDE*. `https://www.xsede.org/`. 04/27/2018.

[200] B.-D. Lee et al. “Run-time prediction of parallel applications on shared environments”. In: *2003 Proceedings IEEE International Conference on Cluster Computing*. IEEE. 2003, pp. 487–491.

[201] S. Seneviratne and D. C. Levy. “Task profiling model for load profile prediction”. In: *Future Generation Computer Systems* 27.3 (2011), pp. 245–255.

[202] N. H. Kapadia, J. A. Fortes, and C. E. Brodley. “Predictive application-performance modeling in a computational grid environment”. In: *Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No. 99TH8469)*. IEEE. 1999, pp. 47–54.

[203] M. A. Iverson, F. Ozguner, and L. C. Potter. “Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment”. In: *Proceedings. Eighth Heterogeneous Computing Workshop (HCW’99)*. IEEE. 1999, pp. 99–111.

[204] T. Miu and P. Missier. “Predicting the execution time of workflow activities based on their input features”. In: *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE. 2012, pp. 64–72.

[205] H. Li, D. Groep, and L. Wolters. “An evaluation of learning and heuristic techniques for application run time predictions”. In: *Proceedings of 11th Annual Conference of the Advance School for Computing and Imaging (ASCI), Netherlands*. Citeseer. 2005.

[206] R. F. Da Silva, G. Juve, M. Rynge, E. Deelman, and M. Livny. “Online task resource consumption prediction for scientific workflows”. In: *Parallel Processing Letters* 25.03 (2015), p. 1541003.

[207] A. Matsunaga and J. A. Fortes. “On the use of machine learning to predict the time and resources consumed by applications”. In: *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 495–504.

[208] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. “Basic local alignment search tool”. In: *Journal of molecular biology* 215.3 (1990), pp. 403–410.

[209] A. Stamatakis. “RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies”. In: *Bioinformatics* 30.9 (2014), pp. 1312–1313.

[210] H. Casanova. “Simgrid: a toolkit for the simulation of application scheduling”. In: *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. 2001, pp. 430–437.

[211] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. “Overview of a performance evaluation system for global computing scheduling algorithms”. In: *Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No.99TH8469)*. 1999, pp. 97–104.

[212] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya. “A toolkit for modelling and simulating data Grids: an extension to GridSim”. In: *Concurrency and Computation: Practice and Experience* 20.13 (2008), pp. 1591–1609.

[213] *The Standard Workload Format*. 04/27/2018. URL: `https://www.cs.huji.ac.il/labs/parallel/workload/swf.html`.

[214] *The San Diego Supercomputer Center (SDSC) SP2 log*. 01/30/2019. URL: `http://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_sp2/`.

[215] *The Swedish Royal Institute of Technology (KTH) IBM SP2 log*. 01/30/2019. URL: `http://www.cs.huji.ac.il/labs/parallel/workload/l_kth_sp2/`.

[216] S. Hotovy. “Workload evolution on the Cornell Theory Center IBM SP2”. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by D. G. Feitelson and L. Rudolph. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 27–40.

[217] *scikit - Regression Metrics*. `http://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics`. 06/19/2017.

[218] *scikit - R2 Score*. `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score`. 06/19/2017.

[219] *scikit - Mean absolute error*. `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html#sklearn.metrics.mean_absolute_error`. 06/19/2017.

[220] *scikit - Median absolute error*. `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.median_absolute_error.html#sklearn.metrics.median_absolute_error`. 06/19/2017.

[221] D. Tsafrir. “Using inaccurate estimates accurately”. In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2010, pp. 208–221.

[222] D. G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. 1st. New York, NY, USA: Cambridge University Press, 2015.

[223] M.-A. Vef, N. Moti, T. Süß, T. Tocci, R. Nou, A. Miranda, T. Cortes, and A. Brinkmann. “GekkoFS - A Temporary Distributed File System for HPC Applications”. In: *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. 2018 IEEE International Conference on Cluster Computing (CLUSTER). 2018, pp. 319–324.

[224] *Chairman Jobs? The Guessing Goes On*. 1997. URL: `https://www.wired.com/1997/07/chairman-jobs-the-guessing-goes-on/`.

[225] M. Soysal and A. Streit. "JAWS: Just Another Workspace Suite". In: *International Conference on High Performance Computing and Simulation (HPCS)* (12/2020). Submitted.

[226] E. Copernicus. *Copernicus open access hub*. 2017. URL: https://scihub.copernicus.eu/.

[227] C. Tona and R. Bua. "Open Source Data Hub System: free and open framework to enable cooperation to disseminate Earth Observation data and geo-spatial information." In: *Geophysical Research Abstracts* 20 (2018), p. 3808.

[228] *Nature - Recommended Data Repositories*. 09/26/2020. URL: https://www.nature.com/sdata/policies/repositories.

[229] R. W. Watson and R. A. Coyne. "The Parallel I/O Architecture of the High-Performance Storage System (HPSS)". In: *Proceedings of the 14th IEEE Symposium on Mass Storage Systems*. MSS '95. USA: IEEE Computer Society, 1995, p. 27.

[230] *IBM - Spectrum Protect*. 09/26/2020. URL: https://www.ibm.com/support/knowledgecenter/SSEQVQ_8.1.10/tsm/welcome.html.

[231] *Sun QFS and Sun Storage Archive Manager 5.3 Information Library*. 09/26/2020. URL: https://docs.oracle.com/cd/E22586_01/html/E22571/gkwas.html.

[232] *Hierarchical Storage Management in the Lustre* Filesystem*. 09/26/2020. URL: http://lustrefs.cn/wp-content/uploads/2018/08/HSM_Zhanghongchao.pdf.

[233] *Lustre Projects*. 09/26/2020. URL: https://wiki.lustre.org/Projects.

[234] *LUSTRE/HSM BINDING IS THERE!* 09/26/2020. URL: https://www.eofs.eu/_media/events/lad13/10_aurelien_degremont_lustre_hsm_lad13.pdf.

[235] *TSM Copytool for Lustre HSM*. 09/26/2020. URL: https://www.eofs.eu/_media/events/lad16/08_tsm_copytool_for_lustre_stibor.pdf.

[236] V. Garonne, R Vigne, G Stewart, M Barisits, M Lassnig, C Serfon, L Goossens, A Nairz, A. Collaboration, et al. "Rucio–The next generation of large scale distributed system for ATLAS Data Management". In: *Journal of Physics: Conference Series*. Vol. 513. IOP Publishing. 2014, p. 042021.

[237] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, and et al. "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems". In: *Sci. Program.* 13.3 (07/2005), pp. 219–237.

[238] M. Tanaka and O. Tatebe. “Pwrake: A Parallel and Distributed Flexible Workflow Management Tool for Wide-Area Data Intensive Computing”. In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC ’10, Chicago, Illinois. New York, NY, USA: Association for Computing Machinery, 2010, pp. 356–359.

[239] W. Shin, C. D. Brumgard, B. Xie, S. S. Vazhkudai, D. Ghoshal, S. Oral, and L. Ramakrishnan. “Data Jockey: Automatic Data Management for HPC Multi-tiered Storage Systems”. In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. ISSN: 1530-2075. 05/2019, pp. 511–522.

[240] H. Berger. *HPC Workspaces*. `https://github.com/holgerBerger/hpc-workspace`. 2020.

[241] *https://wiki.bwhpc.de/e/Workspace*. 09/26/2020. URL: `https://wiki.bwhpc.de/e/Workspace`.

[242] G. A. Jaquette. “LTO: A better format for mid-range tape”. In: *IBM Journal of Research and Development* 47.4 (2003), pp. 429–444.

[243] B. Haeusser, E. Winters, and E. Zwemmer. *IBM System Storage Tape Library Guide for Open Systems*. IBM, International Technical Support Organization, 2006.

[244] *Blackblaze*. 10/04/2020. URL: `https://www.backblaze.com/`.

[245] *Backblaze - Hard Drive Cost Per Gigabyte*. 10/04/2020. URL: `https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/`.

[246] Los Alamos National Laboratory. *Ultrascale Systems Research Center (USRC) Data Sources*. `https://usrc.lanl.gov/data-sources.php`.

[247] M. Weimer. “Towards Seamless Simulations of Polar Stratospheric Clouds and Ozone in the Polar Stratosphere with ICON-ART”. PhD thesis. Karlsruher Institut für Technologie (KIT), 2019. 165 pp.

[248] T. Zirwes, F. Zhang, Y. Wang, P. Habisreuther, J. A. Denev, Z. Chen, H. Bockhorn, and D. Trimis. “In-situ flame particle tracking based on barycentric coordinates for studying local flame dynamics in pulsating Bunsen flames”. In: *Proceedings of the Combustion Institute* (2020).

[249] J. Mantel, J. Ohm, A. Beer, M. Beichter, A. Fedorchenko, D. Dehtyarov, and M. Soysal. “DMD - Data Movement Daemon”. Steinbuch Centre for Computing (SCC).

[250] M. Pilgrim and S. Willison. *Dive Into Python 3*. Vol. 2. Springer, 2009.

[251] M. Grinberg. *Flask web development: developing web applications with python*. O’Reilly Media, Inc., 2018.

[252] R. T. Fielding. “REST: architectural styles and the design of network-based software architectures”. In: *Doctoral dissertation, University of California* (2000).

[253] R. Copeland. *Essential SQLalchemy*. O'Reilly Media, Inc., 2008.

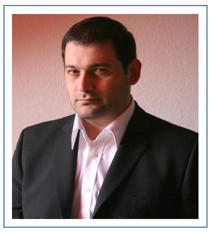
[254] C Dunlap. *Munge Uid N Grid Emporium*. Tech. rep. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2004.

[255] K. Krarup, N. C. Nielsen, S. B. Larsen, and S. L. Bak. "Kommunikation erstatter transport: den digitale revolution i danske forskningsbiblioteker 1980-2005: festskrift til Karl Krarup". In: vol. 30. Museum Tusculanum Press, 2005, p. 321.

[256] S. Versick, O. Kirner, J. Meyer, H. Obermaier, and M. Soysal. "Performance gains in an ESM using parallel ad-hoc file systems". In: *EGU General Assembly Conference Abstracts*. 2020, p. 18121.

[257] A. Xiang, M. Deitersen, M. Potthoff, P. Brosemer, R.-M. Hanne, and M. Soysal. "BOB - Bob offers Benchmarks - A new Scheduling Simulator". Steinbuch Centre for Computing (SCC).

[258] D. Dehtyarov. "Simulation der Knotenausfälle und Reservierungen in HPC Systemen". Bachelor. Steinbuch Centre for Computing (SCC), 09/23/2019.



Mehmet Soysal

Veröffentlichungen

Veröffentlichung

2016	**Exploring opportunities for job-temporal file systems with ADA-FS**, *Sebastian Oeste, Michael Kluge, Mehmet Soysal, Achim Streit, Marc-André Vef, and André Brinkmann*, 1st Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems, 2016..
2018	**Analysis of Job Metadata for Enhanced Wall Time Prediction**, *M. Soysal, M. Berghoff, and A. Streit*, In Job Scheduling Strategies for Parallel Processing: 22nd International Workshop, JSSPP 2018, Vancouver, BC, Canada.
2018	**Batch job scheduling using enhanced walltime prediction.**, *Dalibor Klusáček and Mehmet Soysal*, In CGW Workshop '18 (CGW'18).
2019	**ADA-FS – Advanced Data Placement via Ad hoc File Systems at Extreme Scales**, *Sebastian Oeste, Marc-André Vef, Mehmet Soysal, Wolfgang E. Nagel, André Brinkmann, and Achim Streit*, Software for Exascale Computing - SPPEXA 2016-2019, Lecture Notes in Computational Science and Engineering, page 29–59, Springer International Publishing, 2020.
2019	**Alea–complex job scheduling simulator**, *Dalibor Klusáček, Mehmet Soysal, and Frédéric Suter.*, In International Conference on Parallel Processing and Applied Mathematics, pages 217–229., Springer.
2019	**Using On-demand File Systems in HPC Environments**, *Mehmet Soysal, Marco Berghoff, Thorsten Zirwes, Marc-André Vef, Sebastian Oeste, Andre Brinkmann, Wolfgang E. Nagel*, In 2019 International Conference on High Performance Computing Simulation (HPCS), pages 390–398, Dublin.

2019 **On the Quality of Wall Time Estimates for Resource Alloca tion Prediction**, *Mehmet Soysal, Marco Berghoff, Dalibor Klusáček, and Achim Streit*, In Proceedings of the 48th International Conference on Parallel Processing: Workshops, ICPP 2019, pages 23:1–23:8,, New York, NY, USA, ACM.

2019 **Ad-hoc file systems at extreme scales**, *Mehmet Soysal and Achim Streit*, Accepted for publishing – High Performance Computing in Science and Engineering'19 (HLRS).

2020 **Walltime prediction and its impact on job scheduling performance and predictability**, *Dalibor Klusáček and Mehmet Soysal*, In Job Scheduling Strategies for Parallel Processing: 22nd International Workshop, JSSPP 2020.

2020 **JAWS: Just Another Workspace Suite**, *Mehmet Soysal and Achim Streit*, International Conference on High Performance Computing and Simulation (HPCS) 2020, Submitted.

Vorträge und Poster

2017 **Poster: SBD Workshop**, *ADA-FS Advanced Data Placement via Ad-hoc File Systems*, Jülich.

2018 **Vortrag: Analysis of Job Metadata for Enhanced Wall Time Prediction**, *M. Soysal, M. Berghoff, and A. Streit*, In Job Scheduling Strategies for Parallel Processing: 22nd International Workshop, Vancouver, BC, Canada.

2019 **Vortrag: Using On-demand File Systems in HPC Environments**, *Mehmet Soysal, Marco Berghoff, Thorsten Zirwes, Marc-André Vef, Sebastian Oeste, Andre Brinkmann, Wolfgang E. Nagel*, In 2019 International Conference on High Performance Computing Simulation, Dublin, Ireland.

2019 **Vortrag: On the Quality of Wall Time Estimates for Resource Allocation Prediction**, *Mehmet Soysal, Marco Berghoff, Dalibor Klusáček, and Achim Streit*, The 48th International Conference on Parallel Processing: Workshops, Kyoto, Japan.

2019 **Poster: ADA-FS – Advanced Data Placement via Ad hoc File Systems at Extreme Scales**, *HLRS*.