Decision Support for Workflow-Aware High-Performance Storage Systems

14.05.2021 // **DISSERTATION**
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
DR. RER. NAT.
AN DER FAKULTÄT FÜR MATHEMATIK, INFORMATIK UND NATURWISSENSCHAFTEN DER UNIVERSITÄT HAMBURG

EINGEREICHT BEIM FACHBEREICH INFORMATIK VON
JAKOB LÜTTGAU AUS HAMBURG

Universität Hamburg
Department of Informatics
Scientific Computing

UNIVERSITÄT HAMBURG, DEPARTMENT OF INFORMATICS, SCIENTIFIC COMPUTING

First printing, January 2022

GUTACHTEN:
PROF. DR. THOMAS LUDWIG
PROF. DR. PHILIP CARNS
PROF. DR. WOLFGANG NAGEL

19.10.2021 // DATUM DER **DISPUTATION**

Abstract

As a result of technological advancements and data center economics, rapid changing systems and increasing complexity render manual system-specific tuning of application and workflow input/output (I/O) unfeasible. The next generation of computer systems enable and require adaptive heuristics for optimal system exploitation. A key challenge to empower adaptive heuristics is to provide decision components with the necessary information (decision aids) to base decisions on. In many cases, these decisions will have to negotiate the user's intent, the temporal unraveling of I/O activity, and an operational perspective. This notion of intent is embedded in the workflows of users and scientists.

The thesis conducts an extensive analysis of scientific workflows as well as the storage data path to inform an architecture proposal adding workflow-adaptivity to existing and future storage systems. At the core of this architecture are decision points and information sources scattered across high-performance computing systems. For a particular workflow, the assumption is, that various decision aids can be compiled from information sources and workflow artifacts that can confidently make claims about the I/O characteristics of a workflow. These decision aids in turn, can be used to optimize utilization and performance. To make adoption of this approach feasible, the architecture introduces a novel API and a new service. The Decision API allows applications and middleware the declaration of decision points and thus delegate a decision to exchangeable components. The Decision Aid Propagation Service (DAPS) simplifies the efficient collection and distribution of decision aids which may be generated in- or out-of-band.

The results demonstrate how workflow-specific optimization can be achieved with the proposed architecture. The results also demonstrate how breaking up an intractable end-to-end goal into smaller independent steps gives rise to a flexible architecture to discover and utilize insights about application and workflow behavior.

Zusammenfassung

Das rasche Entwicklungstempo zunehmend komplexer Systeme als Folge von technischen Durchbrüchen und Rechenzentrumsökonomie führt dazu, dass die händische Optimierung von Speichersystemen allein nicht länger praktikabel ist. Die nächste Generation von Rechensystemen erlaubt die Benutzung von adaptiven Heuristiken um Systeme optimal auszunutzen. Eine zentrale Herausforderung, dabei diese adaptiven Heuristiken zu verbessern, besteht darin, deren Entscheidungskomponenten mit den nötigen Informationen (Entscheidungshilfen/Decision Aids) zu versorgen. In vielen Fällen müssen diese Entscheidungen eine Balance zwischen der Absicht von Nutzern, der zutagetretenen Ein-/Ausgabeaktivität und der Betriebsperspektive vereinen. Dieser Begriff der Absicht ist in den Arbeitsabläufen (Workflows) von Nutzern und Wissenschaftlern eingebettet.

Diese Dissertation führt eine umfassende Analyse von wissenschaftlichen Workflows sowie Datenflüssen zu Speichersystemen durch, um Architekturverbesserungen zu entwickeln, die es bestehenden und zukünftigen Speichersystemen erlauben sich an wissenschaftliche Workflows anzupassen. Ein Kern dieser Architektur sind Entscheidungspunkte und Informationsquellen die sich über die unterschiedlichen Systeme eines Hochleistungsrechners verteilen. Für einen bestimmten Workflow lassen sich so verschiedene Entscheidungshilfen (Decision Aids) aus Workflowartefakten generieren, mit denen sich zuverlässige Aussagen über das I/O-Verhalten eines Workflows machen lassen. Diese Entscheidungshilfen können dann genutzt werden, um die Systemausnutzung zu optimieren. Um die Anwendung dieses Ansatzes praktikabel zu machen, führt die Architektur eine neue Programmierschnittstelle (API) und einen neuen Dienst ein. Die Entscheidungs-API dient dazu Anwendungen zu ermöglichen, Entscheidungspunkte zu deklarieren und damit Entscheidungen an austauschbare Komponenten zu delegieren. Der Decision Aid Propagations Dienst (DAPS) vereinfacht das effiziente Einsammeln und Verteilen von Entscheidungshilfen, welche in Echtzeit oder separat generiert werden können.

Die Ergebnisse zeigen, wie workflow-spezifische Optimierungen mit der vorgestellten Architektur erfolgreich umgesetzt werden können. Weiter zeigen die Ergebnisse, wie die Zerlegung einer großen Ende-zu-Ende-Zielsetzung in kleinere, teilweise unabhängige Werkzeuge eine flexible Architektur für die Entdeckung und Ausnutzung von Verhaltensweisen von wissenschaftlichen Anwendungen und Workflows ermöglicht.

Acknowledgments

It turns out the challenging part about writing a dissertation has little to do with how skilled in writing or how attentive a reader of literature one is. Writing a dissertation, in the end, is a social endeavor. I would not have been able to shape this into what it is without the support of so many people who shared their time, their advice, and expertise with me.
First and foremost, I am grateful for the interesting and stimulating environment I have found with the research group Scientific Computing of the University of Hamburg and the German Climate Computing Center (DKRZ) provided by my adviser Prof. Dr. Thomas Ludwig. When I joined the group, still being a bachelor student, I would have not anticipated to build a career and find so many friends along the way. Thanks in particular go to Michael Kuhn, Julian Kunkel, Hermann Lenhart, Panos Adamidis and Tobias Weigel for introducing me into the academic and institutional processes allowing me to gain experience in projects and conferences, but also for many private conversations with useful guiding advice.
To the people at Argonne National Laboratory (ANL), I am very grateful for getting the opportunity to experience the unique research environment which so effectively promotes interdisciplinary science. Here I would like to express special thanks to Philip Carns, Robert Ross, Shane Snyder, and Kevin Harms as well as Tom Peterka and Orcun Yildez. Your perspectives have been invaluable for me.
These acknowledgments could not be complete, without also thanking the colleagues from my research group at the University of Hamburg and DKRZ. In particular Kira Duwe, Anna Fuchs, Jannek Squar, Michaele Hensel, Merret Buurman, Claudia Martens, Heinrich Widmann as well as Ulf Garternicht, Carsten Beyer, and Kerstin Fieg.
Finally, I would like to thank my family and friends for their patience and unconditional support. I am sure you often would have picked more entertaining topics over make-shift lectures in computer science and high-performance computing if it were not for me. I am grateful for your constant nudges to get out of my comfort zone but also to seek balance between work and life.

Contents

1
Introduction

A brief introduction into why high-performance storage systems require decision support and why this means to make storage systems aware of scientific workflows. The chapter is structured as follows:

- *Section 1.1 starts with a brief overview of current challenges the High-Performance Computing community is facing.*
- *Section 1.2 adds a user perspective, the evolution, and recent influences on scientific workflows on supercomputers.*
- *Section 1.3 motivates the goals of the research addressed in the thesis.*
- *Section 1.4 outlines the structure of the manuscript.*

1.1 Challenges for High-Performance Computing

Supercomputers have become an indispensable tool for scientific and industrial users [Guest, 2012] around the world (see Figure 1.1). Various researching disciplines make extensive use of computing resources to conduct experiments and generate insight in areas that are otherwise too expensive, too dangerous or simply not possible with other currently available technology. Large-scale modeling, simulation, and analysis are being used to peek into the future and to understand phenomena that elude direct observation. Also on an individual level High-Performance Computing (HPC) is about to become more relevant, for example, in finding custom medical treatments.

Figure 1.1: A larger variant of this map is included in the Appendix: Distribution of the Top500 most powerful supercomputers in operation as of 2020. The circle radius corresponds to the rank, with the largest circle being the highest-ranking system. Where city information is unavailable the site is drawn in the center of the host country. Data: [Top500, 2019; OpenStreetMap Contributors, 2019; Mapbox Inc., 2019]

The ways scientists are using supercomputers has long shifted from a limited number of specialized applications to a complex orchestration of tasks and processing pipelines, which are commonly referred to as *workflows*. Compute and storage stacks, however, provide only limited capabilities to

exploit the intent expressed in these workflows. This thesis explores how to allow for workflow-sensitive decisions making throughout HPC storage stacks. It is cross-cutting research skirting many of the different challenges currently being faced in HPC. As such, the outlined approach is meant to be one puzzle piece in a much larger set of building blocks.
Starting with an overview the subsections 1.1.1 to 1.1.7 and Section 1.2 expand on each of the following observations and challenges which motivates exploring new and, more importantly, self-learning approaches to utilize the available storage and compute resources of a system:

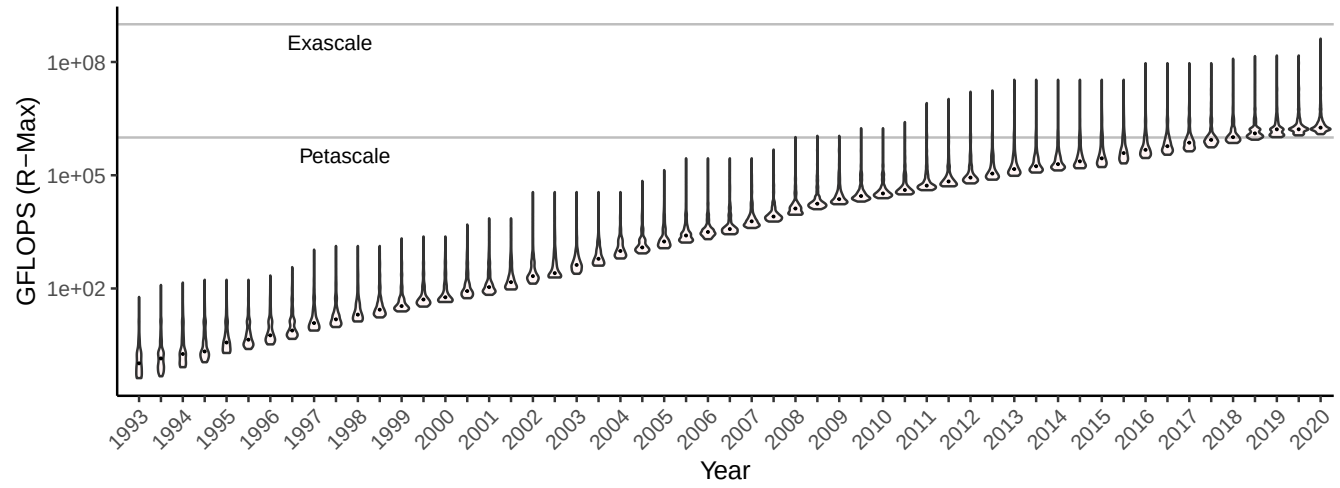
- Technological, economic and political constraints lead to increasingly complex system designs.
- Exascale systems are expected to face input/ouput (I/O) bottlenecks, resulting in more complex memory hierarchies and caches.
- Current heuristics in hardware and software are not flexible enough to address changing workload characteristics. Software-defined hardware provides more flexibility but also requires new paradigms.
- Systems are frequently decommissioned and replaced by newer machines, discouraging machine-specific optimization of applications as well as expert specialization.
- There is a lack of experts to address all the areas that deserve research and the scientific community is competing with industry for similar skills and qualifications.
- Communities, standards and software libraries require time to address changing circumstances and newer system designs.
- Co-design efforts promise highly efficient designs, but only a limited number of well-funded ambitions can afford to take this path.

Finally:

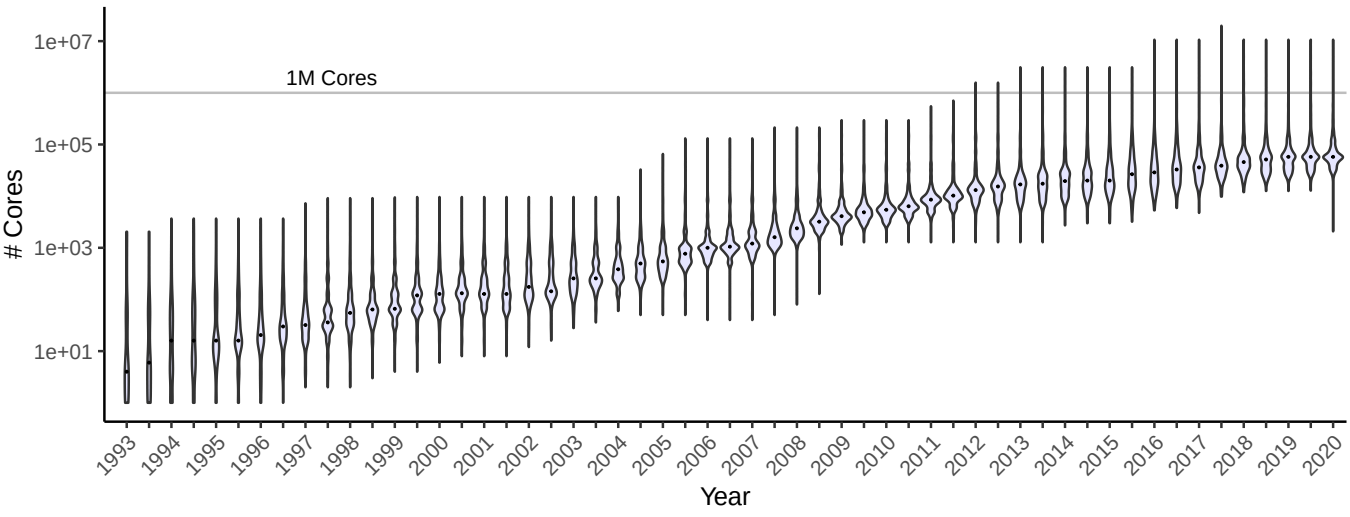
- Workflow definitions, domain-specific languages, and monitoring information expose application details and user intent, creating new opportunities for compile and runtime optimizations for data handling.

1.1.1 Boundaries: Memory, Money and Energy

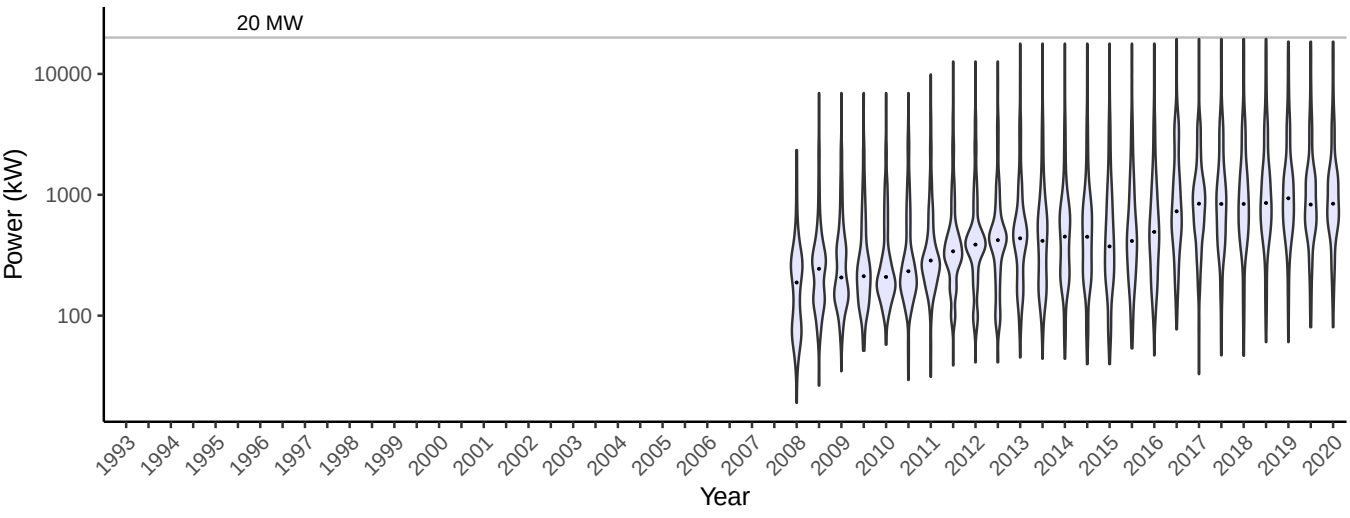
Processor, memory and network technologies advance on divergent trajectories. Clock frequencies did not increase notably for years, and even Moore's law is slowing as the technology approaches economical and theoretical limits [ITRS, 2015]. Yet, compute capabilities continue to rise dramatically due to massive use of parallelism and distributed computing, as illustrated in Figures 1.2a and 1.2b plotting the development of peak performance and processor counts in the Top500 Supercomputer list between 1993 and 2020) [Top500, 2019].
Unfortunately, memory and storage technologies did not benefit from comparable advancements. Of roughly a million floating-point operations, only a single result can be permanently stored. This mismatch is sometimes referred to as the memory wall [McKee, 2004], which holds the implication that researchers have to decide which information is considered valuable enough for preservation.



(a)



(b)



(c)

Figure 1.2: Development of different key metrics in the Top500 Supercomputing list between 1993 and 2020. The violin plots visualize how the 500 systems of each biannual list are distributed. The dot indicates the median. Data: [Top500, 2019]

Besides the memory/storage challenge, policy and practicality demand to limit the next-generation (exascale) systems to stay within a 20MW power envelope [Hruska, 2012]. As of 2019, it is clear that the first generation of exascale systems will exceed this target by about a factor of two. Since 2008 the Top500 (see Figure 1.2c) includes power consumption for many of the listed systems. In 2016 the US Department of Energy released estimates that data centers accounted for 1.8% of the country's total electricity usage [Shehabi et al., 2016].
While adding additional storage hardware promises the realization of arbitrary aggregated throughput performance, seeking parity to compute capabilities with currently available technologies would result in systems that are dominated by ever-larger storage systems which eventually would exceed power limits and available budgets.

1.1.2 *Memory, Storage and Caching*

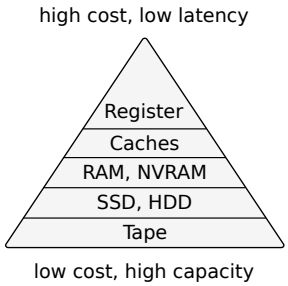


Figure 1.3: A simplified overview of the memory hierarchy, commonly found in data centers.

Memory and storage technologies feature wildly different performance characteristics, discussed in more detail in Section 2.3 (Characteristics of Low-Level Storage Technologies). In summary, it is usually necessary to find a trade-off between latency, capacity, and cost. Lower latency technologies tend to have lower capacity, lack persistency and are more costly. As a result memory hierarchies (see Figure 1.3) are employed in combination with caching techniques to provide fast access to data that is currently in use, while allowing to store large amounts of data on much more affordable storage media.
Data centers may employ additional hierarchies, but in general, they operate similarly to usual computers: the bulk of data is stored and archived on cheap high-capacity but high latency storage media such as magnetic tape, while data with fast access requirements is staged onto lower latency storage tiers based on disks or solid-state flash storage.

1.1.3 *Inflexible Heuristics and Hardware*

To exploit the available resources most systems heavily rely on heuristics and policies responding to performance indicators that have to perform reasonably in a wide range of circumstances. Consequently, these heuristics are often sub-optimal for specialized applications. Many optimization efforts revolve around adapting applications to play well with these heuristics, which even can be manifested in hardware. For example, on the chip level, the cache hierarchies are beyond the control of an application or even the operating system. Applications can only try to find access patterns that minimize the displacement of cached data which will be accessed multiple times. Similarly, distributed computing is enabled by network technologies, but current supercomputers, once wired up, are not very flexible to adapt to changing communication patterns. Research in software-defined hardware is gaining momentum driven by the demand of cloud environments and by affordable Field Programmable Gate Arrays (FPGAs) in reconfigurable networking [Zilberman et al., 2015]. In the future neuromorphic computing/network hardware may be available, which are incompatible with traditional programming paradigms and algorithms.

1.1.4 Complexity vs. Supercomputer Lifetime

The downside to all these approaches is an increase in complexity and innumerable modes to operate HPC systems. Unfortunately, in many cases, we still do not have a good understanding of what is going on with current systems. This especially applies to the interactions between multiple subsystems. Vendors increasingly address the lack of monitoring capabilities in their products as they are widely demanded by the community and indispensable to diagnose and better troubleshoot problems. But due to a limited availability of experts, even a wealth of telemetry data won't necessarily boost innovation, leaving HPC users with powerful systems without the necessary tools to better exploit them.
Manually optimizing applications and systems is not a desirable solution because optimizations that work for one system are not necessarily portable to another. On the one hand, for research at this scale to be reproducible, it must be possible for independent researchers to run applications on other HPC sites. On the other hand, HPC systems age relatively quickly: Reduced power consumption of newer more powerful hardware mixed with more expensive vendor support prohibit economical operation of older hardware despite the huge initial investments. As a consequence, many HPC systems are exchanged after about four to five years. Unfortunately, it is hard to find exact statistics on the true lifetime and dates when Top500 systems are decommissioned. However, it is possible to approximate the system "lifetime" by counting the number of years a system remained listed in the Top500 as is plotted in Figure 1.4. More recently, especially large installations remain operational for longer or are are expanded.

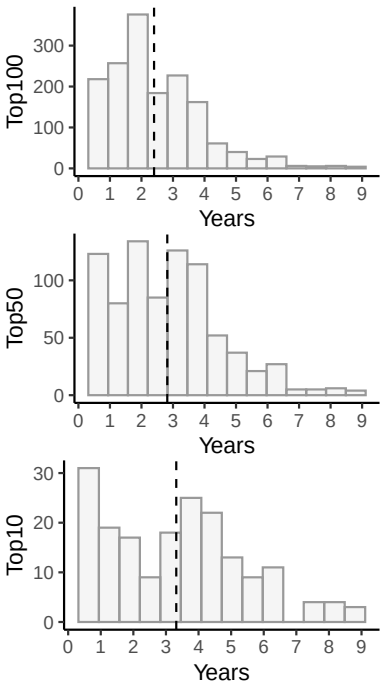


Figure 1.4: Approximation of the mean "lifetime" (dashed line) of supercomputers in the Top500 by considering systems which entered the list as a top 100/50/10 system between 1993 and 2020 and counting the years they remained in the Top500. Data: [Top500, 2019]

1.1.5 Community Efforts, Libraries and Middleware

An important role, limiting the effort required to adapt existing applications to new systems, is fulfilled by standard interfaces, specialized software libraries, and middleware that implement efficient algorithms for common problems. Various such libraries and standards exist [Donnell, 2016], addressing everything from arithmetic like BLAS [Lawson et al., 1979], Intel MKL [Intel Software, 2019], FFTW [Frigo and Johnson, 2005], Trilinos [Heroux et al., 2005] or PETSc [Balay et al., 2019] to concurrency control and inter-process communication such as OpenMP [OpenMP, 1997] and the Message Passing Interface (MPI) [MPI Forum, 2019]. This saves application developers time and also frees them from the burden to ensure portability and correctness in distributed environments.
Figure 1.5 illustrates this hierarchy of software layers utilized by most applications with a focus on libraries related to handling input and output (I/O) to storage systems. In scientific contexts data description frameworks such as HDF5 [HDF Group, 2019] or NetCDF [Rew and Davis, 1990; Unidata, 2019] are popular to allow effortless exchange between scientists. Unfortunately, over time these libraries accumulate legacy. For example, in HDF5 this becomes apparent in optimizations designed for storage systems that differ considerably from today's distributed storage systems. Because HDF5 in the past could not and today does not pass on logical information about the structure of the data to lower levels, there are few ways to account for it. Eventually, well-intentioned heuristics distributed across different layers can impede each other.

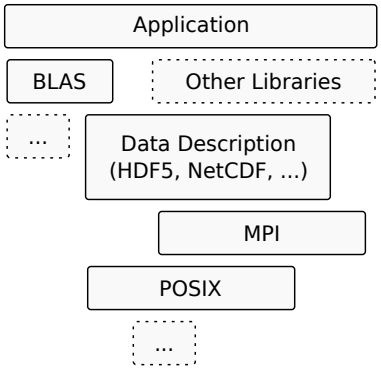


Figure 1.5: Applications commonly rely on third-party libraries for often used functionality and data structures.

1.1.6 Compilers and Domain-Specific Languages

Sophisticated optimizations, however, are already routinely applied by compilers. Unfortunately, even modern compilers generally cannot understand a programmer's high-level intentions from source code written in a general-purpose programming language. Domain-specific languages provide programmers a convenient option to express these intentions while still using a formal description language. In addition, a domain-specific language can specify high-level concepts such as data structures and procedures to manipulate data. In many cases, these data structures and manipulations are associated with constraints that are not expressed when using a general-purpose programming language, but that are defined along with the specification of a domain-specific language. This information is usually used to improve the compute performance at runtime, but the additional information about the structure of the data and potential data manipulations can also benefit I/O systems. Two important choices, for example, are choosing how to serialize and distribute data. In particular, the knowledge of workflows enables the exploitation of data locality.

1.1.7 Specialization and Co-Design

Incremental changes to existing solutions appear to be insufficient to address the challenges ahead. Various efforts, therefore, include all stakeholders and technical layers, which is often referred to as *co-design* [Segers and Nominé, 2017]. A major driver in HPC innovation has always been the US Department of Energy (DOE) which focuses on two [Ang, 2015] approaches to co-design: 1) application-centric and 2) architecture-centric co-design. As exascale systems are approaching and the storage challenge intensifies, multiple efforts (including FastForward [Intel et al., 2014], ADIOS [ORNL, 2017], HDF VOL [Chaarawi and Koziol, 2014], NEXTGenIO [NEXTGenIO, 2018]) are working on the modernization of how applications and libraries down to the storage hardware handle I/O. Co-design may yield highly optimized solutions for special use cases but generally is not affordable for more casual users. Although, efforts such as PARADISE[1] [Michelogiannakis et al., 2019] as well as technologies bridging bare-metal and programmable hardware, such as FPGAs, promise to liberate the tools and manufacturing processes required to enable the community at large to consider specialized tailor-made hardware.

[1] An open hierarchical hardware simulation suite to explore post-Moore architectures.

From the software side, an important driver enabling specialization and the adoption of co-design strategies stems from efforts to move a lot of functionality traditionally offered by operating systems into the userspace. This removes overheads from context-switches and encourages efforts developing various building blocks to offer and connect compute, network, and data services flexibly. Various national and international initiatives such as the US Exascale Computing Project (ECP) [DOE and NISA, 2017] or the Partnership for Advanced Computing in Europe (PRACE) [Guest, 2012] attempt to coordinate these efforts and to offer training to the broader community.

1.2 Computational Sciences & Workflows

The aforementioned efforts to a large extent serve the purpose of advancing HPC as a tool for scientific discovery. Virtually every scientific domain and

many industries have adopted methods from the computational sciences including fields like quantum mechanics, climate research, weather forecasting, oil and gas exploration, molecular dynamics, and other physical simulations.
In modern computational sciences, many of the repetitive duties are automated and researchers find a rich ecosystem of software tools and frameworks with different algorithms to approach a problem and conduct their research. Yet, to allow sharing a supercomputer, workloads are usually still broken up into individual jobs which can be submitted to a batch scheduling system.
This notion of compute jobs, however, is ceasing to be adequate as the compute landscape diversifies. Increasingly industrial and commercial users are relying on data-driven decision-making and big data technologies. These users, as well as scientific users who traditionally have not been users of supercomputers, are often first turning to cloud computing. The provisioning model and the financial incentives of cloud computing have led to advancements in container technology, virtualization, and automation. As a result hardware used in cloud environments has become a commodity impacting HPC data center designs. The result is a convergence of HPC, big data, and cloud technologies and national and international efforts to provide cloud-like science infrastructure [UCSC CGL, 2017].
The measured service model commonly used by cloud providers encourages users to plan ahead which essentially boils down to explicitly defining and automating their workflows.
A second influence stems from the fact that scientific funding often mandates a typical template for how to organize a campaign to generate insight when using high-performance computing platforms. A campaign will often consist of an exploration phase, a data generation/experiment/simulation phase and an analysis phase as illustrated in Figure 1.6. Any of these might rely on HPC resources to perform sub-tasks such as:

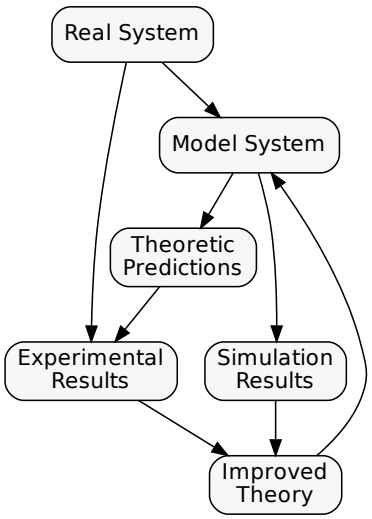


Figure 1.6: An abstract perspective on how the interplay between experiments, simulations and theory advances scientific understanding which then informs optimizations to real systems.

- Processing and analysis of measurement data (e.g., particle accelerators, remote-sensing, or other statistical analysis)
- Simulation of (physical) systems using mathematical models, such as fluid dynamics or molecular dynamics.
- Optimization problems and machine learning, with inverse problems and deep learning being especially demanding.

A large burden for users of HPC systems is to take advantage of these intrinsically concurrent and distributed systems. Another burden are increasing data volumes that routinely need to be handled. Partly this is a result of simulation resolutions being increased to avoid having to parameterize model dynamics. In addition, interdisciplinary models require adding additional variables to be computed and stored. To cope with uncertainties researchers are also increasing the number of scenarios that are simulated leading to a large number of additional datasets. Measurement instruments are becoming more detailed but also far more abundant. This shows in remote sensing where newer missions feature more instruments and higher resolution per satellite but also in larger constellations, with sometimes up to several hundred satellites [Marshall, 2017]. Similarly, the availability of cellular networks, low-power communication and compute chips as well as low-cost sensors enable unseen amounts of collected data, parts of

which need to be aggregated at the edge. Large-scale microscopes, particle detectors, and telescopes are already very selective about which data they permanently store, often turning to machine learning pipelines to predict which measurements might be the most valuable, as it is not affordable to store all information.
An untapped potential for increasing system utilization and contextual understanding of I/O behavior exists by exploiting knowledge about the structure of HPC workflows.

1.3 *Decision Support for Workflow-Aware High-Performance Storage Systems*

The previous sections listed limitations and challenges of traditional approaches to storage performance exploitation. Section 1.2 (Computational Sciences & Workflows) briefly illustrated the changing nature of scientific workflows as far as HPC is concerned. Finally, a number of promising trends that enable more flexible and sophisticated decision-making have been outlined. Unfortunately, current storage stacks are not fit to benefit from recent developments in machine learning and more flexible systems. This thesis introduces an architecture proposal and a number of recommended changes to the existing storage stack to allow gradually adding adaptive decision-making capabilities from a workflow perspective.

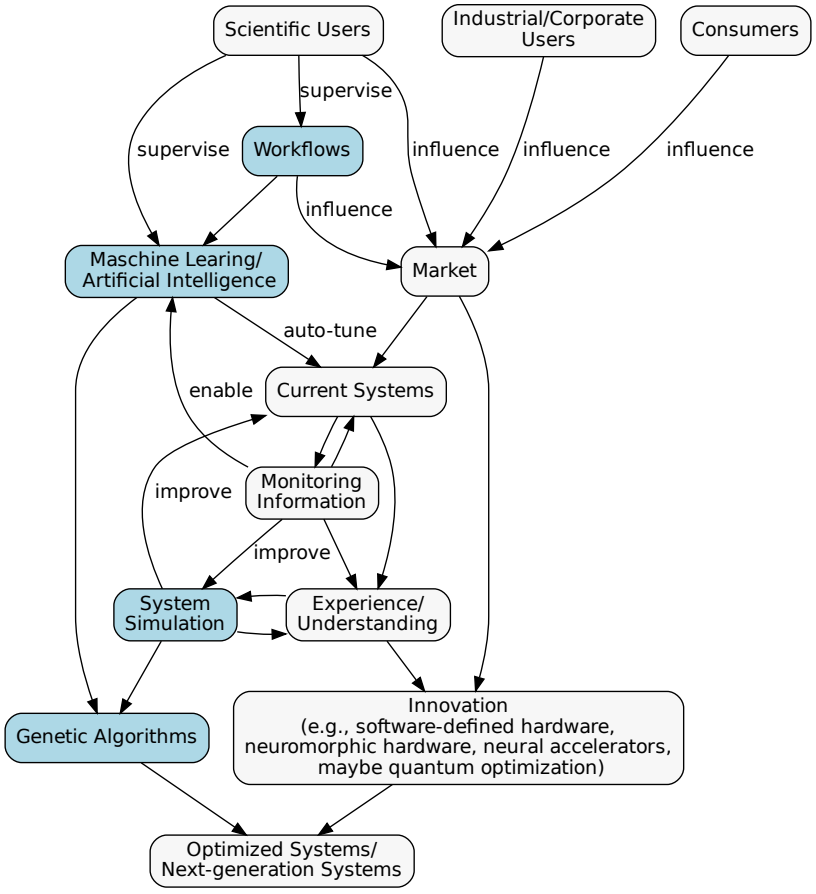


Figure 1.7: Existing and trending influences on next-generation systems. Scientific users are only a small niche market in comparison to industrial users and consumers. Yet, scientific requirements and new innovation affect next-generation systems. Recently, breakthroughs in machine learning promise to enable more ambitious efforts to automatically tune existing systems allowing to offset the lack of experts to improve performance exploitation. Simulated sandboxes may become essential to provide machine learning approaches with sufficient experiences for training and enable rapid evolution, for example, using genetic algorithms.

Technological advancements, simple heuristics, standards, libraries and co-design remain a driving force to better system exploitation. But the impact of efficient software implementations and hardware components could be multiplied by self-learning workflow-aware algorithms and heuristics.

While multiple efforts explore machine learning for workload characterization and anomaly detection for HPC environments, the activities in this field are still in their infancy and wide-spread deployment in production systems is a distant vision.
This research consequently explores the intersection where scientific workflows, storage systems and machine learning methods meet. Key drivers to seek more automation are the fast innovation life-cycle of supercomputers, increasing complexity and the lack of experts. Figure 1.7 highlights four key technologies which are currently not well integrated into storage systems and which promise to be useful to optimize and automate existing systems from a workflow perspective:

- Workflows offer a description of user intent and an anticipateable future. A workflow will be declared explicitly to a certain extent, but other relevant information such as performance implication of workflow activity won't be. As they will vary from system to system and use case to use case, these ought to be learned.
- Learning methods allow to automate adaptation and to train detectors and mappings which are traditionally hard to program explicitly.
- System simulation allows providing machine learning algorithms with the sufficient "experience" to converge towards effectively solving their task and to test them before deployment.
- Finally, from a long-term perspective the combination of genetic algorithms, simulation and learning methods has shown to be useful in advancing existing and discovering new approaches in other fields.

As this outlines a larger vision in part beyond the scope of this thesis, the research focuses on the consequent first step of exploring the technical changes required to gradually employ workflow-aware decision components throughout the storage stack. The scientific contribution of the thesis can be summarized as follows:

- A survey of information sources and decision components in the storage stack relevant from a workflow perspective.
- An architecture proposal to gradually roll out feedback loops and workflow-adaptivity into existing applications, I/O middleware and storage systems.
- Tools to better explore workflow I/O in HPC systems and to make I/O research more reproducible.
- Case studies and performance evaluation for a number of workflow related performance optimizations and decisions aids considering real-world as well as synthetic examples.

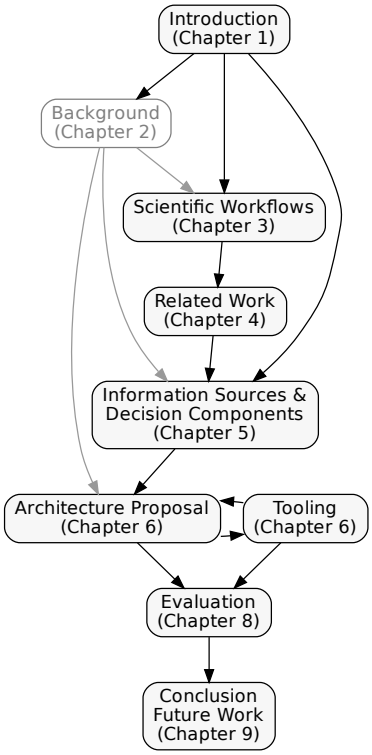


Figure 1.8: Relationship between the chapters of the manuscript. Background, actually being in support of all the other chapters, is greyed out.

1.4 Structure of the Manuscript

The dissertation is broken up into eight chapters with an overview of their relationship illustrated in Figure 1.8. The chapters in order are:

Background: Chapter 2 serves as an introduction of important concepts and a collection of methodologies used throughout the thesis.

Scientific Workflows in HPC: Chapter 3 collects the requirements of scientific workflows from an application and from a technical perspective. In addition, a workflow definition and use cases with a focus on the storage perspective are discussed.

Related Work: Related work focusing on storage-awareness in workflow systems, workflow-awareness in storage systems, telemetry capture and approaches for automatic decision-making in HPC are discussed in Chapter 4.

Information Sources & Decision Components: Chapter 5 explores which information can be tapped to aggregate workflow related decision aids on the one hand, and identify where storage systems are employing decision which would benefit from not being ignorant of workflows, on the other.

Architecture Design & Tooling: Combining the information collected, Chapter 6 introduces an architecture proposal to distil and distribute workflow related decision aids to heuristics in applications, I/O middleware, or the storage systems themselves.

Case Studies & Evaluation: Chapter 7 evaluates 1) the feasibility of the proposed system by demonstrating that workflows offer opportunities for optimization on the one hand, 2) how decision aids can be derived from real workflow and systems data and 3) how workflow-aware performance optimizations can be realized in HPC storage systems.

Summary & Outlook: Chapter 8 summarizes the results and contributions of the thesis and provides an outlook on future work.

2
Background

This chapter introduces nomenclature as well as technical and methodological background as it is required in the remainder of the manuscript. The chapter breaks down into the following sections:

- *Section 2.1 to Section 2.3 introduces the building blocks and most important concepts for high-performance computing and storage systems.*
- *Section 2.4 introduces higher level storage abstractions and data formats commonly used in the context of scientific computing.*
- *Section 2.5 introduces useful data structures and algorithms from graph theory used mostly for modeling and representation of relationships.*
- *Section 2.6 offers brief descriptions and intuition for machine learning methods and mathematical concepts used mostly in the evaluation.*

2.1 A Typical HPC Facility

The scale of supercomputers often demands building dedicated data centers to accommodate them. Data centers are complex facilities housing a variety of different installations. Figure 2.1 approximates the structure of a typical HPC data center. While compute sites differ considerably in details, the following subsystems are commonly found at supercomputing sites:

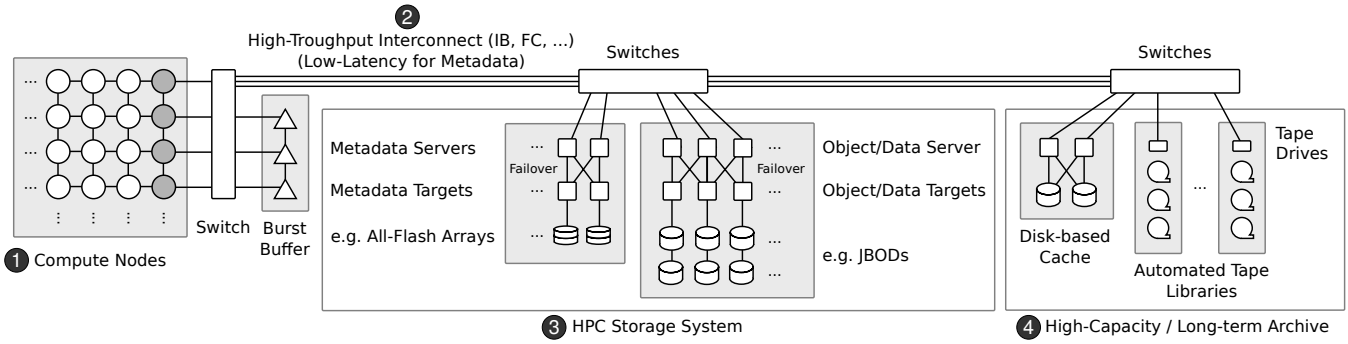


Figure 2.1: A data center with a typical HPC storage system and a tape archive. The network is only an approximation for better overview and may differ in a real deployment.

- A high-performance *compute cluster* ❶ to perform calculations.
- A high-performance *network* ❷ for compute nodes to communicate with each other as well as to access storage systems.
- A high-performance *storage system* ❸ usually based on hard disks and SSDs for regularly accessed data.
- A high-capacity tape system for data which is accessed less frequently or which needs to be preserved in a *long-term archive* ❹.

Newer data centers may also come with additional clusters specialized for different use cases and additional storage systems. As such, it is increasingly common to also find:

- Smaller dedicated clusters optimized for pre- and post-processing, visualization, or machine learning.
- Burst buffers, to quickly absorb data generated by physics simulations or measurement instruments.
- A virtualized environment or a cloud for smaller services which need to be co-located. For example, to host services used to promote and exchange data globally between researchers and institutions.

The following subsections will characterize these systems, with a primary focus on storage systems that are covered in particular detail.

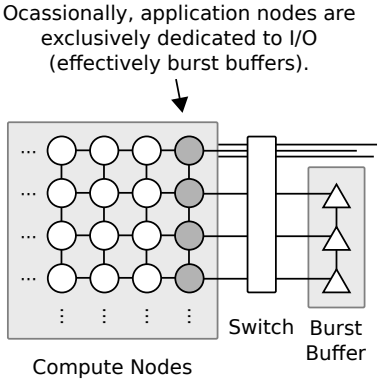


Figure 2.2: Illustration of a typical *compute cluster* with many compute nodes tightly connected using a high-speed interconnect. Often compute nodes will only provide minimal storage to hold the operating system, although deployments with storage distributed across compute nodes exist. Many checkpointing applications, such as physical simulations, turn to two-phase I/O which might use dedicated compute nodes or burst buffers to collect data before asynchronously draining checkpoints to lower tier persistent storage.

2.1.1 *Compute Cluster*

At the core of a supercomputer is usually a compute cluster. Up to millions of compute cores and nodes are connected often using a low latency interconnect such as Infiniband as illustrated in Figure 2.2 The compute system is used to process and generate experimental or model data. A compute node typically has at least a multi-core CPU but co-processors such as general-purpose GPUs are becoming increasingly popular especially due to machine learning workloads. An increasing percentage of Floating-Point Operations Per Second (FLOPS) in modern systems has to be attributed to the proliferation of these accelerators.
Typically, compute clusters are fairly *homogeneous*, as the configuration for all the compute nodes is the same or similar. But as HPC sites attract users from different domains, it is increasingly common to see HPC sites offering a variety of different compute partitions. On the one hand, this gives users more options to allocate nodes with different properties, for example, extra memory, extra storage, or additional accelerators, but on the other hand developing for these more *heterogeneous* systems can also become an implementation burden.

2.1.2 *Network or Interconnect*

The network connects compute nodes with each other and allows multiple compute nodes to coordinate and to exchange data. In addition, the network integrates the storage systems and other services. There is a variety of different network technologies with different properties. As the thesis is focused on storage I/O and not on network I/O, only high-level concepts are introduced for network I/O, especially as storage I/O is often constrained mainly by device performance and only to a lesser extent by network performance. When characterizing networks relevant factors are the network topology, the network bisection-bandwidth, latency, and hop counts as well as throughput/bandwidth.

Network topology: describes the structure of the underlying physical connections. The network topology is made up of links (cables or fibers) and switches. Common topologies deployed in HPC data centers include *fat-tree*, *multi-dimensional tori*, *hyper-cubes* and *dragonfly* networks, with the latter

offering similar properties to hyper-cubes while requiring fewer links. As many topologies are organized hierarchically, metrics such as the bisection-bandwidths and hop counts are considered to determine throughput and latency.

Latency: Most generally, the delay for a certain operation to complete. Depending on the context, latency can have slightly different meanings. In a synchronous context, for example, when using the ping utility, a latency measure often refers to the so-called round-trip-time, while asynchronous messages would typically consider only the time it takes to process and transmit (single-trip) a message. Typically, latency is measured in milliseconds, although low latency networks are operating in a range of 0.5 to 5 microseconds.

Throughput: determines how much data can be moved from one point to another within a certain timeframe. Usually, this is measured in bits or bytes per second, although it is also common to find measurements provided in bytes/sec, for example, 200 Gb/s or 200 Gbps. The lower case *b* will typically be used for bits while the upper case *B* indicates a byte value.

2.1.3 Hierarchical Storage Management

A hierarchical storage system (HSS) also hierarchical storage management (HSM) integrates the different storage *tiers* that will be introduced in the following into a single system. While many different naming schemes for tiering depending on the specialization exist, the discussion here focuses the following three most relevant from an HPC perspective: *distributed HPC storage systems*, *long-term archives*, and *burst buffers*.
By hierarchically organizing storage systems made up of many different technologies with different performance and cost characteristics, they can be exposed as a single system while hiding complexity from users. Somewhat against unintuition, systems hiding such complexity are often referred to as being *transparent* which in this context should be understood as the action or complexity being transparent and thus being invisible to the user. Occasionally, HSM systems are also used to integrate legacy storage systems with new installations.
An HSM system can relocate user data transparently from one storage tier to another, this relocation is typically referred to as data staging. For example, by monitoring how frequently data is accessed, data can be moved from a scratch file system to the archive to conserve energy and safe cost or to make room for other data. However, in practice HSM storage solutions are not trivial to set up, as coming up with and implementing appropriate data movement policies is a complex task. Many theoretically possibly optimizations also remain unused as insight into the workflow, application, or user behavior is often not readily available.

Hot & Cold Data or Data Heat/Temperature: A common policy for data staging decisions to be based on is a concept that is referred to as *heat* or temperature of data. Data is *hot* when it was recently used, and hot data is usually staged onto a high-performing tier. Less frequently accessed data is considered as merely being*warm* or *cold* if it is rarely accessed at all. In the simplest case a HSM might just consider hot and cold data, but the heat

value can also be continuous: Every access may increase a counter which decays as time passes. The temperature analogy is especially popular in hierarchical storage systems but can also extend to in-memory caches.

2.1.4 *Distributed High-Performance Storage Systems*

As large amounts of data are processed and generated, storage systems with matching performance characteristics are required. To keep up with the requirements of modern compute clusters, storage systems are distributed systems themselves, although this fact is typically hidden to users who only interact with higher-level storage APIs.
The core challenges to overcome are bandwidth limitation as well as finding a compromise to cost/latency/capacity dynamics of these technologies (compare Figure 2.3). Storage systems achieve this by combining a variety of different strategies. Storage is clustered into arrays that are exposed as logical devices, allowing to aggregate capacity but also increase throughput by allowing parallel access.
Latency currently can only be reduced by employing more expensive technologies, although typically a relatively small amount of low latency memory or storage used as a cache layer can allow to provide the bulk of capacity through a slower but higher capacity technology. Another strategy is the discrimination of data and metadata, thus reacting to the workload characteristic of different kinds of data. Two particularly important storage technologies are *(parallel) file systems* and increasingly *object storage*. For both kinds of storage systems, it is common to find physical architectures similar to the one illustrated Figure 2.4 which features I/O servers that expose storage services to clients, while fail-over and device aggregation and low-level data distribution being handled transparently.

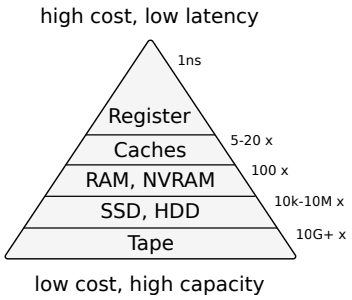


Figure 2.3: A simplified overview of the memory hierarchy, commonly found in data centers.

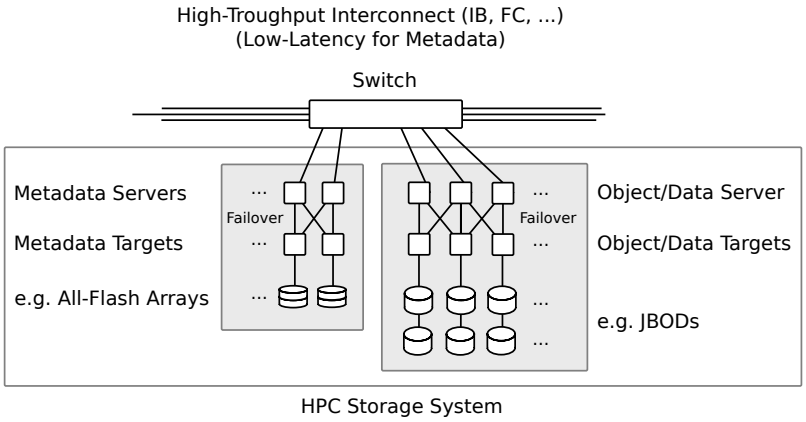


Figure 2.4: A data center with a typical HPC *storage system*. The network is only an approximation for better overview and may differ in a real deployment. This example separates data and metadata physically, but there is also file systems which provide more *converged* packages that combine data and metadata on, for example, a rack to rack basis.

Parallel File Systems (PFS) offer a hierarchical namespace and usually allow to nest files in a directory structure while keeping metadata such as modification and access times as well as permissions to allow/restrict users or groups from accessing the data. In contrast to regular file systems, a parallel file system is designed to take advantage of multiple storage devices to increase capacity and performance. To support legacy software many file systems aim to conform to the POSIX standard, which generously guarantees certain semantic behavior. Unfortunately, some of these guarantees confront designers of parallel distributed storage systems with immense challenges:

- The POSIX standard, for which development started 1988 during a time

which assumed a single computer, guarantees writes to become immediately effective to all readers. While this is a convenience for application developers, it is expensive and wastes precious resources spent propagating and establishing a consistent state in a distributed system.

- Offering access and modification timestamps is problematic for two reasons. On the one hand, it turns read access effectively into a read-modify-write operation, which is more expensive than a true read-only activity. On the other hand, in cluster environments thousands of clients might access the same file, for example on job start, effectively flooding the responsible metadata service with requests to update the timestamp, even though the update event usually holds little practical value.

Object Storage or Object Stores in contrast to a file system, is gaining popularity for being cost-efficient and more flexible for applications to exploit performance. Object storage provides a simpler interface to store and access blobs of data only, which get issued only an identifier often called a universally unique identifier (UUID). All additional data management functionality has to be performed by the application as object storage typically does not provide features to support user metadata. A common example might be functionality to retain the UUID in an index or directory services for discover-ability. Different object storage solutions offer different access granularity, some solutions always access the whole object but numerous solutions support byte-addressability within objects. This is relevant, for example, when a dataset is mapped onto an object and a user would like to access only a subset. Most distributed object stores would also use striping to enable parallel access for large objects.

2.1.5 *High-Capacity Storage and Long-term Archives*

Scientists commonly will have data that should be retained but does not need to be kept accessible on short notice. Disk and SSD-based systems become expensive to procure and operate when large data capacities are required. For cold data that is accessed infrequently or which is archived, it is not economical to rely on an online system. Storage media sued in archives are often assumed to be Write-Once-Read-Many (WORM). The term is ambiguous to a degree because it can be relevant in terms of access semantics which can be useful when considering optimizations, but it is also sometimes used to describe a legal requirement where media are temper proof. Among the most common technologies for long-term archives employed by data centers are tape libraries to offer data tiers for cold data or for long-time archives which also would add redundancy and off-site backups. To improve the quality of service for users, modern tape systems are often deployed together with a disk-based cache as is illustrated in Figure 2.5.

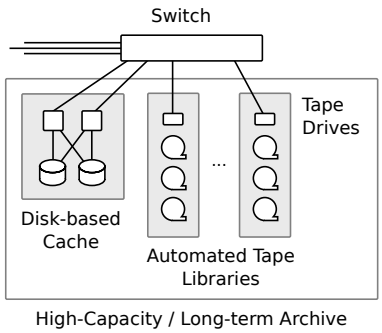


Figure 2.5: A data center with a typical *long-term archive* for cold data. Often robotized tape libraries with a number of tape drives are used together with a disc-based cache. The network is only an approximation for better overview and may differ in a real deployment.

2.1.6 *Burst Buffers*

A relatively new addition in the family of storage solutions, though insufficiently supported or too expensive to be employed exclusively, are *burst buffers*. Burst Buffers provide a limited capacity made of fast storage technologies. The capacity often will be proportional to the main memory of the compute cluster as the burst buffer is used to quickly flush the state of a simulation. Burst buffers allow to snapshot at a higher frequency than

would be possible using traditional high-performance storage systems, they are however also more expensive.
Initially, burst buffers were built around SSDs but more recently also integrate non-volatile memory (NVRAM). Burst buffers are an active area of research and different vendors are experimenting with different models. It remains to be seen which deployment options work best. Typically, burst buffers provide an API that allows to allocate and release high-speed storage resources that can be used by resource managers and batch scheduling systems to stage and unstage data.
Burst Buffers are also at the core of how the deployment of storage systems might change in the future. In the past, software complexity and sufficiently fast networks to move data between compute and storage systems worked in favor of designs that would use *pooled* deployments which would separate compute and storage systems also physically. With NVRAM and Burst Buffers *converged* deployments might become more feasible which would mix compute and storage systems. The term *hyper-converged* is sometimes used to indicate that compute, network and storage would be assumed to be mostly virtualized. Both pooled and converged modes as well as hybrid approaches can be adequate allowing to compromise based on the use case.

2.1.7 Scaling Compute and Storage Systems

When scaling compute and storage systems, a common practice is to deploy pools of nodes that are structured very similarly. This section takes a closer look at a few typical configuration choices for *compute nodes* and *I/O nodes*, and how this affects the cost and the performance of a system.

Compute Node configuration depends on the tasks most commonly performed. Many simulations demand a combination of computing power and memory in combination with a low-latency network. Other use cases, such as visualization or increasingly big data and machine learning applications, may be less dependent on synchronous communication but make use of accelerators. CPUs, GPUs, and potentially FPGAs are the determining factor for power consumption and how fast data can be processed. The memory capacity affects the problem size that can be held for quick access. Memory is usually contended, for example, by storage and network components which require memory for buffers and caching. Nodes may feature storage, which is local to the node, although node-local storage is considered too slow for large amounts of data in comparison to PFS/Object Storage. There is potential for this to change as node-local NVRAM and burst buffers become more affordable. Network interface cards (NICs) of a node determine how fast nodes communicate with each other, but also how fast data can be drained away from the compute nodes, for example when writing snapshots. The network also affects how quick a compute node will start to perform useful work if datasets or shared libraries need to be loaded.

I/O Nodes also vary depending on their specialization to handle different request types. In I/O nodes processors determine the number of requests that can be handled by a single node. I/O nodes often feature a lot of memory for use as a caching layer. But most importantly, they accommodate or connect to a large number of disks or SSDs over which data is strided for fault tolerance and higher performance. In many cases, the storage devices are bundled into so-called Just A Bunch of Disks (JBODs) which then are con-

nected to the I/O nodes, while an I/O node itself has no storage devices. I/O nodes in HPC systems, commonly use advanced interconnects in failover configurations. Which interconnect is favorable depends on whether target applications are leaning toward high-bandwidth for data storage or low-latency interconnects for metadata access.

- *Metadata Handlers and Targets:* Parallel file systems often provide dedicated metadata targets optimized to perform many I/O operations. Metadata servers commonly utilize different storage media than data targets. For example, they often have faster and more expensive solid-state disks and may be candidates for storage-class memory (SCM).
- *Data Handlers and Targets:* Data targets are configured for capacity and high throughput. Data targets may feature a lot of memory for caching but the cost is dominated by the number of hard drives. For HPC systems, usually larger reads and writes are observed, for database systems also the data targets might profit substantially from the usage of SSDs. RAID controllers can also be a cost factor but software-based RAID is becoming very popular for being more flexible.

2.2 *Telemetry & Monitoring for Compute and Storage Clusters*

As compute and storage clusters are distributed systems with many components, there is a lot of potential for components to fail or become overloaded. To cope with this, distributed systems commonly establish special *monitoring* services that ensure if all systems are operational and notify operators or administrators in case they are not. To provide this monitoring service the individual sub-components need to be *instrumented* to collect relevant *metrics*, which are then exposed as *telemetry* accessible from a distance.

Instrumentation: The ability of gathering system or application statistics and events is often referred to as *instrumentation*, but instead of a physical measurement instrument/sensor/probe a software equivalent is used. Often different software solutions settle on different terms or use terms synonymously that are differentiated by other software. Commonly distinguished instrumentation activities include the production of summary statistics as in *profiling*, the process of *timing* to establish latency of an action, as well as the recording of events over time as is done in *tracing*.

(Performance) Counters, Timers and other Metrics: Instrumentation often relies on devices that store a time reference or the number of times a particular event occurred, these devices are called *timers* and *counters*. On the lowest level, hardware such as CPUs, GPUs, and NICs may also offer hardware counters. A level up, operating systems establish various virtual counters, for example, to keep track of CPU-, memory-, and network utilization, as well as process behavior. Finally, an application, a middleware, or service may also define it's own virtual counters and *metrics*.

(System) Telemetry: Because a monitoring service for a distributed system cannot be co-located with all its components, it gathers information throughout the system to compile an overview. The collection of remote measurements and their automatic transmission is usually referred to as *telemetry*. Chapter 5 discusses many examples for telemetry and other information that are useful from a workflow I/O perspective.

2.3 *Characteristics of Low-Level Storage Technologies*

Storage systems depend on lower-level building blocks that exploit structural properties or physical phenomena to retain data for a period of time. This section serves as a reference for a few of the key performance characteristics of storage and memory technologies widely available on the market.

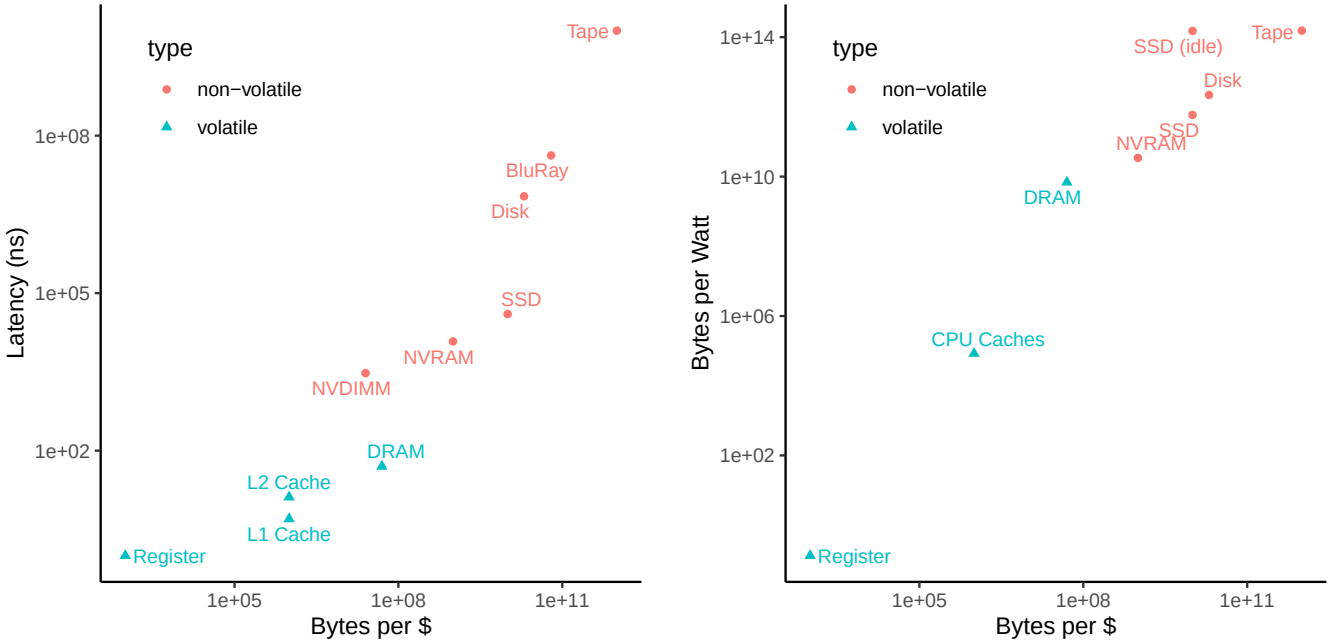


Figure 2.6: Comparison of capacity, latency and power characteristics for the most important technologies of the memory and storage hierarchies.

There are many different storage media such as magnetic tape, hard disk drives and solid-state disks. Many of them exist in different form factors or for different interfaces which leads to an entire zoo of different commercial and experimental options. Figure 2.6 shows two plots that relate the cost to latency and cost to power consumption ratios. The plot distinguishes between volatile (turquoise) and non-volatile (red) technologies. Table 2.1 lists the properties latency, throughput, I/O operations per second, capacity, a cost estimate, power consumption endurance and retention times for some widely available technologies.

Technology/ Form Factor	Latency	Throughput Read/Write	IOPS	Capacity Unit	Cost ~$/GB	Power ~W/GB	Endurance DWPD	Retention Time
Registers	< 1 ns	-	-	32/64 bits	-	-	∞	hours
L1 Cache	~ 5 ns	-	-	32+32 KB	-	-	∞	hours
L2 Cache	~ 10 ns	-	-	< 1024 KB	-	-	∞	hours
L3 Cache	~ 20 ns	-	-	8–12 MB	-	-	∞	hours
DRAM	~ 80 ns	17/17 GB/s	-	< 64 GiB	5.000	0.1500	∞	~ 5 ms
NVRAM	~ 5 µs	2.5/2.5 GB/s	4.6M	< 480 GB	1.200	0.0300	NA/(~1000)	NA/(~10 y)
SSD (NVMe)	~ 20 µs	8.0/5.0 GB/s	1.2M	< 32 TB	0.200	0.0007	~ 0.1-25	> 10 y
SSD	~ 100 µs	2.1/2.0 GB/s	0.8M	< 8 TB	0.100	0.0018	~ 0.1-25	> 10 y
HDD	~ 10 ms	250/240 MB/s	< 500	< 14 TB	0.030	0.0004	-	> 10 y
Tape	> 20 s	315/315 MB/s	-	< 15 TB	0.001	-	-	> 30 y

Table 2.1: A snaphot as of 2019 comparing memory and storage technologies and performance characteristics compiled from various specification sheets for commercially available products [HGST, 2017a;b; Intel, 2017b; Seagate, 2017; 2018; Kingston, 2017; Toshiba, 2017; 7-cpu.com, 2020].

2.4 Scientific Data

The shapes of data commonly observed in high-performance computing environments is the result of an interplay between scientific requirements and the available technologies. Many of these building blocks for clusters and storage systems are not designed especially for the use in supercomputers, but by other demanding environments in industry or for consumer markets. As such scientific users are often not the primary market. This shows in workarounds and an engaged community that builds software and sometimes hardware solutions on top of widely available commodity hardware. In regard to storage systems, it is therefore useful, to take a closer look at terminology and common shapes of scientific data in the context of HPC. Because supercomputers are especially well suited for workloads that do not fit onto a single compute node but still require tight interaction between subsets of data needed to compute the larger problem, there are various implications for generated data.
There is a number of different ways to categorize data generated by measurement instruments or simulations, for example by considering the underlying methods they employ. From a data perspective one can roughly distinguish the following two types of data:

- Gridded or *structured* data, often employing a regular grid and stencils to model neighborhood are widely used in engineering, climate, and weather applications. N-dimensional arrays are also common outputs of sensors/instruments/cameras as generated by remote sensing applications such as satellites, telescopes but also microscopes.

- Meshless, *unstructured* or point cloud data are often used to represent collections of individual measurements or in particle simulations, as are common in molecular dynamics or astronomy.

How to best organize this information, however, depends largely on the objectives such as why data is stored, and how it is being processed. Still, scientific data formats share commonalities which to an extent can be attributed to data description formats such as NetCDF or HDF5. The benefits of such standards can be motivated by a variety of different perspectives.

- *Binary Representations:* Often large amounts of data are generated, so that binary representations are the only viable choice. Often additional, advanced features such as built-in compression, indexing, and support for metadata are required too. But such complex binary data is hard to handle without tooling to make it convenient.

- *Cross-platform:* Scientists want their data to be portable and easy to share. Because the file and directory conventions across different operating systems until now vary considerably, other solutions had to be invented. The required object counts and filesize limits used to pose challenges too, so that special index structures are offered by scientific file formats.

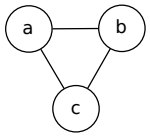
- *Efficient (Parallel) I/O:* Efficient data access across different platforms can be offered by a data description library for a number of common use cases. This is especially important in parallel environments because many users can not and do not want to become experts on the system details which are quickly evolving.

2.5 Graphs

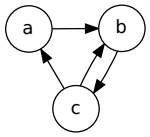
Throughout the manuscript, graphs are being used to visualize and model workflows, systems, and other relationships. This section provides a brief formal definition of important graph concepts and nomenclature.
When reasoning about workflows and computer systems, a subfield of mathematics called graph theory offers many tools and mathematical structures to model relationships between objects. Objects in this context are typically referred to as *vertices* V (with nodes or points often being used synonymously) while connections or relations between vertices are referred to as *edges* E (also links). So that a *graph* G would be defined as follows:

$$G = (V, E)$$

V in this context is a set of elements $\{a, b, c\}$ and E is a set of two-sets like $\{(a, b), (b, c), (c, a)\}$. Graphs are routinely visualized such as illustrated in Figure 2.7, although this becomes impractical for very large graphs.
It is often useful to distinguish undirected and direct graphs when an edge on a graph can be only traversed in one way but not the other. An escalator or a one-way street would typically be modeled using a directed edge. Figure 2.7 (b), for example, visualizes the directed graph for the vertice list $\{(a, b), (b, c), (c, a), (c, b)\}$. In the context of computation and workflows, graphs without any paths from a node back to itself, so-called *directed acyclic graphs* (DAG), are often used as they can be used to determine the order in which tasks in a workflow have to be executed.



(a) undirected



(b) directed

Figure 2.7: Visualizations of two commonly considered types of graphs: (a) an undirected and (b) a directed graph.

Graphs are also used to describe the topology of a network that connects different physical components in a computer system. Here it is often useful to consider weighted graphs, as illustrated in Figure 2.8, which can be used to model latency or throughput in a computer network. For this, another way to represent graphs in matrix form can be used which is also a form which many graph algorithms employ for analysis:

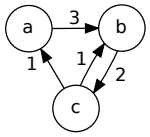


Figure 2.8: A directed graph with weighted edges as might be used to model network throughput or indicate dataflows.

$$G = \begin{pmatrix} 0.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 2.0 \\ 1.0 & 1.0 & 0.0 \end{pmatrix}$$

Graphs enjoy widespread adoption in computer science, biology, linguistics, physics and chemistry, social sciences, and other fields. As a result, there is a wide range of graph algorithms for many common problems faced by different applications. Examples include routing problems, flow determination, graph coloring, graph search and covering algorithms, or graph classification. Often the class of a graph determines which algorithms are most appropriate, for example, there exist algorithms that are especially efficient for weighted graphs, but the algorithm can only be used when no negative edges are in the graph.
A particular useful task to perform on graphs from a workflow perspective is testing for sub-graph isomorphy. That is to test if a smaller graph occurs in a larger one. In a workflow scenario, this can be useful, for example, optimization may apply if tasks relate to each other in a particular way. Unfortunately, unless certain simplifications to the graph are acceptable, testing for subgraph isomorphy is NP-complete so that it becomes intractable for larger graphs [Cook, 1971].

2.6 Control and Machine Learning Techniques

As outlined in the introduction, it becomes increasingly important to find adaptive or self-learning approaches to automate and control the next generation of storage systems. This section, therefore, introduces a number of important concepts, jargon, and methods originating in the fields of control theory on the one hand, and machine learning on the other. As these are broad fields, the focus here is to offer an intuition required for the discussion which follows in Chapter 6 (Architecture for Workflow-Aware Decision Components) and Chapter 7 (Evaluation). Covered in particular are:

- PID Controllers as commonly used in plant automation and engineering.
- An overview of common learning methods based on neural networks.
- Self-Organizing Maps as used for clustering and visualization.

For a more in-depth introduction, a good overview of machine learning and deep learning, in particular, is provided by textbooks such as [Goodfellow et al., 2016]. For specific topics, many educational institutions but also corporations are offering online courses, recorded lectures, or interactive teaching aids helpful to better understand methods.

2.6.1 Control Theory and Feedback Loops

Control theory provides systematic approaches to design feedback loops [Abdelzaher et al., 2008]. On a number of occasions, optimization opportunities for storage systems are expected to benefit from monitoring the system to derive a corrective action, which is compared to finding a higher-level equivalent to *proportional-integral-derivative* (PID) controllers. A PID controller is a mechanism commonly used in industrial control systems which establishes a feedback loop by continuously calculating an error value $e(t)$ given a desired set point $r(t)$ and a measured processes variable $y(t)$. Figure 2.10 and Figure 2.9 illustrate the different computations P, I, and D involved and the resulting correcting action on a system. Each term can be tuned individually by adjusting the respective gain factor K_p, K_i, and K_d. The function of each term can be summarized as follows:

- The P term accounts for the current value and is proportional to the difference between the set point and the process variable.
- The I term allows accounting for accumulated errors, achieved by taking the integral over past error values.
- The D term results in an anticipatory control dampening the control action based on the current rate of change.

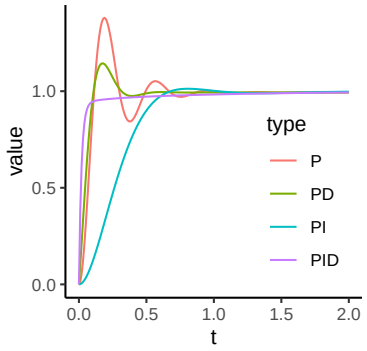


Figure 2.9: Effect of the different control terms on the convergence to the set point $r(t)$. Note, for example, the effect of overshooting which is especially pronounced for P control resulting in decaying oscillations around the set point.

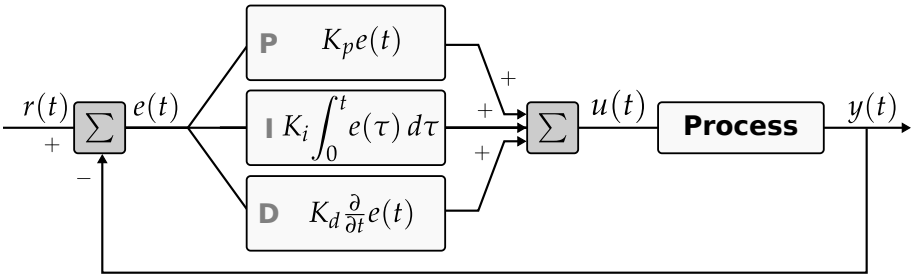


Figure 2.10: Illustration of the computations and their relations in a PID controller. Figure based on [Urquizo, 2011].

2.6.2 *Learning Methods and Neural Networks*

Neural networks have a long history, with the first computational model inspired by biological neurons being introduced already in 1943 [McCulloch and Pitts, 1988]. With the introduction of the perceptron [Rosenblatt, 1957], at the time an analog machine for image recognition, able to process the input of 400 photocells, an architecture was pioneered which seemed promising to allow quick progress towards building consciousness electronic computers which would be able to see, walk and talk. It did not take long to disperse such hopes for a while after a disillusioning report by the Automatic Language Processing Advisory Committee (ALPAC) in 1964 resulted in the withdrawal of funding by the US National Research Council across the field. The publication of a book by [Minsky, 1969] further eroded the belief in "connectionistic" approaches as it demonstrated some fundamental limits to the capabilities of perceptrons. Since then, the fields of machine learning and artificial intelligence have experienced multiple hype-cycles, but many methods today have become standard tools to many researchers. This, in part, is a result of algorithmic breakthroughs such as *backpropagation* which iteratively distributes the error for input-output pairs across weights and layers during training. Another important factor is the proliferation of affordable Single-Instruction Multiple Data (SIMD) hardware well suited to accommodate the calculations routinely required for neural networks. This allowed to consider more complex networks often with remarkable performance in tasks such as classification, pattern recognition or regression.

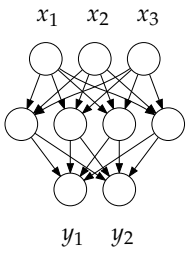


Figure 2.11: A simple feed-forward neural network with three inputs x_n, a single hidden layer with 4 nodes and two outputs y_n.

Feed-Forward Neural Networks: Among the most basic architectures used for *supervised learning* are Feed-Forward Neural Networks (FFNN) which share many similarities with the original perceptrons, but typically will use an artificial neuron, or more formally, an activation function φ for which the derivative is continuously defined to allow the use of backpropagation. The power in FFNN lies in their capability to approximate any non-linear function. A simple FFNN is illustrated in Figure 2.11 with inputs x_n and the network's outputs Y_n, typically these will be vectors. The nodes in between the inputs and outputs are called hidden layers, and many approaches referred to as deep learning will employ multiple hidden layers.

Recurrent Neural Networks: Feed-Forward Neural networks retain no state or context-sensitivity adapting to recent inputs by themselves, but simply map from an input to an out. Recurrent neural networks (RNN) channel back activations from a previous usage back into the next usage [Elman, 1990]. This way it is possible to achieve time or context-sensitive predictions while maintaining a small and compact network, at the cost of higher training times. Understanding an RNN can be especially complicated because the predictive dynamics are obscured in the complex interactions of weights and activations. RNNs often struggle to cope with context-sensitivity for signals which are spaced far apart in time.

Long-Short-Term Memory: An architecture to address long-term temporal relationships is Long-Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] which is particularly suited for time series data. Instead of traditional neurons, a combination of a memory cell along with a number

of gates to record and limit access and manipulations to a memory cell during inference and training are used. The network of weights and "neurons" remains structured similarly to traditional neural networks.

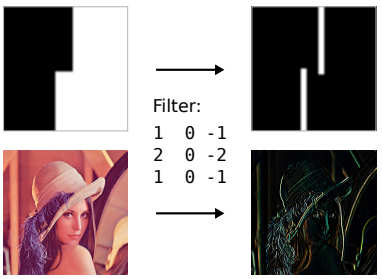
Convolutional Neural Networks: Often used in the context of pattern recognition are Convolutional Neural Networks (CNN) [Lecun et al., 1998]. This architecture often mixes an adaptable feature extraction mechanism with other architectures, typically feed-forward networks.
At the core of the architecture is the convolution operation (see Figure 2.12) which produces a combination of two functions over a certain input window. In CNNs, the input data gets convoluted with so-called filters which effectively serve as small feature detectors. As there are no constraints to the dimensionality of these filters, the approach is broadly applicable but most intuitively understood with images where inspecting the learned filters often yield filter kernels resembling similarities to approaches such as the Sobel operator to detect edges in computer vision. To keep CNNs computationally feasible but also to add robustness to rotational/positional variance, so-called (max) pooling layers are often used to downsample inputs while preserving dominant features before passing on to subsequent layers. It is common to use pre-learned filters to speed up training, and because some filter sets generalize quite well across a variety of applications.

(Variational) Autoencoders: An approach useful beyond classification/regression/recognition for transfer learning, which also helps to keep what was learned explainable, are Autoencoders (AE) [Bourlard and Kamp, 1988]. They are in principle a special case of FFNN and are trained just alike but the architecture creates an artificial bottleneck that forces the model to disregard information and come up with a compressed representation often called the *latent representation*. As the training process does not require any labeled data to create this latent representation they are considered an *unsupervised method*. AEs are often symmetric and it is useful to conceptualize AEs as basically two networks, an encoder X and a decoder X', where X compresses the input and X' reconstructs it. As such they can be used for interpolation, to repair occlusions, or to denoise inputs [Vincent et al., 2008]. While most AEs approaches can be used in theory to also generate useful outputs when feeding arbitrary latent representations to the decoder X' the results are often nonsensical or not plausibly observable. An architecture that is more effective at this by accounting for the distribution of samples are Variational Autoencoders (VAE) [Kingma and Welling, 2014].

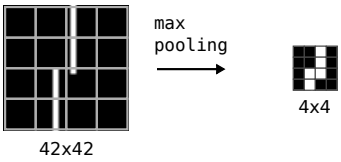
Generative Adversarial Networks: An especially effective approach to learn the latent space or generate convincing data are Generative Adversarial Networks (GANs) [Goodfellow et al., 2014]. They exploit the interaction of two network models that are trained in conjunction. One network, the generator G learns to reproduce input data similar to AE, while a second network, the discriminator D is trained to detect whether it is presented with an actual sample or with a generated one. Whenever G fails to convince D that the generated image is authentic G evolves, and vice versa. The approach is typically more expensive and complex to train, because of the additional networks and because G and D have to be somewhat balanced to allow convergent behavior.

$$(f * w)(t) \triangleq \int f(\tau) w(t-\tau)\, d\tau$$

(a) Convolution Operator



(b) Example 2D Convolution (Sobel$_x$)



(c) 2D Max-Pooling Example

Figure 2.12: Definition of the convolution operation which is the basis for convolutional neural networks. In image recognition for example it is common to use 3x3 filter kernels which can be understood as feature detectors. Max-pooling can be then used to shrink the representation while preserving found features.

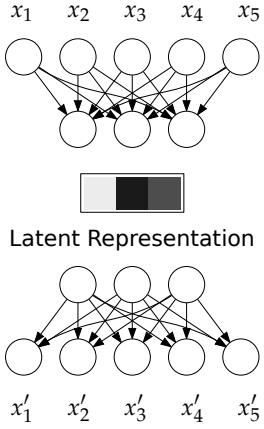


Figure 2.13: Illustration of a simple Autoencoder with a three value latent representation.

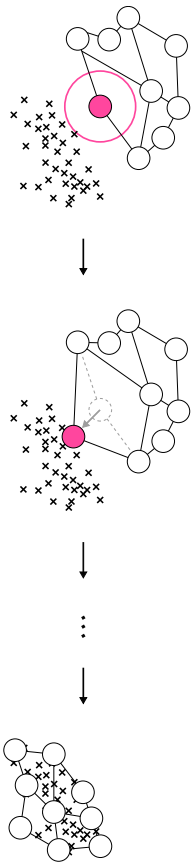


Figure 2.14: Illustration of the training process for a self-organizing maps. The position of every vertex of a topological template (here a grid) is repeatedly updated until the shape of the original data points (crosses) are matched.

2.6.3 *Self-Organizing Maps*

Section 7.3.3 (Decision Aid: File Type Clustering and Classification) makes use of a clustering algorithm often referred to as Self-Organizing Maps (SOM) or as Kohonen Networks [**?**] after Teuvo Kohonen. SOMs are an unsupervised method for dimensionality reduction useful to cluster or visualize data. By taking advantage of a distance function to drive a competitive learning process the algorithm iteratively adjusts weights to better fit an input $x(t)$ and this way effectively trains a projection between a topographic map and a dataset. Figure 2.14 illustrates how a SOM converges during training to fit the input data. How well a SOM performs depends on a number of parameters including the topology, the distance function d, the learning rate α, and the neighborhood function θ. More formally, for an input vector $x(s)$ of size n and s denoting the iteration step,

$$x(s) = [x_1(s), x_2(s), \ldots, x_n(s)]$$

there are n_{topo} nodes m_i so that the number of weights w_{it} is defined by the product $n \times n_{topo}$, where n_{topo} is the number of nodes in the topology:

$$m_i(s) = [w_{i1}(s), w_{i2}(s), \ldots, w_{in}(s)]$$

Often used topological maps include square or hexagonal grids because they can be easily inspected in visualization, although arbitrary meshes can be used in principle. Most commonly, the euclidean distance is used to determine what is called the *best matching unit* (BMU):

$$d(x, w_i) = \sqrt{\sum_{j=1}^{n} (x_j - w_{ij})^2}$$

So that the model can be updated as follows:

$$m_i(s+1) = m_i(s) + \alpha(s) \;\; \theta(bmu, m_i(s), s) \;\; (x(t) - m_i(s))$$

The learning rate $\alpha(s)$ is typically decayed over time hence the step parameter s. The neighborhood function $\theta(bmu, n, s)$ determines which topology nodes (or rather which weights) should be updated. Typically, the proximity criterium sets θ to return 1 when within a threshold and 0 otherwise. Another common choice for *theta* uses the Gaussian function. As with the learning rate, the neighborhood function changes it's behavior over time through the step s, considering fewer neighbors for updates over time. Faster convergence can often be achieved when batching updates. A summary of the algorithm is given in Listing 2.1:

Listing 2.1: Pseudocode outlining the algorithm used to calculate Self-Organizing Maps.

```
m = Som(nodes=len(dataset[0]), topology, weight_initialization=random)

while (!converged(som) or step < max_iterations):
step += 1
x = pick_random_from_training_data(dataset)
bmu = m.find_best_matching_unit(x, distance_func=d)
nb[] = m.get_neighborhood(bmu) # can sometimes be omitted using only θ

m.update_weights([bmu] + nb, lambda w: w + α(step) * θ(bmu, w, step) * (x - w))
```

More recently, many clustering methods are being based on UMAP [McInnes et al., 2018] or t-SNE [van der Maaten and Hinton, 2008] to reduce high-dimensional data into lower dimensional representations, while preserving clusterings from higher dimensions.

Chapter Summary

This chapter introduced important terminology and background to provide a better intuition for each of the different subsystems which are commonly found in HPC data centers. Starting with a high-level overview of a typical data center, the compute cluster, the network, and the different storage systems were introduced. For storage systems, a closer look and the underlying technologies and their characteristics were discussed. Section 2.4 (Scientific Data) continued to relate the previously introduced technologies to handling different shapes of scientific data which are most commonly observed on HPC systems. In particular, these include gridded and meshless datasets which are common in- and outputs generated by simulations. Data formats were only covered briefly, as a more in-depth discussion follows in Section 5.4 (Middleware Level). Finally, the chapter concludes with the introduction and definition of important concepts and algorithms from different fields such as graph theory, control theory, clustering, and machine learning as many design decisions are driven by an attempt to integrate those methods which already have proven to work well for in other fields.

3
Scientific Workflows in High Performance Computing

This chapter introduces a user perspective on the structure and execution of scientific workflows on HPC systems. Section 1.2 already discussed a high-level overview on scientific workflows, here a more formal definition and how scientific methodologies lead to some commonly observed workflow patterns are discussed. Finally, use cases encouraging the declaration of workflows to better plan resource utilization, ensure reproducibility, and to manage ever-increasing amounts of data are discussed. The chapter is broken down into the following sections:

- *Section 3.1 abstractly defines scientific workflows from an I/O perspective.*
- *Section 3.2 describes a number of common workflows seen in HPC.*
- *Section 3.3 offers some examples on how workflows are typically defined when using some common workflow management systems.*
- *Section 3.4 describes how taking a workflow perspective offers performance, reproducibility and data management benefits.*

"Nature uses only the longest threads to weave her patterns, so each small piece of her fabric reveals the organization of the entire tapestry."
– Richard P. Feynman

3.1 Defining Scientific Workflows from an I/O Perspective

The previous chapters discussed scientific workflows in the context of HPC including technical boundary conditions as well as the general ecosystems in which workflow management is performed. This section offers a more formal definition of HPC workflows and adds a storage perspective which is often omitted but required for the remaining considerations in Chapter 5 and Chapter 6.

As outlined in Section 1.2 (Computational Sciences & Workflows), scientists commonly perform experiments on supercomputers using some notion of a workflow. Often, these workflows may not be explicitly specified in a form easily accessible to machines but instead are presented as a high-level description in a project proposal or as sets of scripts used by researchers. For complex workflows, especially when repeated many times, researchers are more likely to opt for workflow management systems (WMS), sometimes synonymously also referred to as workflow engines. A workflow management system provides and implements a task or data model requiring users to merely define the relationships, while the workflow engine executes the workflow with the potential to transparently optimize resource utilization. Besides convenience for users, defining workflows offers opportunities to better anticipate future activity, including activity that will affect storage systems.

As no consistent universal definition for workflows holds over time and across different fields, this section only briefly offers a definition that aims to be useful for workflows in HPC environments with storage systems in mind. To an extent, the definition leans on typical HPC workflows identified and published in an APEX[1] whitepaper on workflows [LANL et al.,

[1] The Alliance for Application Performance at Extreme Scale (APEX) is a partnership between Los Alamos National Laboratory (LANL), Sandia National Laboratories, and the National Energy Research Scientific Computing Center (NERSC).

2016]. The perspective on how scientific workflows are expected to evolve for next-generation workflows and systems is supported by [Deelman et al., 2018] as well as multiple reports by PRACE (Partnership for Advanced Computing in Europe) [Povh, 2016; Mendez, 2019; Carreras, 2018].
Most commonly, a *workflow* describes a number of *tasks* that need to be performed to achieve a higher-level goal. Tasks will usually consume or yield a piece of *data*, which in many cases leads to an order in which tasks need to be executed. Also worth considering are *pipelines*, since it is common to see the same series of steps being performed on different input data, which in many cases gives rise to parallel execution.

Figure 3.1: Data and tasks relationships can be expressed explicitly when using graphs to represent a workflow.

Data: Data comprises information that is retained or passed between tasks. Depending on the lifetime, data can be messages, database entries, files, or objects on a storage system. The granularity of data objects varies considerably between different workflow engines. Many workflow engines use notions of datasets in a hierarchical namespace, while others directly assume the presence of file systems or database management systems (DBMS). In extreme cases, workflow engines might consider individual variables as small as a single byte or integer as data objects. Other systems add means to declare custom array-like structured data or compound data types.

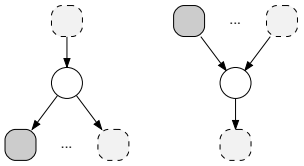


Figure 3.2: Common task roles and behaiviors: producers and/or consumers, as well as considering "split and merge" (or fan-out and fan-in) behavior of a task.

Task: A task is a logical entity or a program that can consume and/or produce data. Usually, a task will require some resources to perform a computation or to process data. It is also useful to consider split/merge behavior of workflows. Where a merge typically corresponds to the aggregation or reduction of data, and a split yields multiple different results or subsequent tasks. Granularity varies depending on the abstraction used by different workflow engines, with a single task potentially mapping to a job, a process, a thread, a function call or even a single operation. From a storage perspective, it is very common to focus mainly on jobs or processes. A job typically will yield a number of files/data objects. For large data objects, moving or consolidating data from one node/process to a storage system is typically limited by the network.

Workflow: A workflow governs the dependencies of tasks and data and is often represented by using a graph, with nodes being used to model tasks while edges represent dependencies. In many cases, this representation will result in a directed acyclic graph (DAG), although complex workflows may contain cycles. Also, considering I/O activity in such a graph can lead to situations where data is being read and written in ways that turn an acyclic task dependency graph into a cyclic task/data dependency graph. Implementation logic of tasks is often strictly separated (e.g., a binary) from the declaration of relationships, while fine-grained workflow engines seem to favor workflow definitions that allow mixing the two.
Not all tasks within a workflow need to be connected. It is not uncommon to see, what is referred to as a *bag of tasks* which are independent of each other as far as the workflow is concerned. During a workflow execution, the workflow might change as it is being executed when new data or tasks are added or discovered during the execution. Other workflows will loop back to an earlier task as is the case in optimization scenarios or in experiments that require external steering, for example, inputs by a scientist.

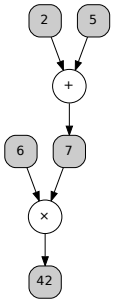


Figure 3.3: A very basic example for a workflow: evaluating a mathematical expression like $(a + b) * c$.

Pipeline: Many scientific workflows feature variations of a repeated chain of processing steps for similar input data, also referred to as a pipeline. Complex pipelines might branch and have conditions to determine which tasks in a pipeline have to be executed. For example, depending on an input set for the workflow, every element in an array might fan-out into an instance of a pipeline. Often pipelines are relatively independent and thus directly expose opportunities to execute tasks in parallel within or across multiple systems. As a result methodologies for task-based parallelism and serverless paradigms are gaining popularity to optimize the usage of compute resources. Unfortunately, this information is currently rarely used to automatically attempt optimizations from a data storage perspective.

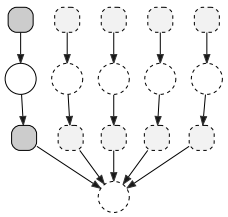


Figure 3.4: Pipelines repeat the same sequence of operations on different inputs. This often gives rise to parallelism. It is further notable because it amplifies the usefulness of any optimization within the base pipeline as it can be applied to any execution.

3.1.1 *Adding a HPC I/O Perspective on Scientific Workflows*

Workflow engines, schedulers and compilers have to implement a task/data model so they can distribute computations and exploit data locality. As long as data is located in memory, moving data occasionally is relatively affordable in comparison to data stored on more persistent media. Most workflow engines attempt minimizing these data movements within an execution and across systems, for example by using caches. This to an extent requires conceptualizing file systems, but most WMS make no attempt of exploiting the characteristics of high-performance storage solutions. Similarly, many modern I/O technologies are not yet fully integrated into many WMS.
For I/O within a running execution of a workflow, technologies such as *Remote Direct Memory Access* (RDMA) allow to asynchronously exchange data over the network at relatively low latency and high bandwidth. On top of these technologies, models such as the *Partitioned Global Address Space* (PGAS) try to offer a simplified memory model to hide the underlying complexity. But many workflow engines are still communicating using file system interfaces while rarely providing users with mechanisms to annotate data which is read and written from persistence storage.
For persistent data storage, some efforts are aiming to hide storage system details from the user and conduct data movements and placement transparently. Examples for this are asset management systems such as iRODS [iRODS Consortium, 2019] or *Distributed Asynchronous Object Storage* (DAOS) [Intel et al., 2014] which aim to transparently integrate a variety of different storage services. DAOS initially also aimed to break with the traditional modes where data is staged onto compute for analysis, and instead allow to ship computations to so-called active storage devices. A prerequisite to perform calculations close to where data is stored is knowledge about the structure and a possibility to extract sub-computations from a larger workflow or application. This model ultimately requires adaptable systems that have metadata at their disposal to decide on the most appropriate way to handle data. Typically, this is achieved at coarse granularity using policy systems, which are sometimes challenging to configure especially when catering to multiple scientific domains.
Defining and enforcing such policies requires to better understand the requirements and I/O access patterns imposed by workflow on storage systems for hot and cold data. While there is extensive research on how to exploit workflows and task-based parallelism for distribution and concurrent computation, few efforts have materialized to exploit the same information in storage systems.

3.2 Common Workflows in High-Performance Computing

In order to address the challenges of current and future workflows, this section introduces workflows commonly observed in HPC environments [LANL et al., 2016]. An outlook on anticipated workflows and recognized challenges that need to be addressed by WMS before they are suitable for widespread adoption in HPC environments is provided by [Deelman et al., 2018]. To a large extent, workflows depend on the facilities, instruments, hardware, and services scientists find in their respective institutions. Workflow execution can span different time-spans from seconds up to years depending on their alignment to a given project. Nonetheless, a number of common patterns for workflows on HPC have emerged, such as simulations, uncertainty quantification, or high throughput computing. More recent patterns originate in machine learning, data analytics, and in situ workflows.

3.2.1 Simulation

Simulation workflows tend to consist of 3–4 major phases. An illustration of a stereotypical simulation workflow is given in Figure 3.5. In the preprocessing phase, raw input data needs to be transformed into an initial state for the simulation. Often this means preparing a so-called initial state which might require the combination of many sets of measurement or observation data. Weather predictions, for example, are driven using data products derived from satellites and countless stationery and mobile measurement instruments.
In a simulation/data generation phase, applications typically will read different input data sets and write snapshots, timestep data, and diagnostic data. Snapshot data is usually preserved only for a few timesteps, as it is stored to add fault-tolerance to simulation steps which can be discarded as the simulation proceeds. Timestep data has to be preserved for longer periods and sometimes indefinitely as it will be used for post-processing or is being archived because the experiments cannot easily be repeated. Diagnostic files are used by researchers and tooling used around the simulation for automation and debugging purposes.
The post-processing and visualization phases are deriving data products that in many cases may be published and preserved. A simulation workflow as outlined here might be part of the pipeline of other more data-intensive workflows.

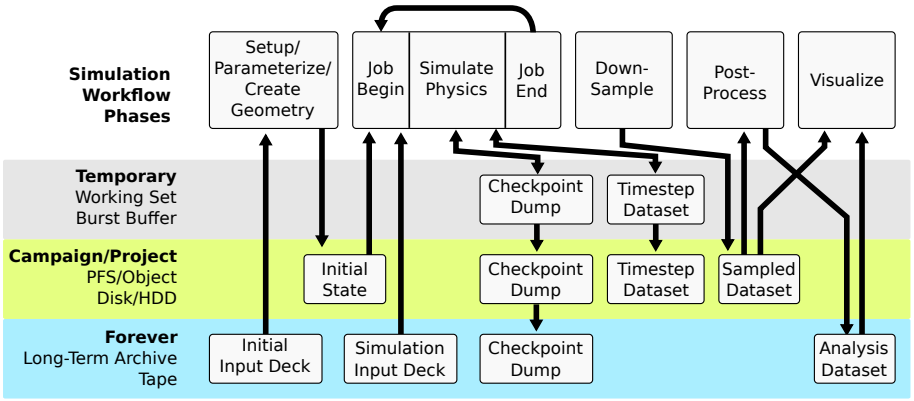


Figure 3.5: A stereotypical *simulation workflow* (see APEX Workflows[LANL et al., 2016])

3.2.2 *Uncertainty Quantification*

Many physical simulations have to deal with uncertainty when modeling nonlinear or chaotic systems. As a result, researchers commonly design workflows for uncertainty quantification (UQ). An illustration of a stereotypical UQ workflow is given in Figure 3.6. Such workflows often consist of a large number of independent pipelines (typically with simulations running for multiple hours) that can be executed in parallel. Results from these parallel executions, often called ensembles, are then combined into analysis data sets. To an extent this approach offers opportunities to perform computations also in parallel over the time domain (Parallel-in-Time [Ruprecht et al., 2019]) in addition to spatial parallelism. UQ workflows seem especially suitable for automatic optimization because similar tasks are being executed many times so that the impact even of small optimizations accumulates. The larger number of observations should also help machine learning methods to learn patterns. Finally, instrumenting a task can cause significant overhead, but if only a fraction of runs has to be instrumented to find an optimization it may be worthwhile.

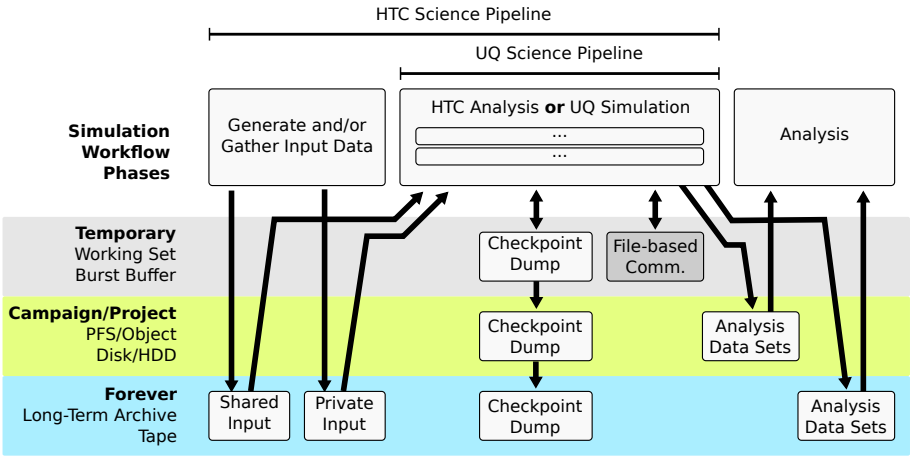


Figure 3.6: Stereotypical structure of *uncertainty quantification* and *high-throughput computing* workflows (see APEX Workflows[LANL et al., 2016]). The figure combines UQ and HTC workflows to highlight their structural similarities, the stress they impose on storage systems is usually very different. While UQ pipelines are executed in an effort to reduce uncertainty, HTC is more often used to explore parameter spaces with different pre-processing per pipeline.

3.2.3 *High-Throughput Computing*

Similar to UQ workflows are high-throughput computing (HTC) workflows, again illustrated in Figure 3.6. In HTC a large number of pipelines are executed, but typically featuring tasks only of limited runtime/data volume. Computational requirements, however, are sustained for longer periods potentially spanning months or years although with lower requirements for communication or synchronization. Often the objective is to explore a parameter space, for example, to find hyper-parameters in machine learning applications or candidate substrates in molecular or protein folding. Sometimes this allows to combine statistical results from many low-fidelity simulations at a smaller overall cost than running a few very large simulations which might even exceed available systems.
From a storage system perspective, high-throughput computing is often associated with many relatively small files. This is not necessarily the case though, a common pattern is for example to see HTC workflows which issue no storage I/O for a large portion of tasks, but if a trigger criterion is matched a dump of a simulation with a large state is written.

3.2.4 *In Situ / Integrated Approaches*

In situ workflows integrate their numerous distinct phases into a single orchestration. Traditionally, these types of workflows have been implemented in the context of MPI but increasingly alternative transports are being used as well. An illustration of a possible in situ workflow is given in Figure 3.7.

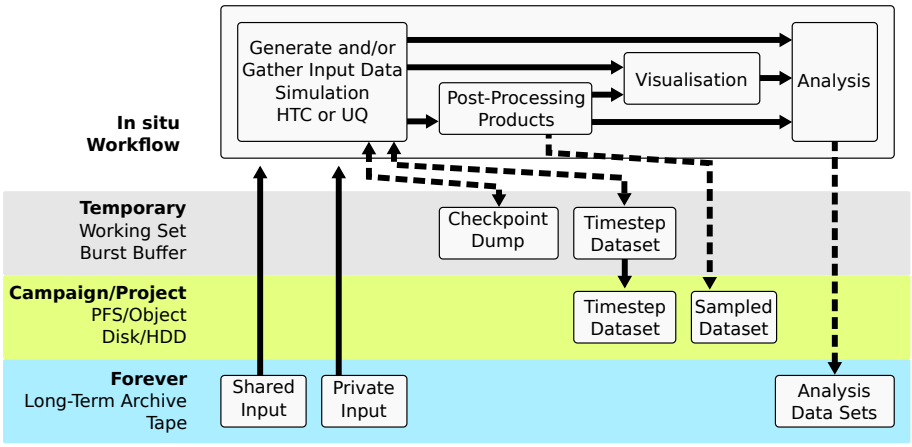


Figure 3.7: Overview of an in *situ workflow* considering the same data life-cycles as in [LANL et al., 2016; Deelman et al., 2018]. In situ workflows are anticipated but not widely spread at this time as frameworks are in their early stages and adapting existing applications is not always straightforward. The dashed lines indicate that also an in situ pipeline might issue I/O to persistent systems, but often does not or at a much lower rate.

Such workflows allow for the conservation of resources by exploiting data locality and avoiding overheads from resource allocation, context switching, and application startup/shutdown. As far as storage systems are concerned, the requirement to occasionally consume large amounts of snapshot data remains; but overall in situ workloads promise to reduce the load across network and storage systems.
As visualization is among the most common in situ workflows many other approaches have adopted interfaces similar to the coupling APIs offered by visualization suites such as ParaView Catalyst [Kitware, 2016].
This is especially attractive for production environments with anticipatable workloads where data generation and post-processing products can be computed side-by-side within a node that already holds the required data.
Applications and requirements for computational steering and interactive visualization are relevant also beyond purely scientific contexts such as for disaster response systems [Coen and Schroeder, 2017].
With new machine learning methods more sophisticated triggers to write data selectively offer opportunities to conserve storage. While in situ data management is anticipated to play a key role in exascale computing many of the implications for applications, data models, algorithms, provenance, and reproducibility are an emerging area of research [Biven et al., 2019].

3.2.5 *Hybrid Approaches*

In practice, workflows are not as pure as illustrated in the previous examples. Workflows will continue to evolve, and in many cases, scientists will mix workflow models that are not easily integrated or solved by a single workflow orchestrator. Since collaborations in projects often span across multiple locations and organizations, one must assume that different workflow tools are being used even within a single project. Consequently, tools to analyze workflows should address this heterogeneity, for example by offering abstractions and modularity to integrate other tools and WMS with reasonable effort.

3.3 Workflow Management Systems

This section compares workflow management systems and engines in detail and collects features commonly found across different frameworks. The study starts with an overview, pointing out a few notable features, especially when taking a storage perspective.

3.3.1 Feature Spectrum in WMS

In this section, the landscape of workflow engines is surveyed for common features and data models, workflow representation, telemetry capture, and mechanism to influence resource utilization. As there is a large number of workflow engines with vastly different objectives. [Crusoe et al., 2019], for example, lists over 200 workflow systems, the following survey is limited to engines and features relevant to workflows executed in HPC environments. For workflow-aware storage systems, a data model and access to telemetry and monitoring information are important to inquire about and adapt to previously observed behavior. Facilities and APIs to collect and query telemetry information are built into some workflow engines, although I/O performance is usually not addressed specifically.

While some WMS are providing features like monitoring APIs to log events [Hendrix et al., 2016; Palazzo et al., 2015], facilities within workflow engines to capture detailed I/O behavior are often missing. Other engines monitor jobs and also keep track of output files even across multiple sites, sometimes even including sanity checks to avoid transferring faulty results [Oliver et al., 2019; Jain et al., 2015]. Increasingly third-party monitoring and instrumentation solutions are being integrated such as turning to MPE[2] for logging in MPI-based workflow engines [Wozniak et al., 2013c].

In some cases, tools for visualization and dashboards are offered by workflow engines directly [Swift Team, 2015; Deelman et al., 2015; Jain et al., 2015; Palazzo et al., 2015; Rocklin, 2015; Kepler, 2016; Tejedor et al., 2017; Oliver et al., 2019].

Increasingly, workflow engines offer users to export task graphs using the dot format to visualize workflows [Rocklin, 2015; Oliver et al., 2019]. External tools and dashboards have also been developed to monitor workflows or WMS. Grafana dashboards, for example, have been created [Hammer Lab, 2015] to monitor Spark.

As scientists continue to add complexity to their workflows and strive for reproducible scientific results, they demand more elaborate tools and frameworks that can automate the execution of workflows. Workflow management systems have gained considerable popularity with an increase in big data applications and have also become increasingly popular with users in the HPC community. Since some of the tools popular in a big data context are challenging to deploy in HPC systems, a number of WMS explicitly designed for use on HPC systems are available and under active development. To an extent, this fragmentation can be attributed to the slightly different priorities from one scientific domain to another. Section 3.3.2 provides an overview of HPC WMS as well as a comparison of features relevant to an I/O and storage perspective.

[2] MPI Parallel Environment (MPE) is a software package for MPI programmers offering profiling libraries, viewers for logfiles, wrapper generation and debugging routines.

3.3.2 Comparison of Workflow Engines

Table 3.1 compares a variety of workflow engines and tools used in HPC environments to execute and manage workflows, which later informs the discussion in Section 5.2 (Workflow and User Level). The comparison matrix considers the following criteria:

Version: Version for which the information apply. When was the last software development activity? If releases are offered the release date and version are recorded. Otherwise, activity in a source code repository is taken.

Engine Type: Engine type describes the underlying approach to the workflow management system. More integrated solutions provide their own runtime, including their own scheduling systems. Some of those are strictly limited to a single site or system. Engines that can coordinate tasks across multiple site are listed as *dist. jobs*. Engines that spawn a runtime and then interpret task or workflow specifications are listed as *runtimes*. Engines that are primarily controlled through a web frontend or a remote API are listed as *platforms*. Engines that allowing to submit jobs into a queue for a HPC cluster are listed as *batch* systems. Engines that coordinate in situ processing are listed as *in situ*.

Data Models: The underlying data model of a workflow description determines which kind of optimizations the engine can perform. Data models span domain-specific languages which can account for single variables, to more coarse granular models that look at datasets, files, or objects. Some workflow tools introduce additional abstraction. More special files such as diagnostic output and semaphore files are usually not explicitly denoted.

CWL/DWL Support: Is a workflow description standard such as the Common Workflow Language (CWL) or the Workflow Description Language (DWL) supported?[3]

[3] Domain-specific languages to consolidate how to express scientific workflows are discussed in more detail in Section 3.3.5 (Common Workflow Language and Workflow Description Language).

Graph Export: Most workflow engines represent the workflow using some notion of a graph, but how tasks and data map to vertices and nodes varies considerably. The representation is often only used internally, for example, to support scheduling. If the workflow engine offers an easy way to export the workflow graph for consumption by third-party tools this field is set.

Telemetry Support: This column indicates if the workflow engine has abstractions and facilities to manage telemetry information. Many engines do, although most workflow engines are very restrictive for which type of telemetry they collect.

I/O Telemetry: In the context of workflow-aware storage systems, it is particularly relevant if I/O telemetry is collected. This column indicates if I/O telemetry is addressed specifically.

Adapt Behavior: This column indicates if facilities or mechanisms to integrate decision components are available. Some open-source codebases, for example, when using Plugins, can be extended without explicitly recompiling the engine. If this is not encouraged in the documentation the list still denotes ✕ to indicate no deliberate mechanism is provided.

Other Symbols and Abbreviations:

✓/✕	Feature present/not present	(✓)/(✕)	Yes/No with restrictions
?	Possibly, but not documented	–	Unlikely, and not documented
CWL	Common Workflow Language	WDL	Workflow Description Language
DSL	Domain-Specific Language	RDD	Resilient Distributed Datasets

WMS	Version	*Engine Type*	*Data Models*	*CWL/WDL*	*Graph Export*	*Telemetry*	*I/O Telemetry*	*Adapt Behavior*
Swift/K [Wilde et al., 2011]	2015 (0.96.2)	dist. Jobs	DSL	×	(✓)	×	×	×
Cylc [Oliver et al., 2019]	2021 (7.9.3)	dist. Jobs	(✓)	×	✓	✓	×	×
Fireworks [Jain et al., 2015]	2019 (1.9.0)	dist. Jobs	–	×	✓	✓	(×)	×
Pegasus [Deelman et al., 2015]	2020 (4.9.3)	dist. Jobs	In/Out, Diagnostic	✓	✓	✓	✓	Planner
Makeflow [Albrecht et al., 2012]	2021 (7.2.8)	dist. Jobs	Files	×	✓	✓	✓	Hooks
Swift/T [Wozniak et al., 2013b]	2018 (1.4.1)	Runtime (MPI)	DSL	×	(✓)	×	×	×
(Py)COMPSs [Tejedor et al., 2017]	2020 (2.8.0)	Runtime (Java)	Python Objects/Files	×	✓	✓	(✓)	(✓)
Dask [Rocklin, 2015]	2020 (2.30.0)	Runtime	✓	×	✓	✓	Bytes per Process	I/O Backends
Tigres [Hendrix et al., 2016]	2016 (0.2.0)	Runtime	"Inputs"	×	–	✓	×	×
Spark [Spark, 2018]	2021 (3.1.1)	Runtime	"RDD"	(✓) Toil	✓	✓	(✓)	×
Ophidea [Palazzo et al., 2015]	2019 (1.5.1)	Runtime	Datasets	×	✓	✓	×	×
Kepler [Kepler, 2016]	2015 (2.5)	Runtime	(✓)	×	✓	–	×	×
Taverna [Wolstencroft et al., 2013]	2018 (3.1.0)	Runtime	Inputs/Outputs	(✓) CWL	✓	–	×	Plugins
ecflow [Santoalla and Bonet, 2019]	2019 (4.14.0)	Runtime	Files, Datasets	×	?	✓	–	×
Galaxy [Afgan et al., 2018]	2019 (19.01)	Platform	✓	✓CWL	✓	✓	×	×
Mistral [OpenStack, 2013]	2019 (10.0)	Platform	JSON	×	✓	✓	×	Policies
Celery [Celery Project, 2019]	2020 (5.0.5)	async. Tasks	Workers	×	×	✓	×	×
Airflow [Apache Foundation, 2019a]	2019 (1.10.3)	async. Tasks	Database	✓CWL	✓	✓	✓	Hooks
Flink [Apache Foundation, 2019b]	2019 (1.8.0)	Pipeline	Datastreams	×	?	✓	✓	?
Luigi [Spotify, 2019]	2019 (2.8.6)	Batch (Pipeline)	Files/HDFS	×	✓	✓	×	–
Toil [UCSC CGL, 2017]	2019 (3.19.0)	Batch (Cloud)	Files	✓both	(×)	✓	?	Interfaces
TaskFarmer [NERSC, 2017]	2018 N/A	Batch (cmds)	Files/Shards	×	×	×	×	×
Slurm [SchedMD, 2019a]	2021 (20.11.3)	Batch	only jobs	(✓) Toil	×	✓	(✓)	Plugins
PBS [Altair, 2019]	2019 (18.1.4)	Batch	only jobs	×	×	✓	(×)	Hooks
FlowVR [Raffin and Dreher, 2014]	2014 (2.1.1)	in situ	Inputs/Outputs	×	✓	Tracing	×	×
Decaf [Dreher and Peterka, 2017]	2019 N/A	in situ	Struct Members	×	✓	×	×	×
XIOS [CEA/LSCE and CNRS/IPSL, 2019]	2019 N/A	in situ	Classes	×	×	✓	Timers	×

Table 3.1: Comparison of different workflow management systems relevant in the context of HPC. Workflow engines are roughly characterized by the engine type and the data model that they implement.

3.3.3 *Workflow Engines relevant for Prototype Implementations*

The experimental prototypes for HPC workflow analysis tools used in the evaluation, see Chapter 7 (Evaluation), includes two workflow engines residing at the two ends of the spectrum in terms of workflow granularity: Swift/T implements a fine-grained and integrated approach to workflows that is anticipated to be increasingly relevant as applications strive to exploit exascale systems. Cylc follows a more traditional and distributed workflow approach and assumes the submission of jobs to batch scheduling systems. As explained in Section 3.2 (Common Workflows in High-Performance Computing), it is anticipated that both models will remain relevant. The following two paragraphs provide a more in-depth description of Swift and Cylc.

Swift/T: Swift provides a domain-specific language with many features of a generic programming language to specify workflows. Swift offers two different runtimes to interpret and execute a workflow description: Swift/K [Wilde et al., 2011] as well as Swift/T [Wozniak et al., 2013b; Armstrong et al., 2014]. This work focuses on the more recent runtime called Swift/T (for the Turbine runtime [Wozniak et al., 2013a]) geared toward supporting exascale workloads. The Turbine runtime implements a highly integrated workflow model that launches itself a single, large MPI application and optionally dedicates communicators to subtasks. This approach may also be especially interesting for applications considering the use of in situ techniques [Dorier et al., 2015].
Swift/K, on the other hand, provides a runtime similar to that of Cylc, adopting a model of job-based workflows potentially distributed across multiple sites. In the language shared by Swift/K and Swift/T, the data model can become very fine-grained, with data objects potentially as small as single integers. Each Swift variable is a future, and tasks are represented as Swift functions that block on their inputs. External data in files is represented in Swift file variables constructed with a mapping to an external name that can be a Unix filename or URL. Thus file-based dependencies are represented as passing variables in and out of functions.

Cylc: Cylc [Oliver et al., 2019] implements a distributed workflow model, with data generation and post-processing and pre-processing potentially being executed on different sites and consisting of multiple jobs. Hence, Cylc also allows the collection of results from multiple compute sites. Cylc is implemented in Python and plays well with many common HPC resource managers such as Slurm or PBS. Cylc also acknowledges the need to integrate with other workflow suites and offers various triggers to do so. In Cylc, users define a directed dependency graph of tasks, as well as data objects that are passed between tasks. A special configuration language allows to conveniently describe workflows that result from complex loops via so-called cycle points. Cylc provides an easy mechanism to export a graphical representation of the workflow in dot format. Cylc, being initially developed by the National Institute of Water and Atmospheric Research of New Zealand, is mostly used in the context of climate and weather applications.

3.3.4 *Examples for Workflow Definitions*

Often it is helpful to consider concrete examples to compare and evaluate differences. In demonstrations throughout the thesis, the following simulation workflow is revisited a number of times. It is modeled to resemble the simulation workflow introduced in Section 3.2 (Common Workflows in High-Performance Computing), as it might be performed by a climate modeler. The workflow was kept simple for illustrative purposes and for easy comparison between subsections. A screenshot of the workflow engine's perspective on the DAG of tasks is illustrated in Figure 3.8. The demonstration workflow consists of the following sequence of steps:

1. *Preprocessing Step*: Multiple input files are combined to create the initial state for a simulation.
2. *Simulation Step*: Multiple timesteps are simulated, and a timestep file is written that is read by the next simulation step as the initial state.
3. *Postprocessing Step*: For every simulated timestep some post-processing is performed. In this example, the results of different tasks are appended to a file shared by multiple tasks (imagine a time-series or a movie).

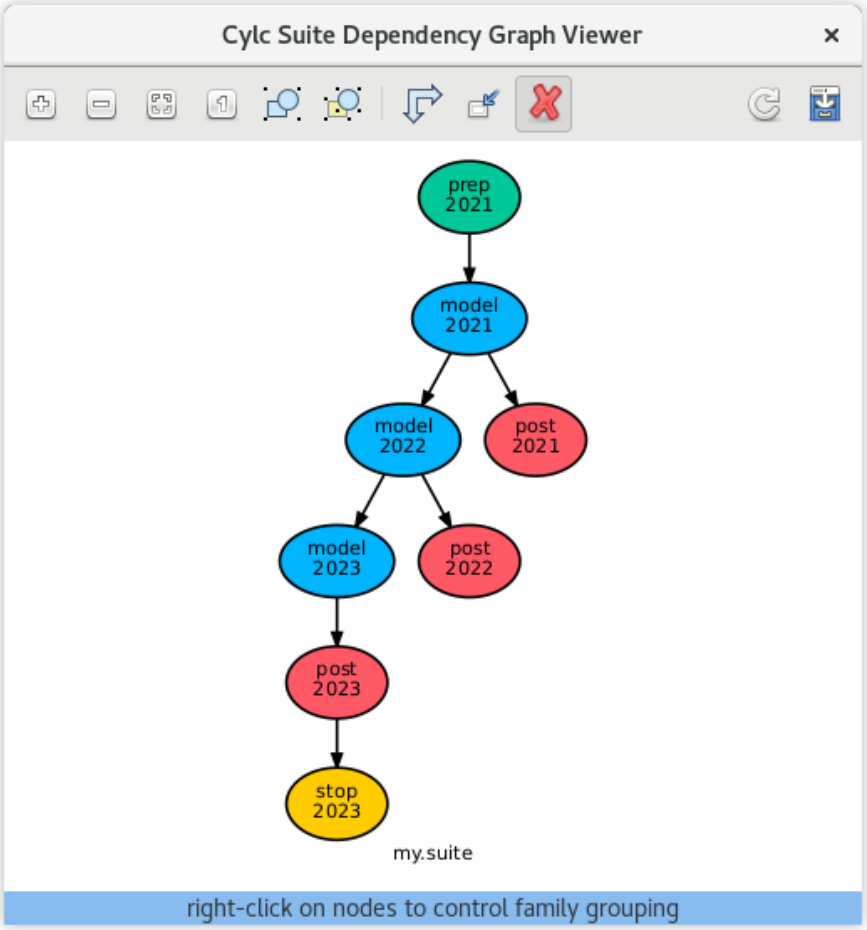


Figure 3.8: A example of a very simple climate workflow here showing only the task dependencies as can be displayed by the Cylc User Interface. The graph is not interactive and provides only very limited information. While various workflow engines offer some form of visualization to view the dependency graph, none of the reviewed engines provided an overlay of I/O telemetry or behavior.

Depending on the WMS, the description of a workflow can vary considerably. Compare, for example, the workflow definition used by Cylc in Listing 3.1 with the workflow definition as it could be defined using Swift

in Listing 3.2 as well as a generic workflow description language such as CWL in Listing 3.3.
In Cylc, a user defines tasks, in this case, for example, as a command-line command to be executed in the `runtime` section. In a separate `scheduling` section the dependencies are defined, including convenience features such as the so-called cycle feature that can spawn a pipeline of events for different input parameters. The application logic and workflow description and configuration are neatly separated. Hereby providing the script parameter an independent MPI application binary is launched.

Listing 3.1: Example of a workflow defined using Cylc.

```
[scheduling]
initial cycle point = 2021
final cycle point = 2023
[[dependencies]]
[[[R1]]]                                # Initial cycle point.
graph = prep => model
[[[R//P1Y]]]                              # Yearly cycling.
graph = model[-P1D] => model => post
[[[R1/P0Y]]]                              # Final cycle point.
graph = post => stop

[runtime]
[[prep]]                                  # Definition of various tasks.
script = mpiexec -np 1 ./prep
[[model]]
script = mpiexec -np 4 ./model
[[post]]
script = mpiexec -np 1 ./post
```

A Swift workflow, on the other hand, looks more like a normal programming language. As a result, identifying individual tasks is not as straightforward. In general, Swift encourages to encapsulate and expose tasks using shared libraries which can be invoked by the Swift runtimes. In this example, association with Swift tasks relies on a convention to have function calls in Swift coincide with the names of binaries to be executed in parallel. This choice was made in order to limit the granularity of the dependency graph since Swift keeps track of individual variable instances and scopes. Instead of dividing aggregate I/O activity for tasks, a more appropriate approach is to associate records with source lines or elements of the acyclic syntax tree of the Swift script. Using the Swift compiler (stc), a Tcl script is generated that can be executed using the *turbine* runtime.

Listing 3.2: The same simple workflow defined in Swift using pseudo-code for the range and array semantics for brevity.

```
import launch;
import string;

int m[]; // assuming integer references to datasets
int p[];

(int output) prep() {
        int nprocs = 1;   string args[] = [];   string env[] = [];
        int output = @par=nprocs launch(nprocs, "./prep", args, env);
}

// Definition of model() and post() omitted for brevity.
// The launch method used here is provided for legacy applications.
// Ideally, Swift/T encourages applications to expose routines in a library
// which are passed their own MPI communicator as a parallel execution context.
// ...

m[2020] = prep();           // a pre-processing step

foreach x in [2021:2023] {
        m[x] = model(m[x-1]);   // a simulation
        p[x] = post(m[x]);      // a post-processing step
}
```

3.3.5 *Common Workflow Language and Workflow Description Language*

Finally, there are approaches to consolidate common concepts into a standardized workflow description language such as the *Common Workflow Language* (CWL) [Amstutz et al., 2016] or the *Workflow Description Language* (WDL) [Broad Institute, 2019]. Workflows defined in CWL or WDL can be mapped and executed on cloud services or batch schedulers such as SLURM or PBS commonly used in HPC environments by using tools such as Toil [UCSC CGL, 2017].
WDL supports a variety of data types that allows getting very granular when describing the data properties featuring concepts such as files, arrays, key-value pairs down to objects or even individual integers. WDL also offers abstractions, for example, to specify dependency to containers such as Docker. WDL also allows using string interpolation, for example, to make input and output names variable. WDL offers scatter/gather semantics.

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.0
class: Workflow
inputs: year: int
outputs: out_filename: string

steps:
prep:                       # a pre-processing step
run: prep.cwl
in:    in_dset: $('raw#' + year)
out:   out_dset: $('m#' + year)

model:                      # a simulation step
run: model.cwl
in:    in_dset: $('m#' + year)
out:   out_dset: $('m#' + (year+1))

post:                       # a post-processing step
run: post.cwl
in:    in_dset: $('#m' + (year+1))
out:   out_dset: out_filename
```

Listing 3.3: The main pipeline defined in YAML using the Common Workflow Language (CWL). A workflow has to specify inputs and outputs and can consist of multiple steps. The examples also demonstrate a particularly useful feature of CWL workflows, which allows nesting by referencing another `*.cwl` script in the run attribute of steps.

```
#!/usr/bin/env cwl-runner
cwlVersion: v1.0
class: CommandLineTool        # Ecapsulated task, here a command line tool.
inputs:
in_dset:
type: File
inputBinding:
prefix: --raw         # e.g., if prep expects --raw argument

outputs:
out_dset:
type: File

baseCommand: mpiexec -np 4 prep
```

Listing 3.4: Definition of a task in a seperate *.cwl file. The specification allows to specify also own classes, the CommandLineTool class here is already provided.

CWL allows describing workflows by declaring data inputs/outputs and the execution context using either YAML or JSON. By binding inputs to data objects or outputs to inputs of subsequent tasks, it is possible to describe data flows. Data is being check-summed to ensure consistency between steps. Workflow logic and inputs/outputs can be declared and passed independently to the cwl-runner. It is possible to add additional metadata information as well as input/output schemas for validation. During declaration of the execution context, it is possible to list requirements, environment variables or to reference container environments (currently Docker). Tasks are modeled as steps, which can be calls to command line commands or built-in functions. CWL also allows nesting workflow

definitions, for example, by referring to a different `*.cwl` file to be included within another CWL workflow. CWL encourages the development of special-purpose tooling around a set of core command-line tools such as validation and visualization tools. An example of how a workflow defined in CWL might look like is illustrated in Listing 3.3.

3.4 Use Cases from a Data and Storage Respective

The discussion so far mostly focused on operational considerations and some of the opportunities to better plan resource utilization in Section 1.1 (Challenges for High-Performance Computing), Section 1.3 (Decision Support for Workflow-Aware High-Performance Storage Systems) and Section 3.1 (Defining Scientific Workflows from an I/O Perspective). In a mid- to long-term perspective, users specifying their workflows help to enable a number of on-going research efforts, but some users will hesitate to invest the extra effort due to uncertain immediate operational necessity or benefits. This section collects some arguments in favor of more explicit declaration of workflows with expected benefits in ensuring reproducibility [Landry, 2019], performance portability [Lordan et al., 2014; Dorier et al., 2015] and management of ever-increasing amounts of data [Newhouse, 2011].
Different stakeholders will find different factors relevant for their use case. [Landry, 2019], for example, distinguishes three different perspectives on workflows in the context of data-intensive sciences. On the one hand, there are *scientific workflows* which enable provenance tracking, reproducibility, and multidisciplinary work. On the technical side are *computational workflows* which allow to transparently distribute work over regions, sites, infrastructure. Finally, as data products mature to become relevant in *operational workflows*, additional requirements such as quality control and integration with scientific and commercial clouds are imposed.

3.4.1 Resource Utilization

The argument for transparent performance portability and exploiting data locality by turning to workflows is also shared by [Lordan et al., 2014].
The idea to shift focus from application workloads to a workflow perspective is not new. A white paper published in the context of the European Grid Infrastructure (EGI) [Newhouse, 2011], for example, has identified documentation of workflows as a common requirement of communities in life sciences, high-energy physics, astronomy, or earth sciences.
More recently, [Deelman et al., 2018] acknowledge a number of I/O related challenges in scientific workflows driven by an increasing heterogeneity of modern data centers and more complex memory hierarchies. Efforts to minimize data movement often require a more detailed perspective on workflow data in terms of inputs, outputs, and transformations. Resource selection and scheduling can only exploit data locality by including, for example, storage resources and not only compute resources as "first-class" resources.
Multiple exascale initiatives [PRACE, 2013; Koethe, 2017; NEXTGenIO, 2018] independently acknowledge the need to be more conservative about writing to high-performance storage systems. To do so, new methodologies for task-based parallelism and task coupling are being investigated, which are enabling technology for prefetching and data staging as well as the

realization of in situ methods that bypass storage systems for intermediate calculations altogether.
With a workflow description, one can more easily associate monitoring and telemetry data with the workflow. More distant is the integration of this information into resource allocation and scheduling systems. Data about how workflows utilize the storage systems will eventually help to develop and train adaptive and more intelligent systems. [Dorier et al., 2015] identified four critical challenges related to workflow management: 1) data dependencies 2) interchangeability 3) data locality and task placement and 4) resilience.
Workflows offer opportunities to anticipate to some extent when certain resources are going to be accessed. As a result, individual tasks within a workflow can be scheduled to minimize interference with other related or unrelated tasks. Recent work has shown that the value of I/O telemetry is greatly increased if it is analyzed together with broader contextual information [Lockwood et al., 2017].

3.4.2 Reproducibility & Replicability

Aiming for reproducibility is generally considered an imperative for good science [DFG, 2013; Pouchard et al., 2018]. The requirements and motivation for funding in support of reproducible research, however, vary from domain to domain. As such the challenges to reproduce results in high energy physics or astronomy often can be attributed to instruments and experimental setups being unique, while computational sciences and simulations face numerical challenges and inherent non-determinism as a result of parallel execution [Ludwig and Geyer, 2019]. [Pouchard et al., 2018] lists multiple factors rendering reproducibility challenging, including the technical platforms and systems environment, software dependencies, application configuration as well as input data. Thus declaring workflows is an important part of documenting but also automating the replication of results while maintaining portability across different compute platforms.
As science is increasingly also relevant to inform public policy, it is essential to lend results credibility by adding processes to enable independent researchers to replicate and reproduce results. [Ludwig and Geyer, 2019] point out that challenges to reproducibility unfold on technical, operational but also legal levels. Across scientific domains, the preservation and publication of artifacts such as primary data [DFG, 2013; Pouchard et al., 2018] and applications ideally licensed as open-source [Sonnenburg et al., 2007; Pouchard et al., 2018] is understood as an important cornerstone to reproducible research. At the technical level, this requires the operation of services for data management and storage systems such as long-term archives. Examples include global initiatives such as the Earth System Grid Foundation [ESGF, 2019].
[Pouchard et al., 2018] also note that as experiments and data analysis is growing in complexity, manually capturing meta-information necessary for provenance tracking and reproducibility becomes unfeasible. This is why initiatives such as the European Open Science Cloud [EOSC, 2019] are trying to establish virtual laboratories that allow to include workflows in part to automate metadata and provenance capture, which are then also feeding into various data catalogs.

3.4.3 *Data Management & Provenance*

In many scientific contexts, projects are encouraged to thrive for the implementation of the so-called FAIR principle [Wilkinson et al., 2016]: Data should be *findable*, as in identified by unique identifiers and disseminated into search platforms. It should be *accessible* through standard interfaces and protocols using their identifier, often this is relaxed to mainly metadata also for cost and logistical reasons. Data and metadata should further be *interoperable* by adhering to formal knowledge representations and by including related (meta)data using qualified references. Finally, data and metadata should be *re-usable* by including licensing, provenance information, as well as by meeting standards established in their respective scientific domains.
Long-term archives are an important conservatory for gathered knowledge which also serves future unanticipated applications. An interesting example, for this, is the MEDEA project, a US/Russian cooperation established in 1993 as one of the first after the Cold War, in which intelligence satellite data and others were unclassified and shared to better understand climate change [NOPP, 2009].
But many scientific domains suffer from physical or economic constraints, which allow only a single shot to record data and thus require integration with long-term archives. In particular, this is true for astronomy observations, but also high-energy physics experiments. Here complex workflows routinely filter and select data that is deemed to be most valuable for preservation.
An increasing amount of data expected to be generated and analyzed also poses challenges to data discovery. Therefore, many different initiatives are also revolving around generating and populating data catalogs.
Opportunities to exploit workflow knowledge also exist for keeping track of large amounts of data. Traditionally, researchers would come up with their own schemes to arrange data in hierarchical namespaces as they are offered by file systems. As data is shared across institutions by ingesting data into systems such as the Earth System Grid Foundation [ESGF, 2019], the Basic Local Alignment Search Tool (BLAST) [NCBI, 2020] or the Materials Genome Project [Jain et al., 2013; MGI, 2011].
[Miles et al., 2008] also note that workflow automation is critical in keeping track of provenance information. For a defined workflow, it is much easier to automatically attach metadata about experiment parameters and to create a dataset than expecting users to invest the effort to attach this information themselves. As a result, some meta storage services try to consolidate across different more specific storage and database services as can be observed with [iRODS Consortium, 2019] or Distributed Asynchronous Object Storage (DAOS) [Intel et al., 2014].

3.4.4 *Multi- and Transdisciplinary Research*

Finally, there are some trends towards more cross-disciplinary research in the computational sciences. Models from the geosciences, biology, and economy increasingly are being coupled to inform policy or investments. But the interdependence also improves specialized model skills by using data from other fields to drive domain insights and vice versa.
Climate models, for example, start to look beyond purely physical phenomena by including models of the biosphere, while research in the social sciences is relying on calculations and predictions coming from climate

models. Similarly, in case of disaster response, for example, to wildfires, small-scale high-resolution models accounting for topography, fuel density (as in wood from vegetation or structures), and wind information to deploy fire-fighting crews [Altintas et al., 2015].
In high-energy physics, particle accelerators are being used not only to generate insight into the structure of matter but are increasingly also being used as high-precision microscopes as successfully used in non-destructive imaging technologies for biological, medical, and archaeological applications. Data generated by these instruments routinely requires post-processing on HPC resources to reconstruct an image useful for analysis [Fangohr et al., 2018].

Chapter Summary

In this chapter scientific workflows have been discussed from a number of different perspectives. Currently, workflows are mainly defined to automate the execution and mapping onto computational resources.
To do so, a common schema is to declare task-data dependency graphs to be handled by a workflow management system (WMS) or a workflow engine. By comparing more than 25 workflow systems relevant in the HPC space, an overview was compiled to capture how workflow automation is approached and how I/O activity and adaptable behavior in scientific workflows are addressed. The study shows that while many systems offer facilities for telemetry information, I/O telemetry is often not considered. While most WMS offer a mechanism to export the workflow graph, only a few are using more portable abstract dependency descriptions such as the Common Workflow Language (CWL) or the Workflow Description Language (WDL). These efforts, however, are essential to allow interoperability between different tools. Three descriptions to declare workflows are discussed in detail to provide an intuition for the spectrum of different formal approaches to describe workflows as they will be later revisited.
The chapter concludes with use cases where scientific workflows are anticipated to become more important besides automation and resource utilization. In particular, coupling workflow and data management with efforts for accessible and reproducible research seem to become more relevant. While this introduces even more conflicting requirements to a diverse and opinionated landscape of workflow tools, high-level workflow descriptions ultimately benefit from these ongoing consolidation efforts. When in the past, a workflow system stood in isolation to achieve narrowly defined automation, increasingly storage systems that do not take into account longer-term temporal relationships, such as exposed by existing workflow systems, seem unlikely to meet performance expectations in an increasingly data-driven environment.

4
Related Work

The previous chapter on scientific workflows already collected the state of the art surrounding operational aspects of workflow systems used in production. This chapter collects research activities targeting storage-awareness in workflow and resource management systems and workflow-awareness in storage systems. The chapter concludes with research into adaptive control mechanisms for HPC and storage systems in preparation of Chapter 5 (Information Sources and Decision Points) and Chapter 6 (Architecture for Workflow-Aware Decision Components). The chapter is broken up into the following sections:

- *Section 4.1 discusses research activities that combine resource-, workflow- and storage management systems.*
- *Section 4.2 considers telemetry capture and collection systems as well as aggregation and visualization in relation to workflow I/O and storage systems.*
- *Section 4.3 collects research into decision components and specialized machine learning within HPC or cloud environments to improve storage performance.*

4.1 Resource, Workflow and Storage Management

Resource managers, workflow engines, and storage systems are the systems most capable to realize workflow adaptation in respect to I/O. It is notable that most resource managers and workflow engines will typically account for I/O mainly to realize necessary data staging and as a way to reduces communication. This section focuses on these systems and discusses related work from the following perspectives:

- The discussion begins with Resource and Workflow Managers, which if at all, typically consider I/O mostly to reduce communication, but rarely consider the dynamics of persistent storage technologies.
- It follows the counter perspective of workflow-friendly storage systems, which either cater toward an assumed predominant workflow or implement specialized workflow engines themselves.

4.1.1 Storage-aware Resource and Workflow Management Systems

In this section resource management systems and workflow engines are discussed which include a storage perspective. Both types of systems have been analyzed and evolved for decades, but it is not common practice to explicitly consider the dynamics of parallel distributed storage systems. The most prevalent systems, which typically only feature rudimentary workflow support, are site-wide resource management systems such as batch schedulers. Slurm [SchedMD, 2019a] or PBS [Altair, 2019], for example, allow users to declare jobs with dependencies to previously submitted

jobs. I/O optimizations targeting storage, however, are not integrated at this level. Slurm used to provide an interface to control checkpoints, but the feature has since been deprecated. To take load off storage systems there is a broadcast feature, as well as an API for data staging aimed at burst buffers. While the scheduling components can be customized via plugins, the most commonly used scheduling strategies deployed at most sites do not take storage resources into account.

More specialized workflow engines allow finer granularity, and they do care about task-data dependencies. Many workflow engines rely on established file-based intermediate storage to communicate inputs and outputs between tasks. Often, however, they are more concerned with portable and effective data management than with efficient utilization of distributed storage. Still optimization of I/O in communication and data distribution and staging are active areas of research considered by various WMS.

Especially workflow systems designed in support of distributed science and grid infrastructure spanning multiple sites and increasingly also cloud resources often have to take a more holistic perspective to address the need for data movements [Deelman et al., 2004; Altintas et al., 2004; Wozniak et al., 2013b; Oliver et al., 2019].

Some of these engines have evolved over decades, such as Pegasus [Deelman et al., 2004; 2019]. Pegasus requires users to provide a logical workflow definition which is then compiled into an executable workflow which depends on the available resources, thus allowing to address a wide variety of different architectures and sites [Deelman et al., 2015]. Commonly, logical identifiers are established for all datasets and executables to determine which resources might need replication. Pegasus breaks out some data responsibilities to separate tools such as `pegasus-transfer`, which might attempt to move data in parallel when using third-party tools such as `globus-url-copy`. But many I/O related performance decisions on high-performance storage tiers remain with the applications.

Kepler/Ptolemy [Eker et al., 2003; Kepler, 2016] has a similar legacy and is notable for providing an interactive graphical user interface which also allows direct inspection of results. The interface, however, is designed to hide complexity and system detail from users, and not to relate it to I/O behavior. Workflows are described by using so-called *directors* which coordinate *actors* which are performing computations and which can communicate with other actors by passing *tokens* (data) through so-called *ports*. The choice of director type allows to run workflows across different systems using different execution paradigms. The system is modular and provides a number of actors for common use cases, for example, integration with Globus. Unfortunately, the last stable release is from 2015.

A relatively young workflow engine designed to meet the requirements of numerical weather prediction is Cylc [Oliver et al., 2019]. Cylc can schedule workloads across multiple sites and takes care of data staging. Workflow descriptions are relatively coarse granular but offer an expressive domain-specific language (DSL) for cycling workflows with date-based time-steps, which could be taken advantage of from a storage perspective. Cylc provides an easy mechanism to export a graphical representation of the workflow in dot format but does not include I/O information.

Swift [Wilde et al., 2011; Armstrong et al., 2014] takes a focus on HPC workflows, and offers a very fine-granular approach to define workflows, closer to a programming language. Swift has two runtimes, the older Swift/K also

adopts a job-based workflow model to distribute tasks across multiple sites. The more recent Swift/T engine is based on MPI in anticipation of in situ and exascale workloads. In the language shared by Swift/K and Swift/T, the data model can become very fine-grained, with data objects potentially as small as single integers. Each Swift variable is a future, and tasks are represented as Swift functions that block on their inputs. Swift/K allows for external monitoring via the web browser, a Java Swing tool, or a command-line interface. While data staging and communication between different processing groups use optimizations to conserve I/O, associating storage I/O activity on a task to task basis is not supported.
Similarly, (Py)COMPSs [Tejedor et al., 2017] allows to define workflows using established programming languages such as Python, Java or C. While the runtime is using Java, defining workflows is especially convenient in Python by just decorating a function definition with `@task` and the COMPSs runtime will compute the result utilizing resources defined in a configuration file. The runtime accesses remote resources via SSH and creates isolated directories for each task. A number of different instrumentation backends can be used, but I/O telemetry is not related to targeted distributed storage systems.
Dask [Rocklin, 2015], is a Python-based workflow engine, which allows to transparently distribute computations across clustered resources. To do so Dask does have some understanding of common data objects such as Xarray or NumPy arrays as well as Pandas data frames, for which Dasks builds a workflow graph to decide how to split and combine computations. Dask provides a real-time interactive dashboard which produces aggregate statistics also for I/O communications.
Other workflow engines focus on optimizing in situ and in transit computations such as XIOS [CEA/LSCE and CNRS/IPSL, 2019] and Decaf [Dreher and Peterka, 2017]. Decaf allows users to specify data structures and transformations which have to be performed, when for example, moving data from a simulation to a visualization. Decaf uses MPI communicators to manage parallel execution contexts, but comes without instrumentation to keep track of I/O to storage.

4.1.2 Workflow-friendly Storage Systems

Storage systems in explicit support of scientific or business processes have been proposed before, but most systems are not built with interoperability in mind and therefore do not take a holistic perspective.
The batch-aware BAD-FS proposal for example was designed for pipeline workloads [Bent et al., 2004]. The goals were to expose control over traditional fixed policies for caching, consistency, or replication. But the approach does not match the complexity of modern scientific workflows.
Not a storage system, but an I/O middleware to delegate I/O responsibilities away from the application is ADIOS [Lofstead et al., 2008]. It introduces an alternative interface that, transparently for the application, utilizes different I/O engines based on a configuration file declaring data structures and communications. ADIOS anticipates the integration with workflow engines with a particular focus on metadata handling and the transparent integration of in situ workloads. ADIOS does, however, require application developers to port their application to use ADIOS APIs.
Widely uses in scientific contexts, with elements of application and workflow-

awareness is the Hadoop Distributed File System (HDFS) [Shvachko et al., 2010]. It is widely used but focuses mainly on the support of map-reduce workflows. The system relies, among other mechanisms, on a heartbeat mechanism to control and instruct data nodes to replicate data across other nodes to increase read bandwidth performance for often requested blocks data. But [Donnelly and Thain, 2013] note poor performance for other DAG structured workflows.

A storage system aiming to support high-level scientific workflows and data preservation across a tiered data infrastructure is iRODS [Conway et al., 2011]. It is metadata-driven and takes a fully virtualized perspective enforced by a policy system, which can be adapted to suit cross-institutional workflows or data life cycles. While it promotes the idea of reactive and proactive decisions and actions in a storage system, it is not designed for performance tiers for which it recommends asynchronous data ingest.

An approach using extended file attributes as a bidirectional communication channel for workflows, applications, and storage is followed by the Workflow-Optimized Storage System (WOSS) [Al-Kiswany et al., 2013; 2017]. This contrasts with the common practice of many workflow engines to communicate through intermediary files. WOSS instead proposes an intermediate storage system which executes parts of the workflow, and this way allows per file optimized operations and location-aware scheduling. This way a number of common workflow patterns such as pipelines, broadcasts as well as reduce and scatter operations are optimized, while also allowing applications to provide hints to control preference of caches, local storage, contiguous blocks placement or data replication. The authors note that it is sometimes challenging to balance overheads and gains, and that additional information for example about access patterns are needed to inform decisions.

[Donnelly and Thain, 2013] propose Confuga as an active storage cluster file system for DAG workflows. The system uses relaxed POSIX semantics and bypasses a number of metadata bottlenecks by taking advantage of knowledge about the workflow. Unlike HDFS the approach deliberately uses whole file replicas assuming that this is more appropriate to support DAG workflows. As no active storage hardware products existed at the time the evaluation focused on design considerations.

A larger co-design effort is DAOS [Intel et al., 2014] aiming to restructure the storage stack on top of persistent memory technologies by offering a new alternative interface to POSIX. While DAOS does not actively integrate workflow declaration, various design decisions directly support workflow adaptation. In particular, it is possible to tune for users and projects by using data containers and virtualized pools. This way allowing to improve pool performance by dedicating additional resources if required. Containers allow catering to the specific needs of different data formats such as HDF5 or NetCDF by using data structures optimized for distributed data. DAOS does not exclusively target HPC requirements but also aims to find a compromise that is attractive to business use cases. It also remains an open research question how to best inform decisions on pool size and internal data handling of containers.

[Zhao et al., 2014] implemented FusionFS which takes advantage of data locality information exposed by workflow engines to provide a file system abstraction for metadata- and write-heavy workloads. FusionFS is run in userspace and colocates compute and storage resources. Metadata man-

agement makes use of distributed hash tables and was tested on up to 16K nodes. While data is also distributed across nodes it does not necessarily reside on the same node as its metadata. The system adapts to I/O patterns and redirects data either to local or remote storage while potentially creating replicas. The decoupling of data and metadata lends the system flexibility to address the specific requirements of applications and workflows.

WaFS [Wang et al., 2015] is a workflow-aware user-level file system for the cloud. The system provides a POSIX-compatible interface and is designed to address the drawbacks of existing batch schedulers which often do not take dataflows into account. To do so, a special scheduler which aims to find a trade-off between storage overhead and cost in a measured service environment is proposed.

A scalable tempory ad-hoc file system is AdaFS/GeckoFS [Oeste et al., 2016; Vef et al., 2020]. It pools together fast node-local storage, to provide a short-term storage system which can be tailored to specific jobs. GeckoFS offers relaxed POSIX semantics and is intercepting I/O calls via `LD_PRELOAD`. Part of the proposal for AdaFS includes data-aware scheduling and data-staging which would take I/O behavior of the job into account.

Faodel [Ulmer et al., 2018; Widener et al., 2019] acts as a data backplane in support of different applications participating in a workflow by providing a unified namespace for data objects. Integration with worklow definitions and deployment tools are under investigation.

The *Tiered Data Management System* (TDMS) [Cheng et al., 2018] aims to accelerate scientific workflows while hiding the underlying storage complexity by basing data management decision on the access patterns. Users can declare access patterns such as pipeline, as well as scatter, gather, and reduce operations.

UniviStor [Wang et al., 2018a] offers a unified view onto hierarchical distributed storage with a light-weight workflow engine build-in. In particular, a locking mechanism attached to `MPI_File_open` and `MPI_File_close` which stores workflow state such us `writing/write done` or `reading/read done`. The build-in workflow manager is used to coordinate and avoid situations where a read request has to wait until a writing process completes.

An active storage solution with workflow-awareness is AnalyseThis or anFS [Sim et al., 2019]. The system is exposed to users and applications as a regular file system via FUSE, but AnalyseThis will automatically perform predefined pre- and post-processing tasks on datasets that can be configured on a folder to folder basis.

The Mochi project [Ross et al., 2020] takes a service-based approach to storage, which allows combining different smaller services into a specialized tailor-made storage system. While this allows to fit individual tasks or even a larger workflow, no service to automate workflow-awareness is included at this point.

4.2 Telemetry, Monitoring and Instrumentation

The enabling information sources for higher-level and workflow-aware decision components are primarily telemetry and monitoring information collected using different instrumentation mechanisms. In this section the discussion is broken up into three sub-areas which are complementary to some of the novel tooling introduced as part of this work:

- Instrumentation for telemetry and monitoring capture, which often neglects comprehensive I/O perspectives.
- Aggregation and collection services for analytic queries, which are needed to cope with the variety and complexity of involved systems.
- Visualization and interactive analytics, which are needed to navigate and explore the amount of information but tooling by I/O researchers is often custom-built, reducing productivity.

Telemetry integration in workflow systems was already surveyed earlier finding that most workflow systems do collect some telemetry, but I/O telemetry is often missing or insufficient to inform optimizations, an overview is provided in Table 3.1 in Section 3.3.2 (Comparison of Workflow Engines).

4.2.1 Instrumentation, Profiling and Tracing

I/O telemetry is routinely captured also in HPC environments. Workflow engines sometimes collect I/O telemetry themselves, but typically aggregate information collected from system interfaces or third-party instrumentation.
TAU [Shende and Malony, 2006], short for Tuning and Analysis Toolkit is a tool framework for profiling and tracing that supports, for example, call-paths, phase- or flat profiles. TAU uses the Scalable Performance Measurement Infrastructure for Parallel Codes (Score-P) [Knüpfer et al., 2011; Score-P Developer Community, 2019] for instrumentation. Score-P focuses on MPI communication, threading, and computation, but can optionally instrument the POSIX and MPI interfaces for I/O information.
[Adhianto et al., 2009] developed a low overhead toolkit to instrument, profile, and analyse HPC applications. Their system supports statistical sampling, event triggers and can be used to generate context trees or roofline models.
Darshan [Carns et al., 2009; 2011] is notable because multiple large HPC sites are running Darshan by default with most of their jobs. Darshan captures I/O activity related to the application and I/O library layers in an HPC stack with very low overhead. In particular, Darshan can instrument many I/O interfaces (POSIX, STDIO) and libraries (such as MPI, HDF5) and can collect and aggregate its own I/O-related performance counters. Darshan offers comprehensive I/O instrumentation and profiling because multiple layers are captured.
As part of the OVIS project [Brandt et al., 2009; Open Grid Computing Inc., 2019] the Lightweight Distributed Metric Service (LDMS) [Agelastos et al., 2014] which considers sub-second metric collection of network contention or shared file system open, close, read and write operations. LDMS proposes a multi-level hierarchical approach with samplers and aggregators and also tests it across tens of thousands of nodes.
Panorama [Mandal et al., 2016], proposes an end-to-end framework for modeling, monitoring, and anomaly detection in scientific workflows. A core motivation is the lack of understanding of expected and realistic behavior, which to an extent is driven by failures and anomalies. This makes automatically detecting and reacting to anomalous behavior harder as well. The system also integrates with Aspen [Spafford and Vetter, 2012], an analytical performance modeling language, used for runtime predictions when problem-size or compute resources are being changed.

Caliper [Boehme et al., 2016] aims to be a framework for performance introspection for HPC software stacks. It introduces a general abstraction layer with additional tools for analysis in mind. It also uses the concept of context trees, already familiar from HPCToolkit. The project envisions a loop to tune application parameters automatically. Applications and libraries that want to use the system require annotations before they can use it. A workflow perspective is not considered but could be built on top.

Information Representation and Aggregation: An important enabler for interoperability are standardized representations. Tools like hwloc [Broquedis et al., 2010] and LIKWID [Treibig et al., 2010] provide a standardized interface to control features and inquire about the topology of CPU architectures (initially only x86). Besides a variety of command-line tools, also human-readable or graphical representations can be generated. While CPU cache hierarchies are often relevant also from an I/O perspective, the approach does not extend to storage systems and workflows.
Originating in cloud environments, Prometheus [Prometheus Authors, 2019a] is becoming popular in HPC deployment as well. Prometheus comes with data models to support time-series data for counters, gauges histograms and summaries, and allows to collect and associate data for a wide variety of instrumentation solutions while also offering a query interface. Multiple approaches [Mandal et al., 2016; Wang et al., 2018b; Tuncer et al., 2019; ProfiT-HPC, 2017] aim to build analytic frameworks on top of existing telemetry and monitoring solutions. A particular focus on I/O systems is pursued by IOMiner, which provides a holistic interface to a variety of I/O related system-level logs and instrumentation [Wang et al., 2018b].

4.2.2 *Exploration & Visualization of Telemetry Information*

Monitoring and telemetry information is only useful when it can be analyzed and explored with relative ease. This includes tools for interactive visualization but also tools to perform aggregations and data analysis.
On the visualization side, this breaks down into *general-purpose tools* and dashboards which allow for a lot of customization on one end of the spectrum, and into *special-purpose tools* especially applicable for HPC and parallel environment and scientific workflows and on the other.

General Purpose: Nagios [Nagios Enterprise, 2019] is commonly used to collect telemetry and create monitoring dashboards for a variety of system metrics. Ganglia [Massie et al., 2004] provides a monitoring dashboard specifically geared towards clusters. While Ganglia provide overviews over a variety of nodes and services, the information is mostly targeting operators to monitor load and stability of the system.
Often used in conjunction with Prometheus for it's telemetry collection and query capability is Grafana [Grafana Labs, 2019] which adds more advanced dashboards. Grafana comes with a module system and has an active community which developes plugins for common monitoring tasks.
A graphical user interface to explore distributed traces are Zipkin and Jaeger UI [OpenZipkin, 2015; jaegertracing.io, 2020]. Both solutions as frameworks for distributed tracing provide tools to interactively explore the trace. While the tools are web-based, a workflow perspective is not available and the tools are not considering many of the dynamics of HPC systems.

HPC specific: HPCToolkit [Adhianto et al., 2009] and Vampir [Knüpfer et al., 2008] allow interactive visualization of traces and communication patterns, but do not consider application interactions or workflows. TAU [Shende and Malony, 2006] and Scalasca [Geimer et al., 2010] support topology overlays but neither relate information to a workflow perspective or include storage systems.
OVIS [Brandt et al., 2009] uses 3D visualization of racks and blades which can be colored according to a performance metric, which can also extend to service nodes as may be used by a storage system.
External tools and dashboards have also been developed to monitor workflows or WMS. Grafana dashboards, for example, have been created [Hammer Lab, 2015] to monitor Spark workflows.
[Adhianto and Taffet, 2016] is concerned with the challenges of visualizing huge call-path traces of long-running jobs amounting to potentially gigabytes of log data.

Workflows in HPC: Tools to make sense of workflow-related telemetry are often integrated into the workflow engines. Tools for visualization are provided by Cylc [Oliver et al., 2019], Fireworks [Jain et al., 2015], Ophidea [Palazzo et al., 2015], Kepler [Kepler, 2016], and Pegasus [Deelman et al., 2015]. The integration of telemetry information in visualizations is rare, and I/O information is typically only considered as dataflows but is rarely visualized.
External tools and dashboards have also been developed to monitor workflows or WMS. Dask [Rocklin, 2015] and [Apache Foundation, 2019a] feature a web-based performance and live profiling dashboards.
A number of efforts purpose and use workflow profiles [Duan et al., 2009; Pietri et al., 2014; Król et al., 2017] to gain workflow understanding or to make predictions in support of decision processes. Even when I/O telemetry is related to the workflows, only a few tools allow to relate them with the underlying storage software and hardware which is often necessary to inform I/O related optimizations and design decisions.

4.3 Decision-Making and Prediction in HPC and Workflow Contexts

Adaptable systems have to perform decisions. In the context of scientific workflows, these will typically depend on third-party telemetry information which is further discussed in Chapter 5 (Information Sources and Decision Points). Attempts to apply machine learning and neural networks for more speculative decision support in computing and HPC contexts have been considered in fields like branch prediction for decades [Calder et al., 1997; Jimenez and Lin, 2001] and have found their way into multiple CPUs. Decision-making in support of storage resources and scientific workflows, however, require far more complex models and is still an active area of research. Multiple proposed storage systems considered automatically discovering or reacting to workflow patterns [Zhao et al., 2014; Wang et al., 2015; Al-Kiswany et al., 2017; Cheng et al., 2018].
[Liu et al., 2018] proposes the introduction of smart data transform nodes that use deep reinforcement learning to find optimal transfer parameters to optimize overall throughput.
[Karniavoura and Magoutis, 2019] surveyed different decision-approaches to improve performance and quality of service with a focus on the require-

ments of cloud storage systems. They identify five main types of methods used for decision-making namely control-theoretic approaches, multi-dimensional optimization, reinforcement learning, Markov-decision processes, and policy or event-based methods. They also stress that many systems rely on predictions to decide on the most beneficial action. Only some of the discussed methods are also directly applicable to HPC deployments. A key research question when it comes to decision-making in support of workflows is also if there is opportunity in specialization. A number of performance and job analysis studies, however, suggest that there is [Guo et al., 2018; Lockwood et al., 2018; Di et al., 2019].
[Guo et al., 2018] analyzed different machine learning-based classifiers (XGBoost and random forests) for runtime prediction trained on log data. They could show that sub-setting based on application type allows improving prediction quality considerably across different metrics such as precision, recall and F1 score.

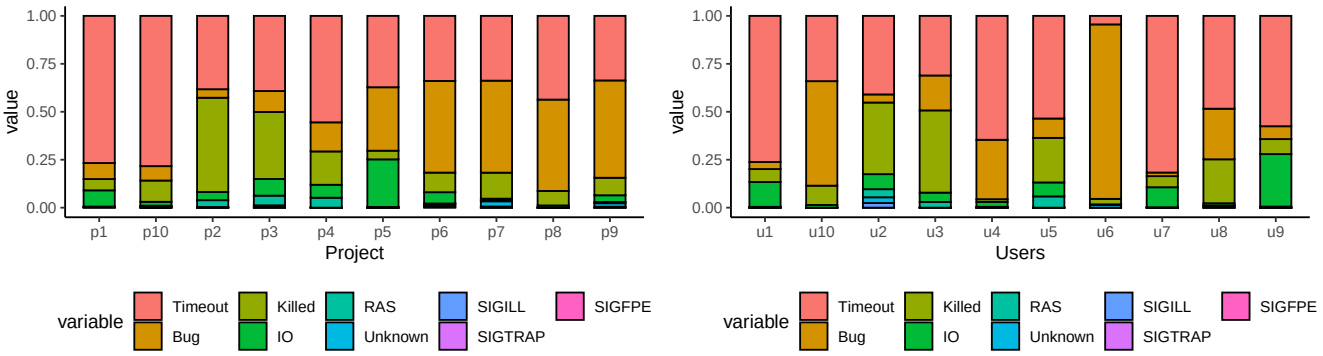


Figure 4.1: Analysis of of error code statistics collected over more than 2000 days of operation of the Mira Supercomputer at ALCF, suggest user and project specific usage patterns which might offer opportunity for adaptive behavior. Data: [Di et al., 2019]

[Di et al., 2019] performed an extensive analysis of job failures observed on the Mira supercomputer recently decommissioned at Argonne Leadership Computing Facility. The ALCF dataset spans 2001 days, totaling 32.44 billion core hours. Their dataset includes per user and per project aggregations varying for job exit codes as shown in Figure 4.1.
A similarly extensive body of job statistics suitable to make a similar claim for I/O behavior is not available. Although [Lockwood et al., 2018] collected I/O-related metrics over the course of a year at NERSC which support the case for specialization as well as the utility for self-learning/adaptive approaches.

Prediction: Various approaches are utilizing prediction to improve system performance and data center economics. [Gao, 2014], for example, explored a machine learning-based predictor to overcome the complexity and interaction resulting from feedback loops which make it difficult to model PUE with established formulas. They could show high accuracy, but note that the approach is limited by the quality and quantity of inputs, which in this case used data collected in Google data centers. The case study considered 19 features for prediction, of which many were derived using approximations or aggregations of telemetry data.
Predictions are an integral part to many decision-making processes [Karniavoura and Magoutis, 2019]. While multiple approaches produce I/O related predictions applicable to I/O systems, few combine them with workflow information and most do not consider how the prediction would be provided to and integrated into I/O middleware.

[Dorier et al., 2014] propose a grammar-based approach for spatial and temporal I/O pattern prediction in support of, for example, prefetching, caching, or scheduling decisions. A similar approach might also work for workflows but was not evaluated.
[Verbitskiy et al., 2018] present a runtime prediction system for distributed dataflows based on recurring job combinations. In particular, their model considers the interference of overlapping jobs under the assumption of I/O contention. In an evaluation of 5 different applications, the system could demonstrate predictions within 8% of the target runtime for all allocations. While the interactions of different applications are considered, the approach does not extend to considering workflows.
[Guo et al., 2018] use random forests and XGBoost for runtime prediction of HPC jobs to warn users about jobs not finishing within the proposed allocation to prevent avoidable re-submissions. The approach collects data from Ganglia and analyzes job logs and notes that predictions often perform better when sub-setting data, for example, by application. While workflows are not considered, this is promising for task- and application-specific predictors and decision components.
[Madireddy et al., 2018b;a] notes the lack of established methods for modeling I/O variability on HPC platforms. To model performance variability [Madireddy et al., 2018a] compare Conditional Variational Autoencoders (CVAE) and Gaussian process regression (GPR) on data for a Lustre deployment at NERSC. While the VAE-based approach performed better and is more flexible, it comes with additional cost for hyper-parameter training. In a follow-up study, an adaptive learning approach for concept drive was explored and evaluated against a number of well-defined I/O motifs [Madireddy et al., 2019]. This way misprediction is handled gracefully because the model internally "switches" to different prediction formulas.
[Haskins et al., 2019] propose a feedback loop for containerized applications for performance reproducibility, by taking telemetry data to feed a performance model. The system then alerts users if the performance predictions are not met using post-hoc analysis.
[Monjalet and Leibovici, 2019] evaluated different methods to predict file lifetimes based on path names. In particular, random forest regressors and convolutional neural networks have been compared.

Placement and Replication: There are also approaches to influence placement and replication decisions. Runtime Asymmetric Data-Access Driven Scientific Replication (RADAR) [Jenkins et al., 2014] proposes a data replication system which is based on I/O patterns obtained from MPI-IO tracing. Access patterns and location information are also considered by [Sun et al., 2015] for data movements, for example, in in situ applications. The approach also uses trace data and performs a case study on a real combustion-analyses workflow which finds up 3x speedups in an adaptive scenario.
A feedback loop for load balancing in Lustre is implemented by [Wadhwa et al., 2019]. While iez taps Lustre OST capacity information and applies a machine learning model to determine data placement, it does not target scientific workflows.

Learned Index Structures: [Kraska et al., 2017] propose learned index structures as replacements for B-trees to save space but also to improve lookup times, although multiple hierarchies of learned indexes might still be em-

ployed. [Wang et al., 2019] extend on the concept to allow spatial queries on learned index structures. They also anticipate that hardware acceleration might allow considering more advanced models in the future. [Hadian and Heinis, 2019] seek to find a compromise that uses interpolation-friendly B-trees.

4.3.1 Performance Modeling & System Simulation

System simulation is routinely used to inform design decisions for hardware and software also in the context of HPC. Because it is often not possible to simulate the dynamics of an entire supercomputer, typically only sub-systems are being modeled in high-fidelity. More recently, machine learning approaches that used surrogate models, genetic algorithms, and self-play have been used to accelerate and improve the results when training machine learning algorithms, but for system optimization, this is not common practice although.
System simulations with a particular focus on computer architecture and networks have been used for a long time [Fujimoto, 1990; Nellans et al., 2004; Binkert et al., 2006]. More recently approaches to model complete HPC systems are being explored. The Structural Simulation Toolkit (SST) [Rodrigues et al., 2011] is designed to inform the exploration of new architecture and aid the procurement of HPC systems. SST includes a number of processors, memory, and network models.
The Rensselaer's Optimistic Simulation System (ROSS) [Jr. et al., 2013] implements a parallel discrete-event simulator for execution on supercomputers in support of large-scale simulations of up to millions of object models. The CODES project [Mubarak et al., 2014] is using ROSS to perform HPC storage and interconnect case studies. CODES also provides a collection of system models. The Panorama project aims to integrate with CODES and ROSS for performance modeling of workflows.
[Costa et al., 2014] use simulation for a number of workflow scenarios for I/O intensive applications. The case study uses a simplified simulated storage system, synthetic benchmarks, as well as a real workflow to reduce resource spending to inform allocation and provisioning decisions.
While these approaches demonstrate the utility and practice to simulate new system architectures, workload generators and learning components for workflows are not common practice to improve and train decision components for HPC storage systems.

Chapter Summary

This chapter discussed related work in respect to research activities surrounding scientific workflows and storage systems in HPC environments. First resource-, workflow- and storage management systems with respect to *storage-awareness* and *workflow-awareness* are covered. Resource management systems, typically, feature only light-weight dependency management, while workflow engines consider detailed task-data relationships. Both types of systems consider storage systems mostly as staging areas but do not attempt to actively exploit the underlying distributed architecture of storage systems. A number of storage systems are designed explicitly in support of particular workflow types or implement a workflow engine internally to offer active storage. But most workflow-friendly storage systems are

not relevant in production environments at this point.
Various solutions for instrumentation and telemetry capture for workflows, also consider I/O characteristics. Unfortunately, a relation to the underlying storage systems in respect to the workflow execution is usually not performed.
Similarly, a variety of efforts try to automate decision-making and prediction in the context of HPC workflows or storage systems. But only few of these solutions are found in production systems. An important research question is also if machine learning-based approaches might benefit from specialization, where multiple case studies using learning techniques on job-, user- and system studies suggest that there is opportunity.

5
Information Sources and Decision Points

Decision-making requires identifying possible points of action (a decision point) and influencing factors that inform a decision (an information source). This chapter surveys the storage stack to find suitable abstractions for workflow-aware decision-making by focusing on the following aspects:

1. *Which information sources are exposing workflow intent or behavior?*
2. *Which decisions points would benefit from a workflow perspective?*

After offering an overview, the storage stack is surveyed beginning from the user perspective at the workflow- and application-level while gradually getting more technical down to the level of hardware and distributed services. The chapter is broken down into the following sections:

- *Section 5.1 starts with an overview of the hardware and software perspectives and limiting criteria to the focus of the survey.*
- *Section 5.2 and Section 5.3 work down from the user perspective covering workflow and application knowledge respectively.*
- *Section 5.4 takes a closer look on information sources and decision components exposed by I/O related middleware such as data description frameworks.*
- *Section 5.5 covers node-local operating system services and hardware down to the device level, although privileged access prohibits many optimizations.*
- *Section 5.6 breaks down the network layers and considers optimization opportunities from off-loading and re-configurable networking.*
- *Section 5.7 covers site-wide distributed services such as resource management systems as well as policy and storage services.*
- *Section 5.8 concludes with emerging approaches aiming for a more holistic perspective of system and workflow telemetry.*

"The greatest scientific discovery was the discovery of ignorance. Once humans realized how little they knew about the world, they suddenly had a very good reason to seek new knowledge, which opened up the scientific road to progress."
– Yuval Noah Harari

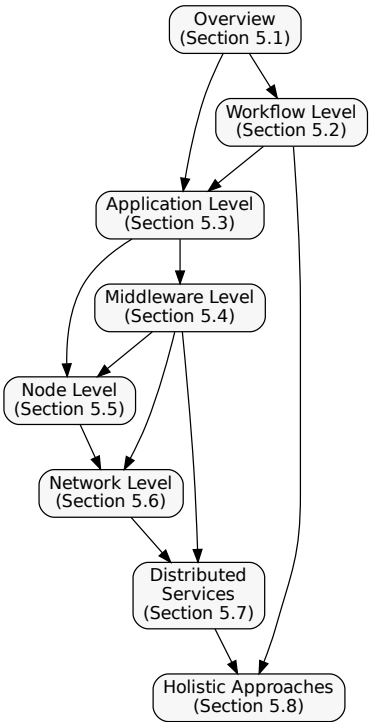


Figure 5.1: Relationship of the section of this chapter.

5.1 Overview

Chapter 2 (Background) and Chapter 3 (Scientific Workflows in High Performance Computing) introduced the structure of compute and storage systems on the one hand and discussed scientific workflows on the other. This chapter surveys the pathways data are taking as they are moved and accessed by users and their workflows. As no comparable analysis could be found, a particular goal was to identify potential *workflow artifacts* collectable from telemetry sources but also *decision points* along these data paths. To approach this systematically, the stack is traversed starting with the user/application perspective at the top and follow data through the various layers and services to the devices at the bottom (compare Figure 5.1).

(a) Information Source

(b) Decision Point

Figure 5.2: Throughout the thesis information sources a) are depicted by a little heartbeat (indicating activity), and decision points b) are depicted by a little gate or a switch.

The chapter concludes with a number of approaches aiming for a more holistic view. The survey constrains itself to the workflow and storage perspectives and focuses on two different aspects:

- *Information Sources* describe opportunities and subsystems which hold information that are useful to reconstruct the behavior of a workflow. As such they are candidates for instrumentation, or they already offer facilities for telemetry collection. In general, it is tried to limit this to information sources that are plausible to obtain, instead of everything which might be theoretically possible.
- *Decision Points* mark points which allow to inject interventions along the path data has to travel when considering a workflow perspective. Traditionally, decision components would mostly rely on very local information and only rarely on long-term temporal information such as the history of observed behavior for an application or workflow.

Because workflow intent is usually only sparsely expressed in *explicitly* machine-accessible form, it often becomes necessary to find and combine artifacts that allow to deduce a user's intent. In many cases, these approaches can be distinguished in terms of whether they are leaning towards an *explicit* in contrast to an *implicit* perspective:

- *Explicit* workflow-related information include, for example, the workflow definition, however, it is usually incomplete as performance implications depend on the system and many other contextual factors.
- *Implicit* workflow-related information can include monitoring, profiling, traces and logs. They will often come without information about their relationship to the workflow. Ideally, when gathering enough information the workflow and its implication can be reconstructed, even if no explicit description of the workflow exists. In practice, it will typically not be affordable to do so because of logging overheads.

The complexity and fast-paced evolution of the involved systems does not allow for Figure 5.3 to be exhaustive. Instead, the selection of information sources and decision points mapped out serves as examples with concrete technologies and products being assumed to be superseded by new technologies.

Finally, technical systems can be subdivided into a logical and a physical perspective as illustrated in Figure 5.3. The logical perspective is usually constructed in software which is built on top and spread throughout the physical hardware systems that form a data center:

- *Software Perspective (right):* Most interactions are occurring with software components. Often they can be changed also after a system is deployed, although traditionally not all software components are kept accessible for updates. Most of the changes considered as part of this thesis are software components, but it is anticipated that the usefulness of the approach outlined in Chapter 6 extends to hardware components as well.
- *Hardware Perspective (left):* Logical software components are realized on top of hardware. However, many functions which used to be software are today integrated as hardware, to speed up operations, to conserve energy or to move to a smaller form factor. With specialized hardware such as accelerators for encryption/compression or machine learning, but also with affordable FPGAs, possibly being integrated along data pathways, considerations for hardware components are mandatory. The hardware perspective is also of particular importance as it is the determining factor for information and data locality.

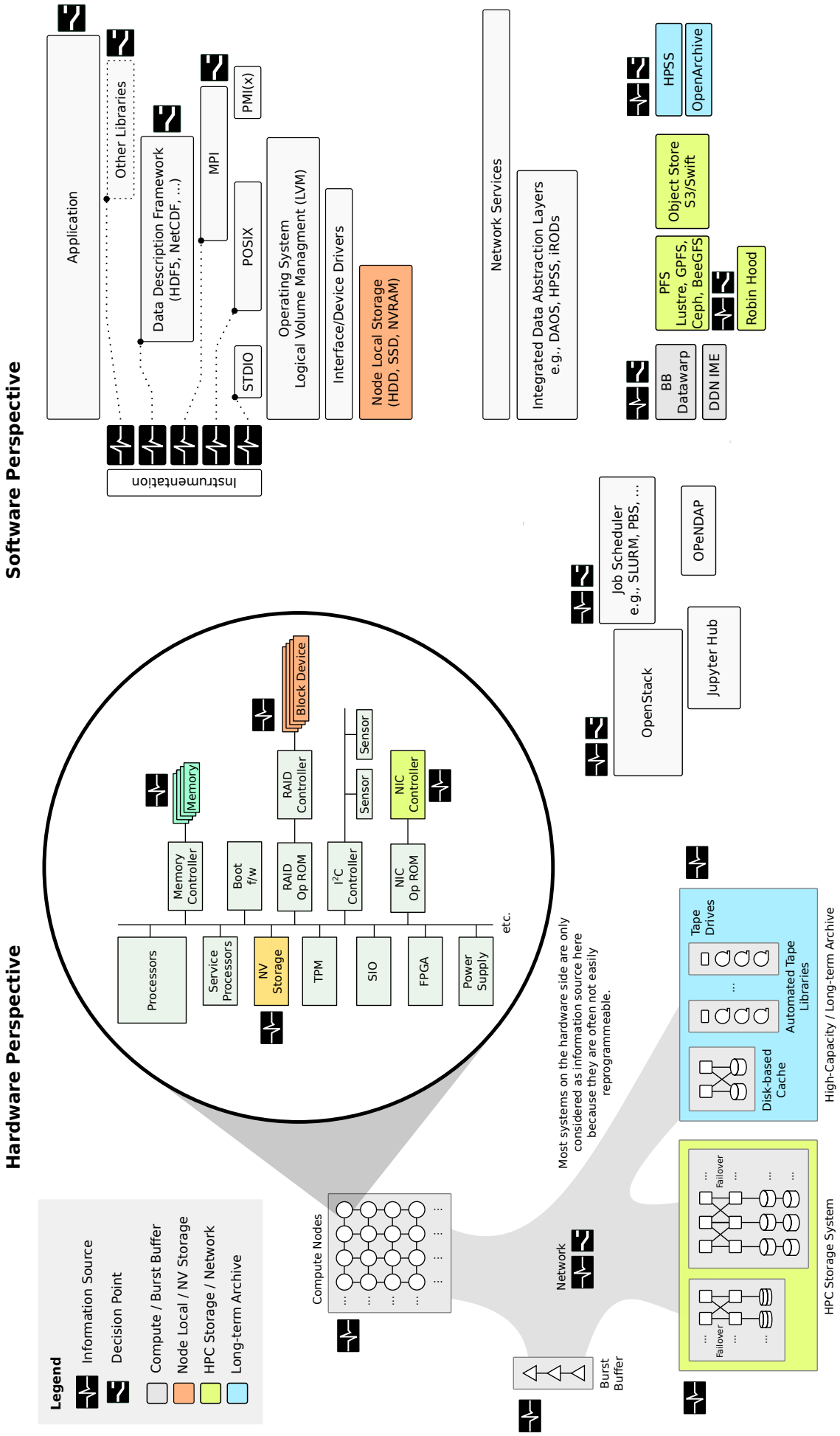


Figure 5.3: Schematic overview illustrating how HPC hardware and software components, applications, and workflows relate to each other.

5.2 *Workflow and User Level*

A promising starting point for finding workflow intent is the declaration of a workflow. Somewhat simplified, at the workflow level, all information about interactions a user could know of or anticipate might be described. This is notable because many optimizations that are applied when manually tuning an application or a system, are only possible because they can reliably anticipate a future state or interaction of the system. Technical systems don't offer similar cognitive capabilities, but a workflow description can be seen as an approximation for such user knowledge.
Unfortunately, workflow information is often not explicitly defined in a machine-friendly format. Even when workflow management tools are used, much relevant additional information from a storage perspective about the underlying semantics will be omitted. In many circumstances, these semantics also vary across different systems and over time so that the additional effort is often considered not worth it. Still, users increasingly do declare their workflows as they have to orchestrate large amounts of resources or because they seek that their results are easily shared and reproduced by colleagues. So assuming some notion of the workflow is being defined, it is useful to consider the following two perspectives:

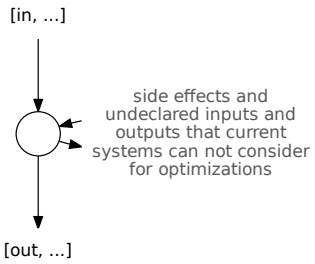


Figure 5.4: Most workflow declarations work by defining inputs and outputs, which are then used to generate a data dependency graph. Side effects and other files such as logs and diagnostic files will typically remain undeclared.

- *User/Explicit Perspective:* From a user perspective workflow intent is revealed through workflow definitions. This can happen in workflow engines, where the whole workflow is described. It can also occur near individual tasks of a workflow, where a part of the workflow is declared in more detail.

- *Technical/Implicit Perspective:* Decisions are made throughout the storage stack, but usually, the information which is being considered is limited to information that is more local to a service or the physical system which accommodates the service.

Figure 5.4 illustrates how these two perspectives can relate to tasks in a workflow. Most workflow engines require users to define inputs and outputs, and how these relate to other tasks. What is not declared are side effects of executing a task, which impact the system but are not directly relevant from the workflow perspective.

Contextual Embedding and the Workflow Graph: Most workflow engines construct a directed acyclic graph (DAG) to better plan and determine the order in which to execute tasks. This graph can also be useful for a storage system to inform placement and data staging/scheduling decisions. Different workflow engines use a variety of mechanisms and sometimes an explicit export function to get a graph representation. For some workflow engines, the graph can only be reconstructed from tracing or log data.
A possible strategy to influence workflow behavior requiring no or only minimal changes to workflow engines is the injection of additional tasks to perform additional data staging, transformation, or cleanup. In fact, some workflow engines offering features such as namespace management across multiple sites to perform data staging automatically are using this mechanism but typically would not expose interfaces to third-party orchestrators.

This notion of adding tasks out of technical necessity is also acknowledged by some systems which distinguish abstract workflows from "executable workflows" [Deelman et al., 2015]. An executable workflow takes concrete resources into consideration.

Performance Metrics and Feedback: Most workflow engines will keep track of failures because this is to be addressed by the scheduler and has consequences on subsequent execution of dependent tasks.
While many workflow engines do collect performance information, they are typically not accounting for the complexity of I/O and storage systems. Most collected statistics are broken down by job, process, or tasks. Workflow tools could offer interfaces to query I/O statistics for whole pipelines or aggregate for similar tasks, but this is not commonly found.

Details on Tasks and involved Applications: Many of the side effects and undeclared information is hidden within tasks. Often a task maps directly to the execution of an application binary, but it is also common to have nested workflows. Some workflow descriptions provide concepts for nesting, which allows extending the workflow graph with relative ease as no new concepts are introduced. However, there are also workflows which nest different workflow engines often leaving both the higher-level and the lower-level workflow engines without means for mutual introspection. An example is illustrated in Figure 5.5, which highlights decisions that might be made at the high-level (left), with decisions that are only known to the sub-workflow (right). The challenge here is that both workflow engines might operate in ignorance of the other.
A variation of this scenario is particularly common for the interaction between resource scheduling systems and workflow engines which act as meta-schedulers.

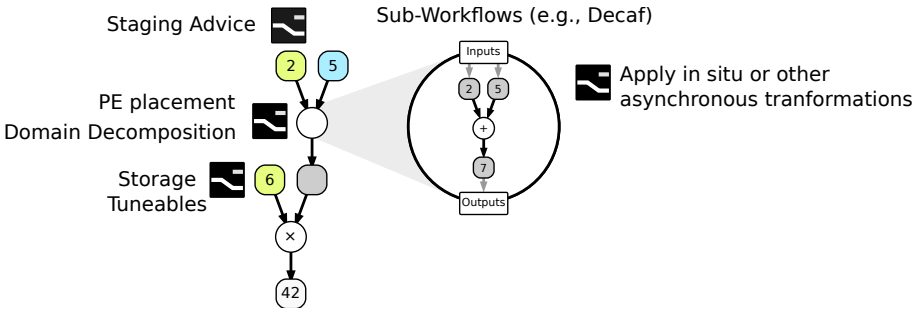


Figure 5.5: Information sources and decision points in a workflow task-data dependency graph. Information of a sub-workflow might not be available to the higher-level workflow.

Details on Domain Decomposition and Data Persistence: Of particular importance in HPC environments is the domain decomposition of compute tasks as well as data. Often there are conflicting requirements between the domain decomposition while operating on the data in contrast to the domain composition when storing data persistently. In addition, workflow engines often operate on a file system abstraction as this is reliably available across different sites, but this perspective will routinely disregard optimization opportunities of tasks and applications which only access subsets of data. A key research question is therefore how to inquire and integrate information about workflow and task characteristics to inform decision-making.

Introducing Advisory Services: The Decision Aid Propagation Service (DAPS) which is introduced in Section 6.5 provides a unified interface to publish and subscribe to these kinds of derived workflow-related decision

aids. Section 6.3 (Actions & Decision Points) complements this advisory delivery system by abstracting decision components to make them more easily exchangeable.

Workflow and User Level Summary

Information sources and *decision points* at the workflow and user level, typically, take a high-level perspective which may describe the necessary processing steps as task-data relationships to varying degrees of detail. If only a high-level workflow is automated using a workflow engine, many input and output files remain undeclared. If a fine-granular workflow is described, coverage of inputs and outputs can be very good, but technical side effects still remain undeclared. This is especially true for technical I/O activity used for coordination and data staging, which are often not accounted for even if workflow engines offer telemetry integration. In addition workflow engines may be nested, which in many cases leaves at least one of the involved orchestrators ignorant of the other.
In principle, users and workflow implementers have large freedoms to address any shortcomings, but on a practical level limited interoperability between solutions and the range of involved technologies render such co-design activities unfeasible for most users. A more convenient abstraction interface to inquire about workflow dependencies, performance information, system details would be needed.

5.3 Application Level

The range of information available at the application level is highly application-dependent. A wide range of decisions can be performed at the application level, but in practice, many will be delegated away to libraries and middleware. For simplicity, some non-I/O-issuing libraries will be listed in the application level, although this may be inadequate when taking a different perspective. Usually, workflow context at this level is not considered beyond the current task and its operational parameters.
As it further complicates an often already complex software architecture, traditionally, an application would not be designed with a cross-cutting contextual embedding in mind. As such while a lot of undeclared information materializes at the application level as illustrated in Figure 5.6, only a few interfaces are offered to exchange and query this information. Instead, applications will routinely use specialized formats for configuration.
In this section, workflow artifacts that can be collected at the application level are discussed. A common information boundary is found between the workflow perspective and the application perspective for which it is useful to consider the following:

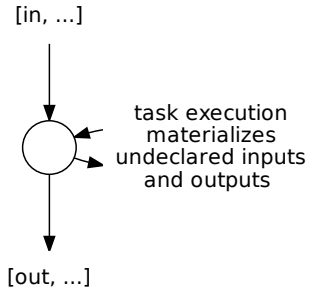


Figure 5.6: On the application level, many of the undeclared inputs and outputs can be obtained.

- A workflow engine is likely to inject information or references about the current task into the execution context of the application. If this is the case, the application can, in theory, inquire about its embedding. It can also pass on this information to middleware and services.

- The execution context, in particular available resources, impacts domain decomposition, and communication behavior. Affinity for this may be lost at the higher-level perspective, as well as at the lower level when responsibility is delegated to middleware.

- An application will effectively obfuscate internals which could offer an opportunity to optimization, compare Figure 5.7. This is especially true for applications requiring compilation, which are the norm in HPC.
- Within the application, there is additional information about the interpretation and semantics of data. This shows in memory layouts, but also on procedures and methods which are defined to manipulate and analyze data. But extracting and utilizing this information is hard, but could become easier when domain-specific languages are used.

Partial Workflow/Task Knowledge: A single application usually only has access to information about parts of the workflow. Being executed directly or indirectly by the workflow engine which holds a more complete high-level workflow, it should be possible in many cases to inquire about the higher-level workflow. In practice, this is typically not the case because not all workflow engines inject contextual information into the environment, for example, because the workflow has dynamic elements, or because no bidirectional interfaces for such communication are established.

Application Knowledge: An application in many cases can be considered an opaque task or sub-workflow to the workflow engine. As such, in many cases, only an interface to which arguments can be provided is exposed. Interestingly, in many situations, sub-components of an application also remain black boxes to the application itself as responsibilities are encapsulated into modules or delegated to third-party libraries as is illustrated in Figure 5.7. While source code offers introspection in theory, software is rarely written in ways that would make it feasible to consider automatic extraction of data semantics and operations. The proliferation of libraries, domain-specific languages or annotations might expose more attractive interfaces for these use cases in the future.
Similarly, for files created by an application, additional metadata could be attached to files that would leak their intent which can aid staging and tier selection decisions. Granularity would not need to be limited to files or objects but could extend to substructures such as datasets or even parts of datasets. I/O operations accessed through APIs offered by the OS or by distributed services are only accessed indirectly via middleware. There are however many exceptions where an application will inquire with the OS or client module of a storage system before interacting with a middleware. In most cases, this is performed to achieve resource affinity.
Library interaction becomes especially complicated as all the operations are executed in a parallel environment. Sometimes, this can be performed transparently, but in most cases, parallel logic is split by rank carefully engineered to result in some desired emergent behavior. Many applications will perform some inquiry about the execution context to pick parameters with which underlying middleware is driven.

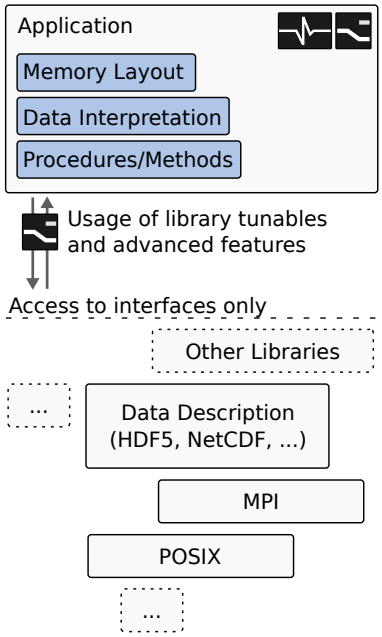


Figure 5.7: Applications commonly rely on third-party libraries for often-used functionality and data structures. Much information can be collected within the application itself, although this information extraction is likely closely embedded with compilers. Similarly, many decisions can be influenced at this level either in the applications itself or through tunable passed on to middleware.

Argument Interpretation: Exposed to the workflow engines is usually only an interface through arguments or configuration files. The workflow engine in most cases will only be able to populate these for a few predefined cases. So while the application holds the logic on how to interpret many of these inputs, there is typically only little structured information about the relationships between options. As such applications allow to be configured using contradictory or nonsensical inputs, typically resulting in invalid

executions. Sometimes illegal inputs are sanitized or prohibited by the application. Some input parameters are also obtained implicitly, for example when domain decomposition is automatically calculated from available resources in an allocation, such as number of nodes and cores.

Data Semantics and Domain-Specific Languages: While some structural information might be passed onto lower levels, the data semantics and the meaning of data often is not. Semantics further depend on the application domain or even the context within a discipline. Domain-specific languages allow users to specify their intentions more conveniently but in a formal way. Domain-specific languages are usually defined around some integral concepts and operations of a certain domain to allow less verbose interactions than would be required with a general-purpose programming language. These can expose valuable information about the interactions of data and allowable operations on the data.
A domain-specific language in many cases can also be beneficial from a performance perspective, as most users would not optimize the application to the same extent or prefer an effective but simple implementation over a more complicated efficient implementation. In many cases, this is incentivized by limited resources for development efforts.

Domain Decomposition, Topology, and Resource Affinity: A variety of information about the execution context and resulting decisions can be obtained at the application level. Based on the available resources and the topology, an application (or a middleware) will decide on a domain decomposition and a mapping of processing elements (PEs) to the topology. Placement of compute PEs and potentially I/O PEs can have a huge influence on memory layouts, communication patterns, contention, and performance. Here a common conflict is a mismatch of *in memory* and *on storage* data layouts. To an extent, this mismatch is bridged and performed transparently for example by collective operations which might reshape data before it is written to persistent storage. These operations are most commonly implemented by data description formats such as HDF5 or NetCDF. Workflow intent, however, is rarely passed in these contexts.

Purpose and Structure of Files and Objects: The application potentially can have information about the file layout but in many cases will merely know to access certain datasets. In principle, the application usually has the most detailed knowledge about the structure of persistent data. But because self-describing data formats are commonly used, an application might not have information about the precise layout of in- and output data. Until recently, most storage systems and most low-level middleware would not attempt to exploit data structure if the service offered a general file system interface.

Instrumentation and Profiling: Naive profiling might simply collect time to completion, to get a performance estimate, which includes I/O times but does not break down how time was spent performing I/O. In combination with other systems, I/O activity can be correlated to the application in post-mortem analysis. When using libraries as proposed by [Boehme et al., 2016], more fine-grained instrumentation can be achieved, but applications have to include annotations that control instrumentation at compile or at runtime. So far, mostly instrumentation mechanisms that require the active participation of developers have been discussed. Application profiling allows to

independently gather performance information without modifying an application. From a workflow perspective, instrumentation at the application level likely poses logistical challenges as complex call-graphs with many non I/O related activity would need to be analyzed. How to achieve this automatically is a standing research question unlikely to be solved except for some special or very common cases, consequently encouraging the use of middleware. A wide range of approaches to instrumentation exist as discussed in Section 4.2 (Telemetry, Monitoring and Instrumentation), but most general approaches will take advantage of the compilation process or calling conventions to libraries.

5.3.1 *Compilers*

Users in scientific computing routinely have to compile applications from source code. While it may be often considered an annoyance that applications and libraries have to be compiled on each system they are supposed to run on, it also opens a number of interesting opportunities. One obvious benefit is the option to consult the source code for technical details, but the compilation process also offers to inject a certain system- or user-specific optimizations with minimal impact on runtime performance.
Figure 5.8 illustrates potential leverage points to adapt applications and workflows to gather intent, add instrumentation, or apply optimizations. The example assumes GCC, but other compilers such as developed as part of LLVM sometimes offer a more modular architecture to apply sophisticated optimization on the compiler level. Examples for such modifications include *task/intent extraction*, *instrumentation*, as well as *optimizations*.

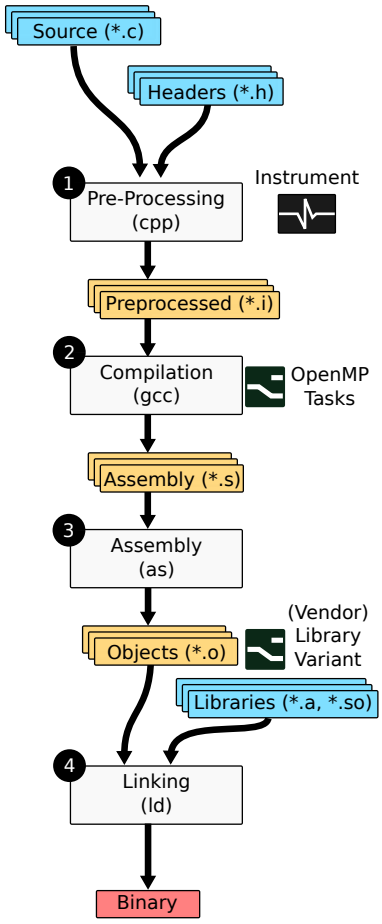


Figure 5.8: There are also information and optimizations which can be extracted or applied at compile time. Examples include adding instrumentation, extracting information about task parallelism or switching from one library variant to another.

Intent extraction at a very fine granularity is typically not possible automatically in absence of supporting annotations. However, programmers are already adding similar annotations in support of parallelization and for testing. Examples for this include OpenMP pragmas, assertions, or loop invariants. More promising, however, are intermediate representations, or intent expressed using domain-specific languages. OpenMP is an abstraction to help with the parallelization in shared memory environments. OpenMP uses pragmas which allow influencing how, for example, `for`-loops in an application are handled by the compiler. In particular, OpenMP is used to automatically execute the program in question using threads for parallelization. In more recent developments, the notion of tasks was added to OpenMP, which decouples elements that need to be computed more explicitly from the remaining application. As such a task might be a routine that compresses or aggregates data, it can also be useful for approaches like active storage.

Instrumentation in some cases can only be applied at compile-time. Though in many cases approaches using shared libraries, for example via preloading, are often more appropriate, as this offers greater flexibility later on. This seems also to apply to many workflow-related use cases that rely on slower phenomena which barely profit from the marginally lower performance overheads of compile-time instrumentation.

Optimizations can also be applied at compile-time, this includes I/O optimizations. ADIOS, for example, builds application specific I/O runtimes based on a configuration file. It seems plausible, to tap into a site global expert system to overrule or improve user optimizations. It is important,

to communicate if this is being done to avoid confusion and to ease debugging. *Datatypes and data structures* are two common targets for optimizations. A simple but common optimization, for example, is switching from an array of structs (AoS) to struct of arrays (SoA) representation and vice versa. In many cases, this allows to better exploit data locality. Which representation is more appropriate depends on the calculations which are performed on the data in question. Finally, *library variants* are relatively common in HPC environments. For example, different implementations exist for common interfaces such MPI or BLAS. Often vendors modify existing implementations such as OpenMPI or MPICH and apply product-specific optimizations.

Application Level Summary

Information sources and *decision points* at the application level are manifold, but many I/O decisions are delegated to middleware. From a workflow perspective, tasks and applications often coincide with binary applications. As these are traditionally hard to inspect and opaque for the workflow, I/O behavior and undeclared input and output files can be obtained at this level to complement partial workflow knowledge. Still, a lot of intent about the usage such as the purpose of files, the structures of data or memory access semantics typically materialize at this level. In practice, most codebases cannot be automatically scanned or interpreted to extract such intent, which is why domain-specific languages or special purpose libraries gain popularity. Instead of analyzing the code, performance implications are often best captured by instrumenting executions. Various methods exist for this, but a custom code that is not relying on common libraries will suffer from lower performance while producing a lot of unintelligible instrumentation data.
The previous sections focused on functionality and workflow-related information embedded in the actual application. But common functionality useful across different applications is usually exported into external libraries and middleware. From the application I/O perspective, these are especially relevant because:

- Library usage marks delegation of responsibility to external libraries and services, which are attractive targets to apply optimizations or instrumentation. Discussed in more detail in Section 5.4 (Middleware Level).
- Loading libraries can result in lots of additional small I/O accesses, which is often overlooked and thus not optimized by users. There is potential to learn and pack an image file or a container which prevents this I/O activity to impact shared storage systems.

5.4 Middleware Level

With I/O dispatch being often delegated by applications, common APIs offered by the operating system or middleware, such as MPI, HDF5 or NetCDF, are attractive targets to capture activity but also to apply optimizations.
The workflow context, unfortunately, is rarely actively passed to middleware, although there are some opportunities to inquire about the active workflow by analyzing the environment as some workflow engines populate environment variables with task or workflow identifiers. Middleware

consisting of multiple components spread across multiple systems, however, requires that this context is passed actively.
In this section, the focus is on middleware and components of middleware residing on the same compute nodes as the calling application as illustrated in Figure 5.9. The application is using among other libraries a *data description* framework such as HDF5 or NetCDF. Operating in a parallel environment, both the application but also the data description framework are likely to rely on additional middleware such as MPI for *coordination* and collective I/O. In addition, both data description and communication libraries have to *serialize* and potentially *compress* data.

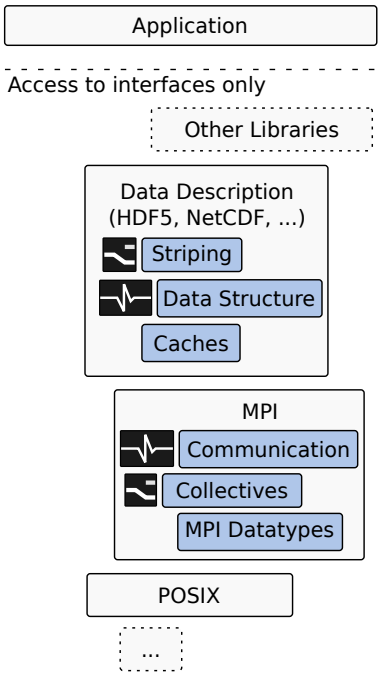


Figure 5.9: Applications commonly rely on third-party libraries for often-used functionality and data structures. The list of information source and decision points marked here is not exhaustive. This example demonstrates that various workflow relevant decisions are made in I/O middleware such as HDF5/NetCDF and MPI. Unfortunetly, the various middleware have to rely on heuristics which are usually ignorant of the workflow.

Data Description: Middleware and libraries for data description such as HDF5 and NetCDF are popular with scientists for a variety of reasons. Data is portable and easier to use by collaborators if data description frameworks are used. To still allow advanced users to fine-tune performance, different optimizations implemented by a middleware are typically exposed through so called tunables. These tunables are good candidates for pluggable decision components which could be optimized for a particular workflow or data center. Unfortunately, a lot of existing middleware does not yet define interfaces that would allow doing so.
Data description formats are also directly relevant from a workflow perspective for two reasons. On the one hand, metadata describing the data can be added automatically. On the other hand, access to output files can be optimized for subsequent use in consuming tasks. Often workflow engines will operate on file granularity, even though it could be advantageous to only consider subsets of data. MPI is a special case because it is an I/O library primarily used by applications for coordination, but it also offers file I/O interfaces. In most cases, however, applications would not use MPI-IO directly but instead, rely on a data description framework which would optimize parallel I/O using MPI-IO.

Serialization and Compression: A common task every application or I/O library has to perform before transmitting data is to turn it into a serial sequence of data, commonly referred to as serialization. A variety of programming languages come with standard libraries that offer serialization functionality even for complex data structures out of the box. Examples are Python's pickle or Boost Serialization in the context of C++. As many software projects are not limited to a single programming language, meta-serialization frameworks have been developed such as protocol buffers [Google, 2008]. Simulations, for example, might be routinely implemented in Fortran or C/C++, while post-processing increasingly relies on higher-level programming languages such as Python. Whenever an application is serializing data before writing, there is an opportunity to avoid potential follow-up costs for consumers. In the HPC context, this applies as well, but such fine-grained granularity is typically not feasible to consider due to increased data management overheads and lack of information to base such decisions on. Storage libraries such as HDF5 typically would implement serialization on their own, as the data format was carefully constructed with certain serializations in mind.

Coordination and Communication: The Message Passing Interface (MPI) is used by many applications in HPC to coordinate communication and parallel execution across multiple processes and nodes. As such MPI offers

some cues about the domain decomposition, but more importantly often has fairly complete information about the communication patterns. Even more relevant from a storage perspective is the MPI-IO interface, which can be used to realize parallel and collective I/O operations to files. Using the interface also helps with avoiding interleaving reads and writes, which provides favorable consistency guarantees.
For messaging as well as for I/O, MPI allows registering custom composite data types, which considerably simplify using send/receive and read/write calls. MPI datatypes resemble C structures, and variable size or schema-less objects are not supported unlike with other serialization libraries. There are, however, libraries that implement this on top of MPI. There are multiple competing MPI implementations, which typically provide the foundation for vendor optimized MPI libraries that can take advantage of non-standard hardware features offered by different product lines. As a result, most MPI implementations adopted underlying interfaces such as the *Process Management Interface* (PMI) for scalable coordination between MPI initialization and scheduling systems.

5.4.1 *Software Stack Complexity and Optimization Opportunities*

The attractiveness of targeting middleware but also the complexity associated with HPC software stacks is best demonstrated using an example. Figure 5.10 illustrates the software dependencies graph of the HDF5 library with support for the Blosc [Alted, 2010] meta compressor. Four different types of dependencies are highlighted: *Data description* (blue), *compression libraries* (green), *middleware for communication* and *coordination* (turquoise), and *hardware affinity* (orange). Each dependency introduces an opportunity to instrument or influence activity which later impacts the storage system. Most of these libraries are operating relatively isolated of each other, but from a workflow perspective, it may be desirable to influence multiple libraries simultaneously.

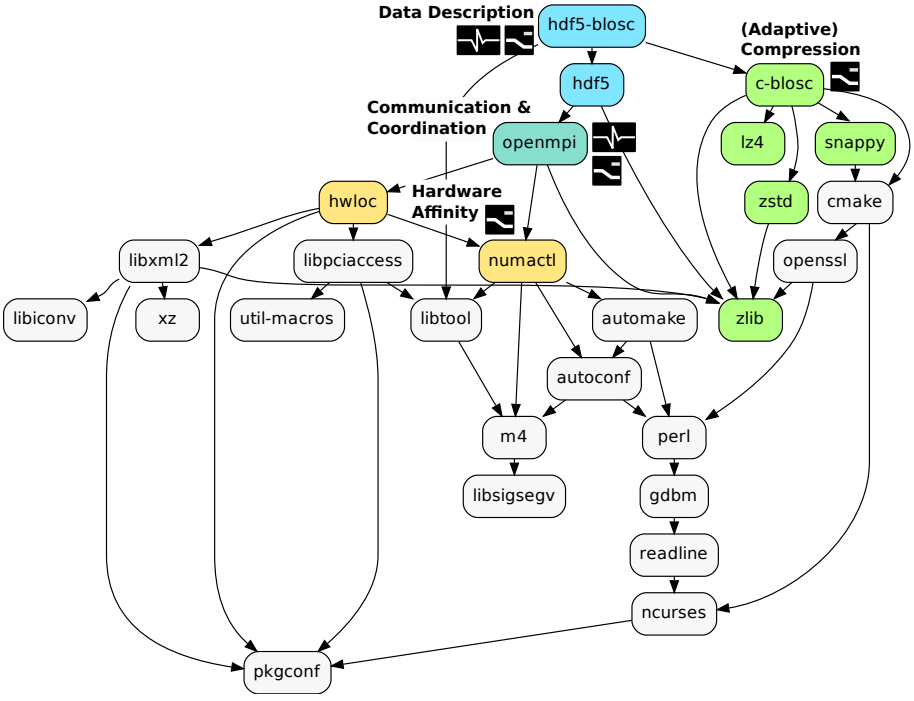


Figure 5.10: Software dependencies as reported by Spack for HDF5 with Blosc support. Both HDF5 and Blosc are commonly used by HPC applications, but they also they rely on additional third-party libraries to perform, for example, communication, efficient computation, or adaptive compression. To do so, the different libraries are typically relying on locally available contextual information such as the hardware affinity, environment variables, or performance counters.

Dependencies using the operating systems standard interfaces are not in-

cluded in this dependency graph. A challenge to changing the behavior of software dependencies is rooted in the complexity of managing and compiling the software tree provided by HPC sites. Initially, traditional package management did offer little functionality to support different library versions or different providers for the same interface. A particular problem here is that changing a dependency downstream often requires recompilation and re-linking dependencies upstream. As a result, operators would only be able to offer support for a carefully selected range of packages. More recently, tools like Spack help to automate the software build process, making it more feasible to consider more experimental changes to commonly used dependencies [spack.io, 2020]. This can also benefit workflows, which are typically hard to set up across different HPC sites due to their potentially heterogeneous software requirements.

5.4.2 Information Sources and Decision Points in Data Description Libraries

This section takes a closer look at I/O related knowledge available to data description frameworks. The following discussion therefore is focused on middleware specifically designed to handle scientific data. Most I/O libraries will provide users with the following functionality:

- A self-describing data model with native data types and primitives to describe repeating or structured data. Often it is possible to register compound data types to easily represent particles or volumes which routinely consist of multiple properties and values.
- Most frameworks allow attaching metadata, this to an extent can be exploited to improve performance. Examples include listing regions of interest, or annotating when advanced or custom compression schemes are applied to the data or to reference additional index structures.
- Some data description formats allow modules to optimally support different storage backends, for example, in addition to POSIX-style filesystems also object storage. Often backends expose tuning parameters but finding the right parameters for a given system or application often requires rigorous parameter studies. Typically, these mappings to storage system details is not concerned with the embedding of a workflow.

As different types of information are available depending on the data description framework, it follows a closer analysis of a number of formats particularly relevant from an HPC perspective.

Hierarchical Data Format 5 (HDF5): The Hierarchical Data Format (HDF5) is used by many scientists as it provides a portable and scalable interface to store and receive multi-dimensional datasets. An overview of HDF5 is illustrated in Figure 5.11. HDF5's high-level API provides a filesystem-like structure to organize complex heterogeneous data into datasets (H5D) and groups (H5G) which can be extended with custom metadata via attributes (H5A). In addition to a number of native datatypes, users can define their own datatypes (H5T).
While a dataset in HDF5 typically is bounded, HDF5 also supports virtually unbounded datasets. In either case, it is important to provide a convenient mechanism to subset data for which HDF5 introduces hyperslabs (H5S). Monitoring these hyperslabs is a valuable information source that could be used to inform prediction models for future data access.

In addition, HDF5 comes with many tunables via property lists (H5P) and a number of different filters to transparently transform data in addition to file drivers which take advantage of different underlying storage systems. As such HDF5 can perform parallel I/O via MPI-IO or take advantage of distributed file system features such as striping. Setting these tunables often can be associated with a decision point.

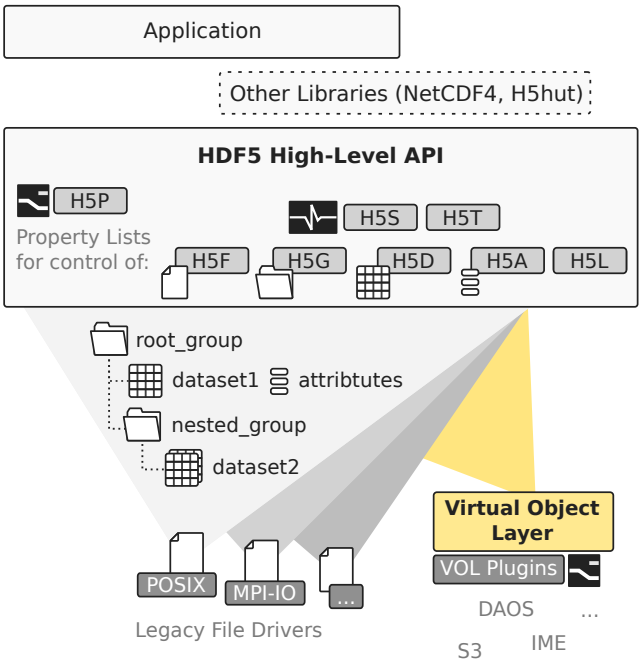


Figure 5.11: HDF5 High-Level API with tuning parameters, decision points and information sources highlighted. A recent addition to HDF5 is the Virtual Object Layer which offers fine grained-control over how to store and organize HDF5 objects such groups, datasets and attributes.

With the introduction of the Virtual Object Interface (VOL) developers and vendors can take wide-ranging control over the handling of different logical HDF5 objects such as datasets, groups, and attributes. While this means potentially re-implementing large parts of functionality currently being part of HDF5, the abstraction layer allows mapping functionality and more specialized object handling onto different distributed services.
Because HDF5 is feature-rich, code can get fairly verbose, which is why multiple libraries offer simplified interfaces to HDF5 data. Various application facing tools are built on top of HDF5 such as NetCDF4, H5hut, or F5. Similarly, HDF Group provides a number of extensions that implement some common data types. Examples include *H5Part* for variable-length 1D particle arrays, *H5Block* for Rectilinear 3D scalar and vector fields, or *H5Fed* for adaptively refined tetrahedral and triangle meshes. For all of these, the goal is to encapsulate the complexity of HDF5 and expose only the parts most relevant for a certain type of application.

Network Common Data Form (NetCDF): Another prevalent data format in HPC is Network Common Data Form (NetCDF) which originated in the atmospheric climate sciences but is used also by many other domains. NetCDF provides a strong promise to maintain backward compatibility while also offering a relatively simple interface to structured data. As of version 4 NetCDF introduced support for parallel I/O first on top HDF5 and in a follow-up release also on top of PnetCDF [Li et al., 2003].
NetCDF is commonly used in the climate community and in CFD applications. As such additional community conventions, such as the CF conventions, are sometimes influenced by the provided features and data model and vice versa. As a result, additional tooling develops around such data

formats, such as climate data operators (CDO) or NCL. [Unidata, 2020] lists over 120 such utilities for manipulation and display of NetCDF data.

Other scientific formats on top of HDF5/NetCDF: Because HDF5 provides so many features and optimizations it is sometimes considered too complicated to interface with directly. As a result, specialized but more convenient to use wrappers to HDF5 or NetCDF are used in various scientific domains such as:

- *h5hut* and *h5part* provide data models and parallel I/O aimed to support physics codes working with particle data.
- *F5* implements many complex data formats such as meshes, grids and fields on top of HDF5 aimed at visualization.
- *GIO* implements parallel I/O for structured geodesic data on top of NetCDF4 and pnetCDF.

Adaptable Input Output System (ADIOS): Applied at compile time the Adaptable Input Output System (ADIOS) [Lofstead et al., 2008] aims for automatic optimization of application I/O by allowing applications to declare I/O using a configuration file instead of implementing lower-level I/O them-selves. ADIOS uses a number of abstractions to allow mapping to other established tools for adaptivity. As of ADIOS2 core abstractions include I/O variables, attributes, engines, and operators. Users are expected to provide a configuration file that describes I/O, for which then code is generated. ADIOS supports a number of so-called engines, which allow to exchange and store data in different formats (such as BP, HDF5, InSitu MPI, DataMan over WAN, or Sustainable Staging Transport) with a varying degree for fine-tuning. The BP3 format offers the most options with options for multithreading, buffer-sizes, the collection profiling information, or whether to perform metadata operation collectively. To inquire about and inspect the BP format, users can use the `bpls` command.
ADIOS2 has language bindings for C++11, C, Fortran, Matlab and Python. All language bindings rely on the ADIOS2 C++11 Library implementation. The Fortran and Matlab bindings are built on top of the ADIOS2 C bindings. ADIOS requires applications to explicitly specify variables and datasets, thus explicitly expressing the structure of data. The choice of engine type, data format, or transport are *decision points*, which might be matched to workflow-, user-, or project behavior.

Databases and other persistency services: While less common in HPC settings, some sites provide specialized data services such as relational databases, schema-less databases, or key-value stores. To an extent, this is encouraged also by offerings found in cloud platforms. Similarly, various metadata services are using databases to establish a global state across multiple systems. Databases commonly can come with a number of tuneable options such as choosing index structures, but also database schemata that directly impact performance. Databases optimized for a large variety of different applications exist including relational databases, column or row optimized databases, schema-less and document-oriented databases, or graph databases.

5.4.3 *Instrumenting Middleware*

Instrumentation of libraries and middleware is often far more useful than instrumenting the application. Activity passing through the interface of an API is associated with well-defined semantics that have been abstracted by the middleware. A challenge with information collected at the middleware level, however, is that workflow context has usually been lost already, which in turn hinders associating activity at one place with activity at another. It is, however, not always necessary to correlate activity at runtime, but instead rely on post-mortem analysis.
Solutions to transparently instrument and collect activity in applications and libraries are often applied at link time. Examples include Tuning and Analysis Toolkit (TAU) and the Scalable Performance Measurement Infrastructure for Parallel Codes (Score-P) [Knüpfer et al., 2011; Score-P Developer Community, 2019] for instrumentation. Other approaches such as Caliper [Boehme et al., 2016] provide a runtime for common instrumentation tasks but require application and library developers to actively annotate their software. Focusing on I/O related metrics collected from common I/O interfaces, Darshan is particularly relevant because multiple large HPC sites are running Darshan by default with most of their jobs.

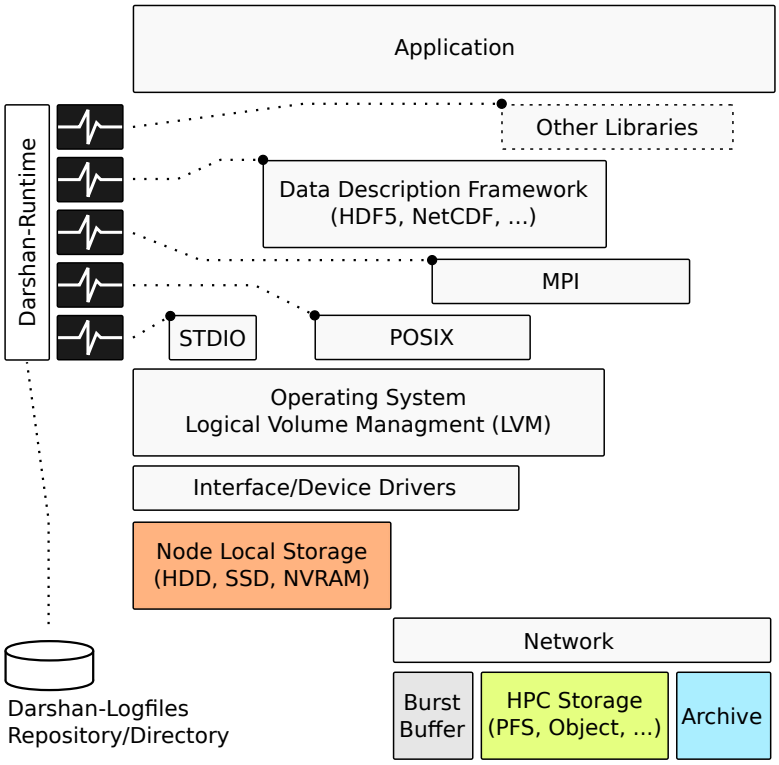


Figure 5.12: With Darshan one can collect I/O-related activity on the application and library levels without requiring special privileges. The dotted lines to STDIO, POSIX, MPI, and HDF5 depict some instrumentation supported with Darshan by default, but users can define additional wrappers for other libraries as well. Recorded log data is stored into log files before a group of MPI processes terminates.

Darshan: Unlike some other tools, Darshan captures I/O activity related to the application and I/O library for multiple layers in an HPC stack. In particular, Darshan can instrument many I/O interfaces (POSIX, STDIO) and libraries (such as MPI, HDF5) and can collect and aggregate its own I/O-related performance counters. Darshan with extended tracing support (DXT) [Xu et al., 2017] also enables the collection of full I/O traces up to a configurable size. In many cases, instrumenting an application is as easy as using `LD_PRELOAD` to interpose Darshan's instrumentation library between the application and its I/O libraries. This approach can be used by all users without requiring special privileges.

Figure 5.12 illustrates some levels at which Darshan can collect I/O activity within a system stack. Darshan comes with a module system allowing to instrument different interfaces through wrapper functions. Darshan can be used to instrument static and dynamic libraries. The static case requires applications or libraries to be compiled and linked against Darshan's wrappers. In the dynamic case, the `LD_PRELOAD` mechanism is used to intercept and resolve symbols to Darshan's wrappers. Virtually every shared library can be instrumented this way, which allows users to collect custom performance counters or even detailed traces with the DXT facilities.

Middleware Level Summary

Middleware is among the most attractive targets both for *decision points* as well as an *information source*. I/O functionality, for example, is often delegated to *data description middleware* such as HDF5 or NetCDF. Because middleware typically serve narrowly defined purposes and they feature limited APIs, thus helping to isolate where to adjust behavior as well as providing more obvious targets for instrumentation. In fact, often one middleware will depend on other libraries or middleware itself. Often middleware also provides interfaces for introspection and troubleshooting which can be valuable *information sources*
While it may not be feasible to optimize abstractions found by individual applications, abstractions used in middleware tend to be very stable and they potentially benefit many different users. As a result, when it comes to instrumenting middleware, I/O profiling information can be captured using third-party tools such as Darshan or Score-P.

5.5 Node Level

HPC applications, as well as distributed services, are typically broken up into nodes with only a few responsibilities. If not virtualized, a node typically directly maps to physical hardware controlled by one instance of an operating system. While service nodes and compute nodes often differ in terms of hardware configurations, both are typically Linux-based and consequently share many properties on the OS and hardware level. From a workflow perspective the following differences are particularly important:

- Compute nodes have first-hand access to application-related information, but will only take a client perspective to most services.

- Service nodes often have an overview of the respective service but lack direct access to application details.

Ultimately, all information sources and telemetry providers have to cope with node level abstractions and features offered only through operating system APIs. Unfortunately, a comprehensive up-to-date collection of node level I/O telemetry providers for HPC environments is not available. None the less, a good general overview of tools and also considerations in support of building an intuition for system performance are collected in [Gregg, 2014]. The collection, however, takes a cloud perspective and focuses on tools to inspect individual subsystems instead of taking a holistic perspective. Still many of the utilities and dynamics presented are helpful

with regard to services found in both HPC and cloud environments. Unfortunately, tools common in scientific contexts and client APIs for HPC storage systems such as Lustre or services like Slurm are not covered. The next best alternative to manual documentation, are the glue code and module collections found in the source repositories of tools like *collectd* or *Prometheus* which introduce abstract collection interfaces for capturing time-series and log data. These tools are covered in more detail in Section 5.8 (Holistic Approaches), as the focus in this section is on decision components and information sources which can be made locally to a node. Here it is notable, that as long as no mechanism is provided to associate activity with a workflow no resulting action can be considered workflow sensitive. In most cases, it will not be attractive to inquire about workflow context on a request to request basis, but instead on a coarser granularity. An exception to this may be requests carrying hints on the workflow context itself. Most storage and I/O related decisions at the node level are related to the utilization of memory, storage and network devices.

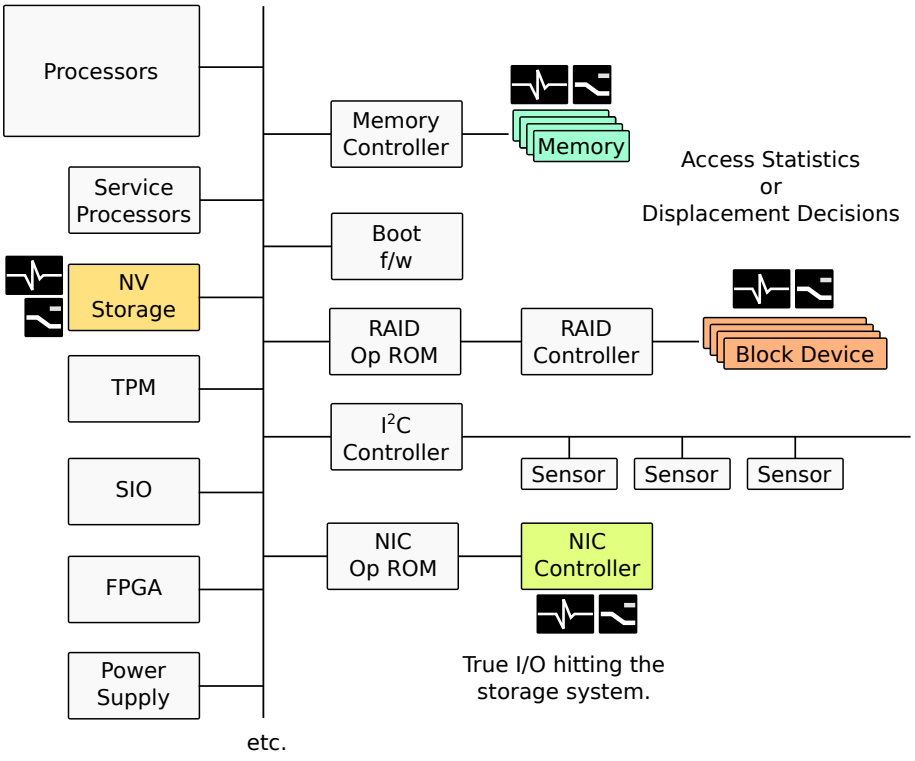


Figure 5.13: Typical complex and firmware-driven hardware components found in compute nodes with memory and storage components highlighted. Adapted from a NIST firmware security perspective [Regenscheid, 2018]. All these components being governed by firmware suggest potential for sophisticated adaptable decision-making also in these controllers.

Node Local Memory and Storage: Figure 5.13 illustrates node local hardware devices and controllers. The particular perspective originally considers firmware security, but is also useful from a storage and I/O perspective, for example, in anticipation of active storage. Memory, storage, and network devices are highlighted in color. For system main memory, as well as secondary memory which might be attached to service processors (green), mostly activity monitoring is assumed, as staging decisions are usually enacted through different APIs, although workflow-aware memory reorganization can be imagined. Block devices such as RAID arrays, hard disks or SSDs (dark orange), are most relevant from a storage perspective. Here workflow aware striping could yield performance improvements. When considering active storage many more workflow opportunities would open up. Similar considerations apply to a new generation of non-volatile or stor-

age class memory (light orange). Non-volatile storage, however, is more likely to be considered in hyper-converged deployments, but how to best integrate non-volatile memory is an active area of research. Finally, network interface cards (yellow), as required to use Ethernet or InfiniBand are connecting nodes to other compute or services nodes.
More theoretically, a workflow optimization could also be applied at boot time, for example, to fetch a specialized image or mount specialized storage services. In practice, a different orchestration mechanism seems more appropriate to apply such interventions.

5.5.1 OS Services and Interfaces for Storage

Traditionally, most storage devices would be exposed as block devices which, as their name suggests, allow to manage and access data via addressable blocks. A variety of abstractions on top of these technologies are used by different operating systems. In HPC systems with Linux this is achieved through a combination of kernel modules that implement drivers that expose devices through the `/dev/*` interfaces. The underlying protocols used to interface with the devices have changed over time, but most relevant are (i)SCSI and increasingly NVMe. On a higher level, many devices and file systems are accessing storage devices using the *Logical Volume Management* (LVM) provided by Linux. While a few exceptional applications and services are directly interfacing with block-level devices for maximum performance, most applications will rely on POSIX-style interfaces for low-level data management at some point. As a POSIX-style interface provides file and directories, a volume (physical or logical) is typically formatted using a specific filesystem implementation. Common choices in HPC environments are ext4 or ZFS. The choice of filesystem can have a profound impact on performance as they employ different underlying data structures which allows to trade off performance characteristics depending on the use case or workflow. While this is an important decision point, it is typically not possible to dynamically switch due to unbearable overhead or because the setting is applied in a privileged environment.
As many of these interfaces aim to keep functionality to a minimum, developers and operators are left with a wide range of different tools and technologies to be familiar with. To an extent this drives efforts to develop tool collections and frameworks of building blocks for storage management. Another common objective is also to move functionality into user space to avoid context switches as much as possible.

File System in Userspace (FUSE): FUSE is a kernel module for Unix systems which allows users to provide file system registered with the Virtual File System (VFS) but with the implementation running in user space as is illustrated in Figure 5.14. It is part of the Linux kernel as of version 2.6.14. Both fine granular logging as well as various decision points can be part of a custom FUSE file system. While this moves functionality to user space, FUSE actually imposes additional context switches.

Storage Performance Development Kit (SPDK): Another effort to move functionality to user space is SPDK which breaks down into abstractions for storage protocols, storage services and integration and command-line utilities. While developed by Intel the toolchain aims to be vendor-neutral. Integrated technologies include low-level protocols like NVMe, SCSI, OS

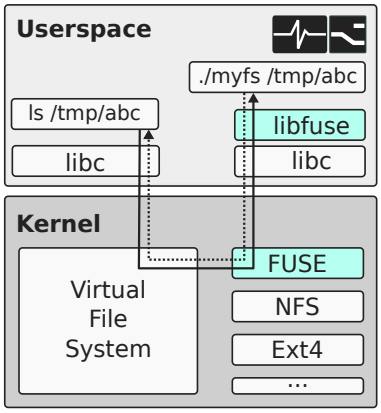


Figure 5.14: A file system in userspace with potential to record activity as well as influence decision. All activity is invisible to the application which is using the file system. Adapted figure from [Gupta, 2019].

abstractions like logical volumes and virtualization through QEMU as well as high-level services like Ceph, Cinder or even database systesm. SPDK includes drivers for NVMe devices, *virtio* as well as accelerators such as Intel's QuickAssist Technology to support compression and encryption of data. Similar to SPDK but with a focus on persistent memory technologies such as NVRAM is the *Persistent Memory Development Kit* (PMDK) which provides common functionality to interact with storage-class memory.

Monitoring system utilization: Many system information and settings are exposed through so-called pseudo file systems on Unix/Linux systems. In particular, these are `/sys/*` and `/proc/*`. Obtainable information includes state and statistics of the network, devices, the CPU, but also for individual processes. Many tools will query this information, but will allow more convenient manipulating and more accessible presentation.

Pressure Stall Information (PSI): As of Linux 4.20, a new load averaging method, called *Pressure Stall Information* (PSI), was added which aggregates reports for CPU, memory, and I/O information [Weiner, 2018]. PSI can be accessed via `/proc/pressure` and overcomes a number of limitations and pitfalls with the already existing load average information. For example, PSI reports the percentage of wallclock time in which tasks are waiting, so accounting for the number of CPUs and relating it to the number of running tasks as reported by the old metric becomes unnecessary. This notion of counting delay is used for all three PSI variants. PSI uses a smaller minimal sampling window of 10 seconds, which can often be more adequate for bursty loads. The I/O information is further distinguishing the notion of *some*, as in time of which some tasks were delayed, and *full* as in the time of which all tasks were delayed.

5.5.2 *eBPF: Extended Berkeley Package Filter*

A relatively new but powerful and efficient source for detailed custom profiling and tracing information on Linux-based systems are extended Berkeley Package Filters (ePBF). Originally intended as a filtering mechanism for network packets matching specific rules, hence the misleading name, it has evolved into a much more flexible mechanism used for profiling and tracing or for firewalls among other things [McCanne and Jacobson, 1993; Molnar, 2015]. While currently, not often found in HPC environments, various production engineers and operators of large cloud operations report using eBPF for being fast and safe [Fleming, 2017; Gregg, 2015]. This is realized by offering a register-based virtual machine (VM) using custom 64-bit RISC instructions which can be injected using just-in-time compilation and which are then run inside the Linux kernel. A eBPF program can hook into various triggers including kprobes/uprobes, perf events, trace points, dtrace probes, sockets and more. eBPF is intentionally not Turing complete and prohibits unbounded loops ensuring termination is guaranteed.
Programs are verified before loading using control flow graph analysis to check for unreachable instructions and loops. The virtual machine ensures valid register and stack states and prevents out-of-bounds jumps and data access. For users without special privileges (`CAP_SYS_ADMIN`), a more secure mode is available which further prohibits the use of pointer arithmetic preventing leaking kernel addresses. An example eBPF program is illustrated in Figure 5.15. Figure 5.16 illustrates how eBPF is working on an abstract

```
1  #include <bpf/api.h>
2
3  SEC("to_netdev")
4  int handle(struct sk_buff *skb
       ) {
5          // ...
6          if (tcp->dport == 80)
7          redirect(1xc0);
8
9          return DROP_PACKET;
10 }
```

Figure 5.15: Example for an eBPF program to redirect/drop packages based on the port.

level, and how telemetry information can be received using the `perf_output` callback or the `async_read` call. eBPF offers various data structures that can be used to aggregate data such as `perf_events`, variations of hash tables and maps. Maps, in particular, can be accessed also using user-space programs.

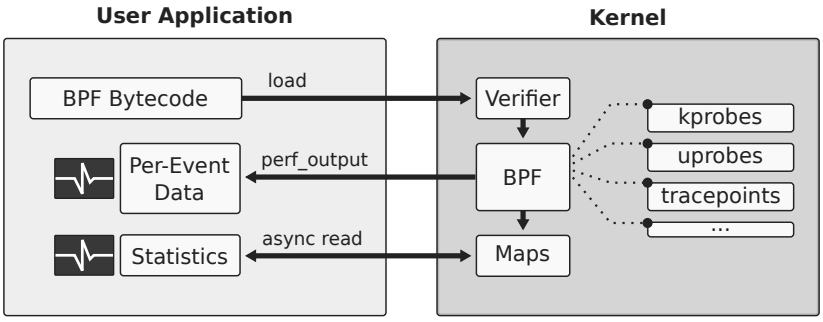


Figure 5.16: eBPF offers synchronous and asynchronous access to collected telemetry. Before actually loading an eBPF program, the verifier checks and potentially rejects loading programs in violation. Illustration adapted from [Gregg, 2019].

From a decision point perspective, eBPF is occasionally used to perform filtering and realize custom rule-based load balancers. Some vendors also in context with the I/O Visor project have signaled to support BPF programs on the hardware level in the future. An example for this is eXpress Data Path which covered in more detail in Section 5.6 (Network Level). The Kubernetes container orchestrator also promotes the use of eBPF programs to efficiently obtain statistics or to increase and customize log levels as may be required for audits [Kubernetes, 2017].

Node Level Summary

Workflow context is rarely used at the node level, workflow artifacts such as performance implications and system load can be collected at this level. But especially service and storage nodes employ caches or provide limited compute capabilities that would allow to perform I/O optimizations. While monitoring data at this level often can be accessed relatively freely, changing behavior typically is restricted to privileged users, usually excluding any type of application or workflow.
As compute node allocations are often exclusive this would not be strictly necessary except for selected services. As a result, a lot of functionality traditionally found in kernel space is being moved to userspace to avoid costly context switches but also to offer applications and middleware more control about how they utilize the hardware.
But also operating systems are trying to become more flexible for example by allowing custom routines for example for analysis or package handling directly for example through eBPF.

5.6 Network Level

On the network level, workflow context is seldomly exposed explicitly. Although a substantial amount of activity, relevant to gauge the workflow impact on the overall system can be collected at the network level. It is challenging, however, to associate activity collected at this level with a particular workflow. For the same reason, it is also hard to perform actions on data, such as in transit transformations. Besides considering the software and hardware perspectives there are two modes in particular to consider for workflow telemetry and actions on the network level:

- *Coordinated use*: Participants in a distributed application/workflow, can to a degree account for their impact on the network, and thus adapt their behavior, or more often have it adapted by the developers.
- Reactive *software-defined networks* (SDN): To a degree next-generation networks are likely more flexible in their routing patterns, as such limited physical resources in many cases might be reallocated using software-defined network equipment. This way instead of changing the workflow to fit the network, the network is changed to fit the workflow.

In practice, it might be more plausible to see hybrid solutions as certain optimizations are easier to achieve by changing applications and others will be easier by adapting the network. The challenge, however, is, to propagate and expose the workflow context without introducing unbearable overheads. There is potential to change this, if workflow information were somehow integrated into package frames. As such using a form of deep package inspection it could become possible to propagate and react to workflow context. In many cases, there will exist better mechanisms to propagate this information though, but for asynchronous pipelines the small overhead could be worth it, as no further coordination will be required.

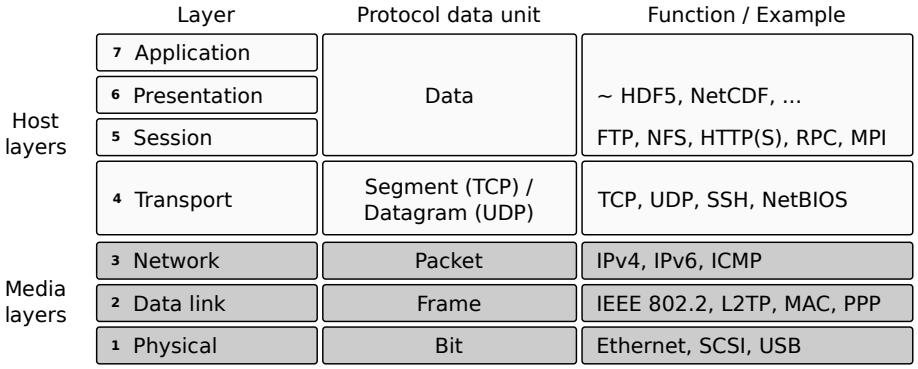


Figure 5.17: The ISO/OSI Network Model, where each layer also offers opportunities to collect data useful for a decision component outside of the network stack. Note that InfiniBand is not included here, as it recombines functionality from multiple layers such as the physical, link, network and transport layers.

ISO/OSI Layer Model It is useful to consider the ISO/OSI Layer Abstraction for information passing through the network stack in this context, as is illustrated in Figure 5.17. For the two previously outlined opportunities the ISO/OSI model highlights how *isolation* helps to make complexity manageable by not leaking context but at the same time promotes passing on many optimization opportunities. Component responsibilities for the different layers, however, are beginning to blur as a lot of protocol functionality is being offloaded either onto the network interface cards or even into switches.

5.6.1 Software-defined and Adaptive Networking

Traditionally, the network in an HPC environment would need to be considered a fairly static resource shared by various nodes and services. This notion is less true in the future for mostly three conspiring dynamics:

- Proliferation of SDN switches and network interface cards (NICs) offering acceleration of, for example, MPI via offloading.
- More flexible network topologies allowing for complex routing patterns, such as the dragonfly topologies.

- Development of mature and more flexible software stacks and frameworks to take advantage of the former two.

Offloading in general: As more applications adopt asynchronous I/O, responsibilities of node-to-node communication can be offloaded, as supported by some Infiniband hardware and others, thus freeing precious CPU cycles the operating system would have otherwise spent for network processing.
Offloading requires capable hardware on data paths and more complex logic to perform computations, and as such is most successfully applied when applications and network hardware agree on common concepts. While this is an active area of research, it is not widely adopted in production systems beyond narrow use cases. Examples include support for encryption and compression as found in Intel's QuickAssist technology [Intel, 2017a] or MPI offloading as performed by Mellanox [Mellanox, 2018]. If more such technologies are to be found in modern HPC stacks, they become important points of actions also from a workflow perspective.

MPI Offloading: MPI's send and receive operations are typically identified by considering the tag, communicator and source information and redirecting them to the appropriate target buffer. Where the list of buffers is referred to as the *matching list*, and the process of finding the corresponding buffer, which is called *tag matching*, can be offloaded to NICs with special features such as [Mellanox, 2018]. This effectively can turn certain operations into one-sided data transfers, which are often favorable for scalable operations. Especially, applications performing a lot of collective operations benefit from MPI enabled NICs reducing CPU overhead and taking advantage of overlaps and reducing jitter [Shainer, 2011].
Considering these particular offloading capabilities offers only limited useful opportunities from a workflow perspective right now. An exception are in situ workflows which, while typically not possible with existing MPI implementations, would benefit from growing an MPI Communicator to include additional hosts that are in charge of analysis workfloads.
There are two important factors to consider, both the network as well as user of the network such as applications, libraries, and services can adopt adaptive behavior.
With multiple adaptive agents oscillating feedback loops and complex emerging behavior can occur. But even non-circular behavior which emerges from decisions made at the application/library level are often not easily compensated at the network level. In addition, such a workaround at the network level may be undesirable if they post-pone resolving more fundamental design problems. On the other hand, network utilization, latency, and throughput are valuable information sources both for the application/library layers as well as for services.
Usually, applications will not attempt to utilize low-level offloading capabilities directly but instead use middleware to allow falling back to software implementations if now hardware support ist available.

Unified Communication X (UCX) is an open source communication framework targeting data-centric and high-performance applications [Shamis et al., 2015]. It targets different communication paradigms including message passing (MPI), partitioned global address spaces (PGAS) as well as remote-procedure calls (RPC).

UCX aims to make developers more productive by offering high-level APIs on top of abstract primitives to allow the utilization of different available hardware resources and offloading opportunities. To do so UCX exposes three kinds of APIs: a services API (UCS), a low-level transport API (UCT), and high-level protocol API (UCP).

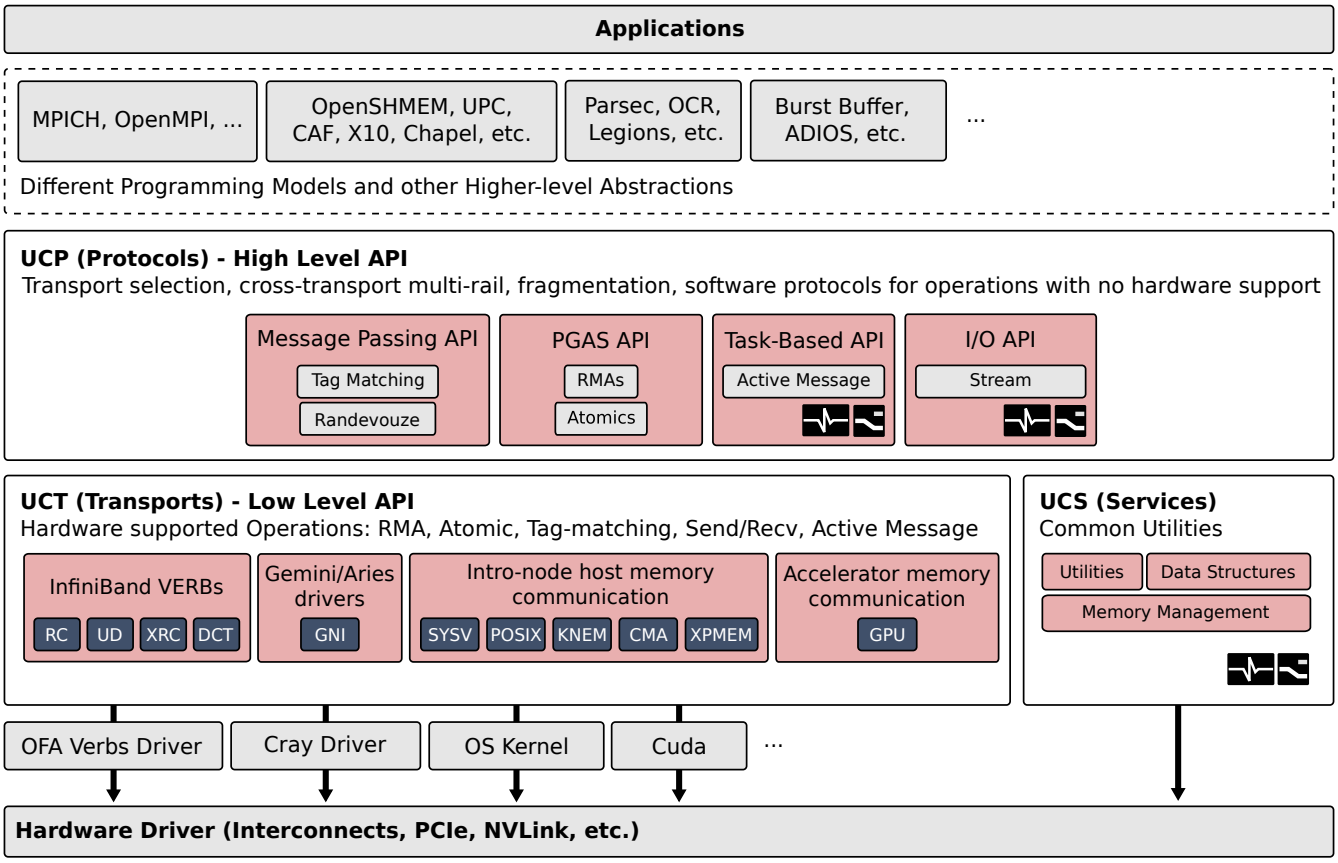


Figure 5.18: Overview of the architecture of Unified Communication X (UCX) with the most important information sources and decision points for workflow I/O optimizations highlighted. At a high-level the task-based API and the I/O API are most to optimize storage I/O. The structure of data may be in part be influenced using the services API which defines data structures. Illustration adapted from official documentation [OpenUCX, 2021].

Figure 5.18 illustrates the general architecture of UCX and highlights three areas which are particularly attractive as *information sources* as well as *decision points*. Very attractive from a workflow perspective is the abstraction for task-based APIs as they directly cater toward the programming paradigms employed by many workflow engines. Similarly, from a storage I/O perspective the high-level APIs offers I/O abstractions for streams. Using the UCS services API it is further possible to introspect data structures or influence memory management decisions.
Various MPI implementations, such as OpenMPI and MPICH, as well as some workflow engines, such as Dask and Spark, can be configured to rely on UCX functionality for communication [OpenUCX, 2019].

LibFabric is an open source library to expose network services while maintaining scalability despite hiding implementation details developed by the OpenFabrics Interface Working Group [OFIWG, 2020]. It aims to cater to application and middleware such as MPI, SHMEM or PGAS by offering a single interface abstraction for which various network vendors offer OFI providers. Various core providers are available including support for Sockets, Verbs, Cray's GNI or Mellanox MXM. The composability of LibFabric extends also into cloud environments where, for example, Amazon Web Services (AWS) based their Elastic Fabric Adapter on LibFabric in support of MPI applications.

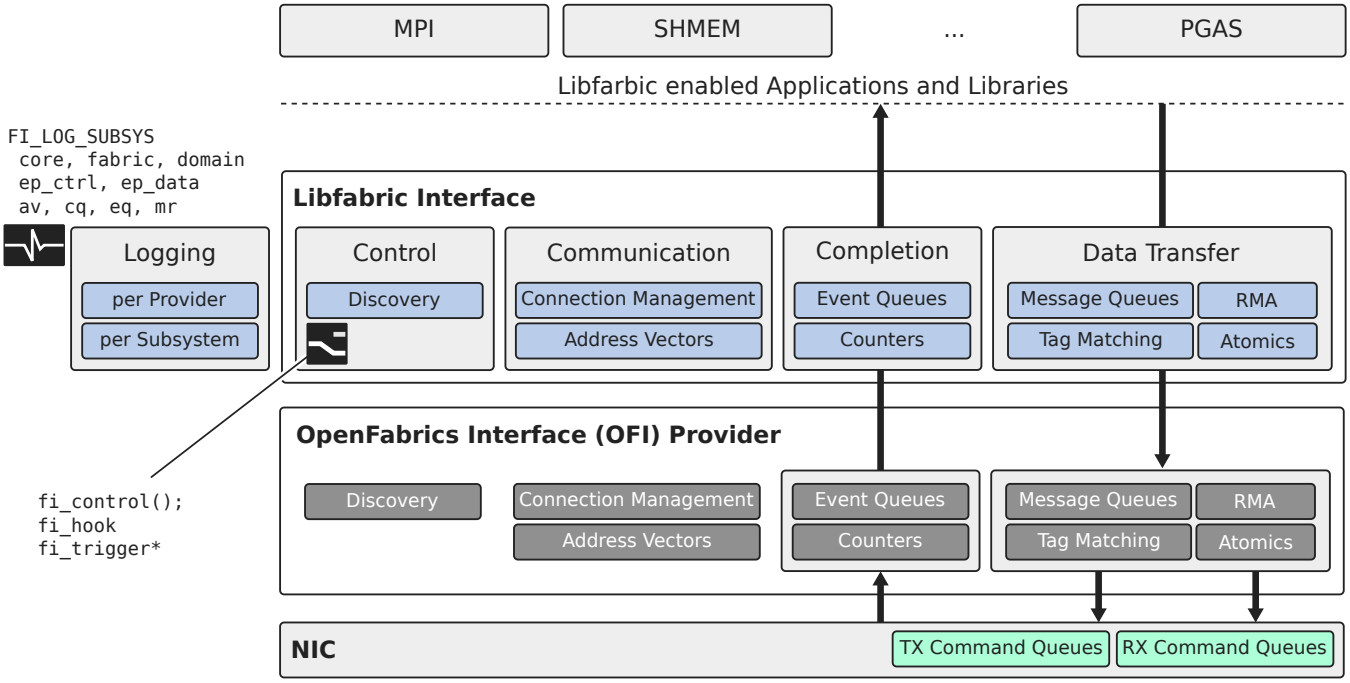


Figure 5.19: Network abstraction especially suitable for HPC environments catering to middleware such as MPI, SHMEM and PGAS. While only a few mechanisms for control and intervention exist, various means to intercept provider and device traffic can be tapped as an information source. Illustration adapted from official documentation [OFIWG, 2020].

The architecture of LibFabric consists of four core service interfaces (compare Figure 5.19): The *transfer services* offering primitives for one- and two-sided communication, RMA, atomics and tag matching. *Completion services* and *communication services* help with ensuring scalability, for example by minimizing cache footprints, by offering light-weight completion services useful for offloading or by hiding address resolution details when addressing groups. Finally, there is *control services* which can be used by the application to inquire and request certain features from the network through LibFabric. Which makes it easier to map, for example, an MPI implementation which requires support for tag matching and RMA to a variety of existing OFI providers while at the same time reducing boilerplate, for example, for error handling. The *control interface*, therefore, can be considered a *decision point*. LibFabric also exposes various network-related limits, which can be exploited by resource managers which might use it for scheduling decisions or to prevent over-subscription. Another interesting feature is offered through LibFabric `mode bits`, which are intended to leak context by providing scratch space to be populated by applications. This is, for example, used by OFI providers to track requests or to optimize for the application.

Data Plane Developer Kit (DPDK) is a kernel-bypass solution that offers a range of open source libraries and drivers for fast packet processing [DPDK Project, 2020]. The goal is to invest a minimum of CPU cycles, so usually less than 80 cycles, to process network packages. It is also used to accelerate virtual switches, as done by Open vSwitch (OvS) over DPDK. DPDK originated from an effort by Intel, but most major vendors are participating in the initiative. As such it could be an attractive target to inject in transit transformation, but also to collect telemetry information.

eXpress Data Path (XDP) is an in-kernel high-performance programmable network data path, which in contrast to DPDK is not a kernel by-pass. It aims to offer dynamic control over network packages but without requiring kernel modifications. It does not replace or compete with the existing TCP/IP stack present in Linux but allows to fast-pass specific packages,

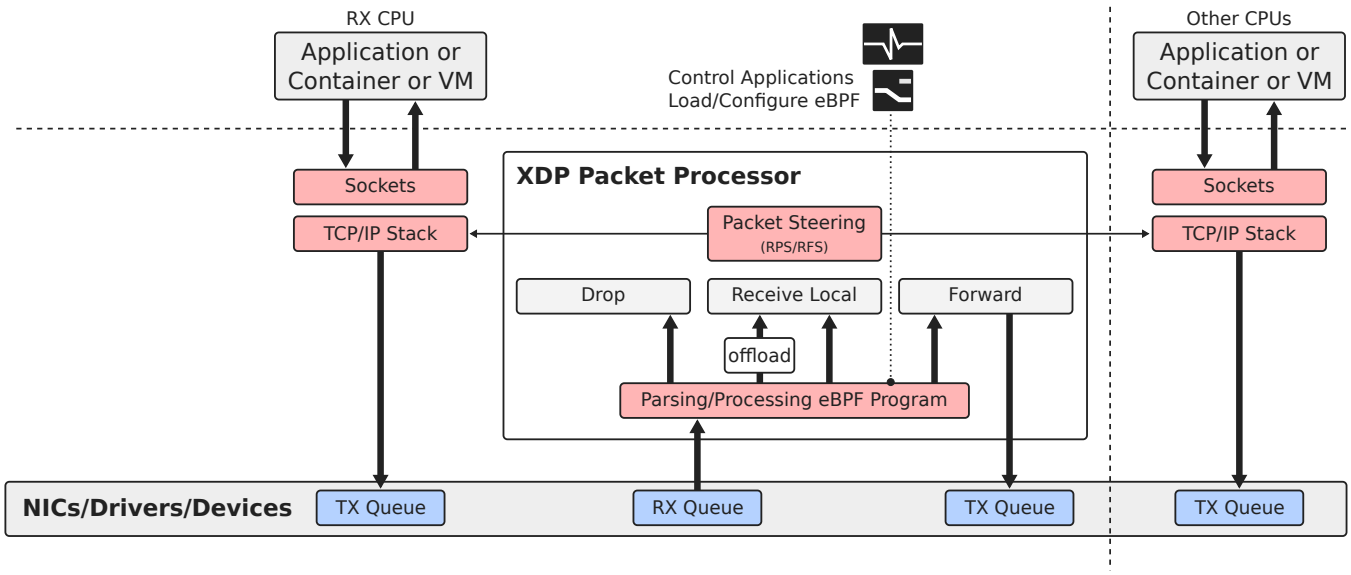


Figure 5.20: XPD is another mechanism which builds on top of eBPF programs to allow analyses and setting up a fast-pass for data matching certain rules. Similarly, it can be used to collect telemetry. Adapted from [IO Visor, 2020].

thus allows speedups by forwarding to offloading devices or dropping packages early.
The basic mechanism with opportunities for information sources and decision points highlighted is illustrated in Figure 5.20. As shown, XDP also relies on eBPF programs (Section 5.5.2) to reach a verdict how the package is to be handled, namely `XDP_DROP`, `XDP_TX`, `XDP_PASS` and `XDP_REDIRECT`. This way XDP is relatively platform-independent and does not require updating the kernel, even if complex rule updates are applied. [Karlsson and Töpel, 2018] points out that XPD achieves good performance when data has to be touched in comparison to kernel-bypass solutions such as DPDK.

Network Level Summary

As with the node level, workflow context is rarely used at the network level. Generally, there are two perspectives to take on a network: A workflows can be careful to take advantage of the system it finds and assume that the network is a passive resource that is unwilling to adapt. In this case, workflow I/O optimizations require the identification of contention points. *Information sources* for this are performance counters and telemetry information exposed by the network equipment. In a second much harder to automate step, a workflow can attempt to reconfigure itself to avoid contention, most of these tunables cannot be performed at the network level as long as no software-defined network is used.
More recently software-defined networks allow the network to be reconfigured, or to adjust to observed workloads itself, or to offload processing which the operating system otherwise would have performed at the expense of an application. Generally, this encourages applications to consider asynchronous processing schemas and data handling to achieve the best performance. From a workflow perspective, this would allow performing certain tasks transformations also in transit.

5.7 Distributed Services

As supercomputers evolve into clusters with potentially millions of compute nodes, many services complementing a supercomputer have to be distributed in nature as well. Examples for services that are especially relevant

in this context are storage systems, resource management, and scheduling systems as well as monitoring solutions.

5.7.1 Resource Management & Scheduling

Particularly important services conducting high-impact decisions in HPC data centers are scheduling and resource management systems:

- *Resource Management*: The resource management has to keep track of available and allocated resources. Resources typically are categorized and come with a number of abstracted attributes, for instance if a resource can only be used exclusively or if it is a shared resource.
- *Scheduling*: The scheduler provides a job queue and cooperates with the resource management and accounting to schedule and execute submitted jobs. Fundamentally, this is an optimization problem to schedule work under a number of different criteria such as, for example, timely job start, fair sharing while maximizing utilization. Many shared resources such as storage and network I/O are typically not directly considered in HPC scheduling systems, but they are an important decision point.
- *Accounting*: Access control, quality of service, and usage statistics are often tightly integrated with resource management and scheduling. In particular, usage statistics and job summaries can be valuable information sources to inform workflow-aware decision-making.

Unfortunately, detailed I/O statistics are often not available at a site. Still there exist approaches to combine, for example, Lustre statistics and job statistics to realize per job I/O accounting [Hammond, 2019]. As measured service models are increasingly popular, especially due to cloud environments, this lack of statistics can be expected to change in the future.
There exist a variety of commercial and open source batch scheduling solutions targeting HPC in particular, such as Slurm, PBS or Moab/Torque. A good survey, studying features of different workload scheduling systems is conducted in [Andreadis et al., 2018]. Although the survey is focusing on higher-level schedulers, many of the resource management and scheduling strategies apply. In particular, the survey identifies two types of decisions regularly conducted, namely, task management and scheduling priority.

Slurm is a batch scheduler commonly deployed in HPC systems [SchedMD, 2019a]. Slurm features plugins to extend functionality, although many sites will hesitate to use radically different strategies without rigorous testing. Figure 5.21 illustrates an overview of potential decision points where additional actions might be triggered to help from a workflow perspective. Users can submit jobs and then have other jobs depend or wait for them to be completed, this can be considered an explicit declaration of workflow dependency. There is a number of ways to influence the SLURM allocation through arguments or via environment variables:

- `--nodelist` to explicitly request specific nodes.
- `--hint` to indicate compute or memory bound workloads. A storage or I/O bound hint does not exist.
- `--qos` to request certain quality of service which are controlled by the site. Defaults can be set on a user/cluster/account basis.

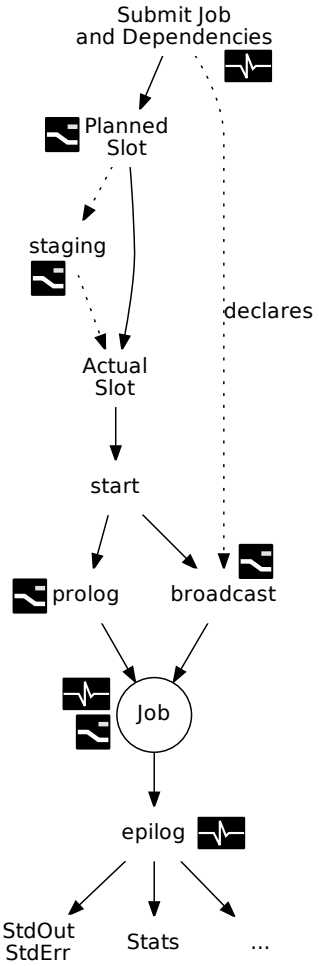


Figure 5.21: Life cylce of a job with opportunities to influence decision or to collect telemetry highlighted. Many of these could be application, user, workflow or project specific.

- `--spread-job` attempts to evenly distribute the allocated nodes.
- `--tmp` can be used to require at least a specified amount of node local temporary disk space.

Slurm will, depending on the scheduling module which is used, issue a planned allocation and optionally notify burst buffers or archives to stage relevant data, for example, when `bb/bbf` arguments are populated. Currently, the only provided burst buffer plugin is for Cray's DataWarp [SchedMD, 2019b].
On job start, Slurm will run a prolog script when provided with the `prolog/task-prolog` arguments. As such they are attractive decision points and can also be used to enable additional telemetry. To reduce stress on the file system by broadcasting the file actively before job start the `bcast` functionality is provided. Optionally, one can enable compression before broadcasting the files.
During job execution, all information and decision points available to applications are applicable in theory, although injecting them is complex when not using a shared interface with applications and middleware. SLURM for the most part can be influenced through `scontrol`, arguments to `srun` or environment variables. It is possible to enable different profiling backends which can also capture relevant I/O information when enabling the task, file-systems or network options. The information is then collected by the `acct_gather_profile` plugin.
Finally, it is possible to execute an epilog script via the `epilog/task-epilog` option, which allows to perform custom actions at the end of a job. After the job completes, typically there will be log files for standard out and error, as well as coarse-grained job statistics collected by Slurm. For a while Slurm used to have limited support for checkpoints allowing to specifying a checkpoint interval and declaring a checkpoint directory, but newer versions have since deprecated these APIs. The recommendation is to use third-party solutions such as FTI, TCL, VeloC and SCR [Bautista-Gomez et al., 2011; LLNL, 2015].

5.7.2 Storage-Level Vendor APIs and Monitoring

For storage systems I/O monitoring data is often not accessible by all users, and for proprietary systems internal performance counters may not be exposed even to site operators. Many modern systems, however, do offer APIs to query different system statistics or even provide event hooks for more sophisticated actions. In this section, the discussion focuses on information sources and decision points found in some common high-performance storage solutions. Commonalities found across different storage solutions include: Data distribution across different targets, data reduction using compression or deduplication, but also data replication to improve performance and data safety. Interestingly, the combination of multiple of these different factors is responsible for the emerging quality of service (QoS) as it can be exposed to users. Unfortunately, because these systems are typically shared resources, users will experience I/O variability due to contention occurring on multiple levels. As a result, information and decision points especially important from a workflow perspective are the following:

Data Distribution: Storage targets and data striping require storage systems to provide placement strategies. Often these strategies are deterministic for a particular deployment, leading to a situation where scaling a system

later on requires rearranging how data is distributed, which can be associated with system downtimes. Data placement, stripe size, and stripe counts mark important decision points in many distributed storage systems. Spreading out data typically allows accessing more data in parallel, thus improving read/write performance if sufficient amounts of data are accessed.

Quality of Service (QoS): Data and metadata or QoS discrimination is pursued by multiple storage systems. In some systems this is realized through separate data and metadata pools. For converged deployments, data and metadata are not necessarily physically separated but often will be. Instead of data and metadata, some systems distinguish small and large I/O. From a workflow decision perspective, this is an extension of the data placement challenge, where the selection process takes additional factors for latency and bandwidth into account.

Data Staging: Some storage systems offer concepts such as data staging or heat, where hot data was accessed recently and cold data is accessed infrequently. Formulas to determine heat can be based on a wide range of different assumptions, so that for different workflows different strategies may be appropriate. In addition, workflow information can be used to increase the heat value for data to trigger the automatic staging of data.

Management and Monitoring: Most storage systems offer utilities to manage and inquire about system health and data placement. Most often these APIs offer a number of tunables, potentially also allowing users to set parameters, while advanced functionality requires privileged access. From a workflow perspective, this allows to consider tunables as potential decision points. Similarly, users often get summarized statistics such as fill levels of different targets, while more detailed information about consensus, load balancing decisions or topological considerations are only accessible to site operators or vendors.

5.7.3 HPC Storage Systems

Commonly deployed systems in HPC environments include Lustre, GPFS/IBM Spectrum Scale, CephFS, BeeGFS for file systems and OpenStack Swift as well as Ceph Rados for object storage. In this section information sources and decision points found in some of these systems are discussed. Most of these systems realize the general architecture for HPC storage systems introduced in Section 2.1.4 (Distributed High-Performance Storage Systems) thus pooling resources instead of aiming for (hyper)converged deployments.

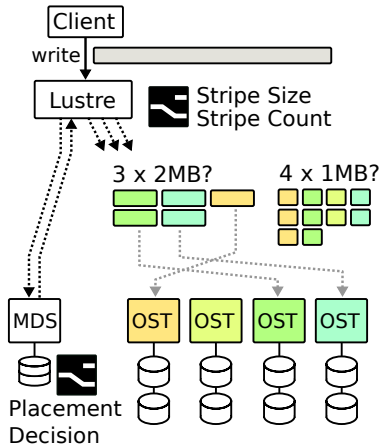


Figure 5.22: Decision points in Lustre which influence data placement and striping. Stripe size and stripe count are also important tunables for other storage solutions.

Lustre: Lustre is an open-source parallel file system deployed at many HPC sites, as it was specifically designed to accommodate the different workload characteristics resulting from metadata and bulk data operations. The architecture consists of four core components that can be scaled, namely *metadata servers* and *data servers* as well as *data targets* and *metadata targets*.
Clients will interact with the server components, which can also perform caching while the target components hold persistent data. Metadata servers and targets will typically take advantage of flash storage to achieve low latency and high operation counts. The server/target approach allows to

provide transparent failover. Bulk data is stored on disks and is typically stripped across multiple servers/targets to allow parallel access as illustrated in Figure 5.22. Lustre offers users and operators some control over data placement and parallel access dynamics via `lfs setstripe`. The `lfs` command can also be used to obtain system state, without requiring special privileges as is illustrated in Listing 5.1. Similar to the `fadvise` mechanism for POSIX file systems, there is `lfs ladvise` as a way to provide hints about future access behavior [Lustre, 2017]. An alternative interface to Lustre telemetry is `/proc/fs/lustre/llite/<system-id>/stats`. That such information can be used with Lustre systems to adapt to load imbalances was demonstrated by [Wadhwa et al., 2019].

Listing 5.1: High-level system information such as fill level and capacity as well as the number of OSTs and MDTs which are exposed by Lustre to all users.

```
1  UUID                       bytes        Used   Available Use% Mounted on
2  lustre02-MDT0000_UUID       2.9T      107.3G        2.8T   4% /mnt/lustre02[MDT:0]
3  lustre02-MDT0001_UUID       3.4T       42.1G        3.3T   2% /mnt/lustre02[MDT:1]
4  ...                          ...         ...         ...   ... ...
5  lustre02-MDT0006_UUID       3.4T       49.6G        3.3T   2% /mnt/lustre02[MDT:6]
6  lustre02-OST0000_UUID     224.0T      163.9T       57.9T  74% /mnt/lustre02[OST:0]
7  lustre02-OST0001_UUID     224.0T      164.7T       57.1T  75% /mnt/lustre02[OST:1]
8  lustre02-OST0002_UUID     224.0T      161.2T       60.6T  73% /mnt/lustre02[OST:2]
9  ...                          ...         ...         ...   ... ...
10 lustre02-OST0093_UUID     224.0T      159.3T       62.5T  72% /mnt/lustre02[OST:147]
11
12 filesystem_summary:        32.4P       23.6P        8.4P  74% /mnt/lustre02
```

As of LU-11234 there are also Data Placement Policies (DPP) for OST pools. Policies are enforced at file creation time by the MDS, which chooses which OST pool is used. Rules are defined by administrators and may react to file extensions, user ID, group ID, network ID or a combination of those. As such these policies are useful from a workflow perspective for routing files based on the access pattern in the workflow of a user or project.

GPFS/Spectrum Scale: Another widely deployed storage solution is IBM Spectrum Scale, which was formerly marketed as General Parallel File System (GPFS) until 2015. It was originally introduced in 1998 and supports Linux since 2001. As a clustered filesystem, it offers fail-over semantics and allows striping to increase read and write performance. GPFS volumes allow for asynchronous replication which can be used to selectively improve for example read performance or data safety. To simplify application correctness for parallel access there is a distributed lock manager. Metadata and data can be directed to different devices to improve performance. Since copy-on-write is used, it is also possible to efficiently create snapshots. Because IBM offers various complementary storage solutions, it comes with many features for HSM integration. By allowing to use declustered RAID, rebuild times can be sped up while also lowering read and write loads on particular devices.
Spectrum Scale comes with a wide range of telemetry and allows to configure multiple collectors of which most are accessible through `mmpmon`. In the architecture so-called *sensors* collect performance data while *proxies* collect metrics for a particular protocol. GPFS features a relatively extensive event system covering mechanisms such as quorum finding, as well as block events in addition to audit logging. These are useful as information sources to react to, but often also mark decision points. A workflow could, for example, leverage the `mmaddcallback` command-line utility that allows to register callback commands that are to be executed when a certain event occurs. Most of the events are not immediately useful from a workflow perspective as they mostly respond to the storage system state.

Ceph: While primarily an object store at the architecture level, Ceph provides a variety of storage service models while also addressing some of the metadata challenges associated with more traditional file systems. The *Reliable Autonomic Distributed Object Store* (RADOS) is the core component on top of which other services are built and with which applications and other middleware can interface through `librados`.
Object storage can be provisioned in compatibility modes to expose Swift or S3 APIs. CephFS, as a POSIX-compliant file system interface that can be either integrated using a Linux kernel module or using FUSE. CephFS is interfacing directly with the RADOS architecture and thus exploits many of Cephs's low-level APIs. Rados Block Devices (RBD) exposes a block device service on top of `librados`. A particular design goal for Ceph was the provision of scalable storage on top of commodity hardware.
The RADOS architecture primarily consists of Object Storage Devices (OSDs) and a small uneven number of monitoring daemons on the backend which are serving the clients. OSDs store and manage data, while monitors are responsible to curate the so-called cluster map which is used to determine system health and to establish consistency based on the Paxos part-time parliament algorithm [Lamport, 1998].
Being an object store a flat namespace is used which does not support a directory hierarchy, although some metadata for objects is allowed in the form of key/value pairs. An object ID is unique across the entire storage cluster. OSDs can be set up hierarchically for transparent replication for high availability and data safety. If a CephFS is configured it is necessary to set up a Metadata Server (MDS) daemon in addition to OSDs and monitors. Ceph uses the deterministic algorithm CRUSH [Weil et al., 2006] to make data placement decisions while being able to scale the system up and down. This way data is distributed uniformly across targets for high performance without the need to issue metadata requests to a more central authority as the placement calculation can be performed by the clients themselves, as is illustrated in Figure 5.23. As with other declustered RAID approaches, this does occasionally require migrations when the system configuration is changed, but not as much as in other solutions. CRUSH is a good example for a decision point in a storage system, although providing a better workflow-aware alternative remains an open research question.

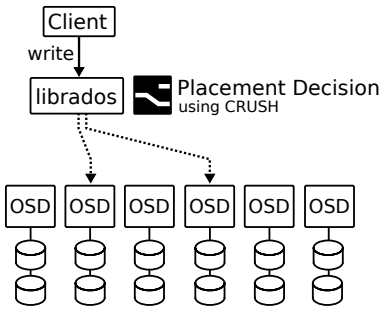


Figure 5.23: Data placement decisions have to be made all the time in storage system. In Ceph this decision point is governed by the CRUSH algorithm which allows to calculate the placement decision on the clients without requiring a central authority.

OpenStack Swift: Swift is a object storage service developed as part of OpenStack, but it is also routinely deployed independently of an OpenStack installation if a scalable open-source object storage solution is required. Swift provides its own API but can also be used to expose an S3 interface. As a result, a Swift storage system does not necessarily implement the original Swift architecture. Some deployments, for example, use a Ceph based Swift object store instead, even in full OpenStack deployments.
If the Swift reference implementation is used the architecture consists of three core components as illustrated in Figure 5.24: At least two *proxy servers* for load balancing and fail-over, the storage cluster network, and potentially multiple *rings* of Swift servers, which are the actual storage targets. Using the concept of rings, Swift can be configured to offer different storage zones, which offer different quality of service and achieve performance isolation for critical storage services such as account databases. Swift can be configured to use dedicated additional networks, for example, for replication.

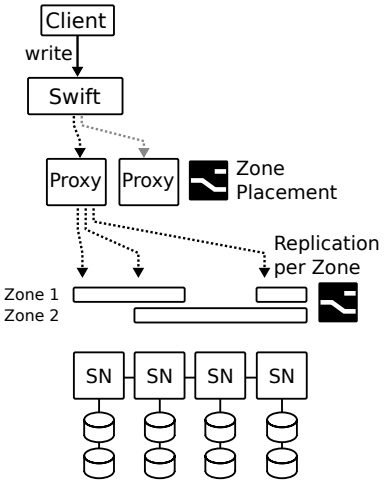


Figure 5.24: Load balancing and placement decisions that might occur at Swift's proxy layer as well as replication decision which are on a zone to zone basis.

Distributed Services Summary

Particularly relevant distributed services for workflow I/O optimization are *resource management* and *storage systems*. Both contain countless *information sources* as well as *decision points*. Resource management and scheduling have a particular high impact on I/O performance, as granted resources determine network contention in space and time. Storage systems, while they allow to collect performance implications, provide various tunables to control how data is mapped to physical resources. This mapping determines the degree of parallelism and thus peak I/O performance for data access. Although many distributed storage systems share fundamental similarities, tuning strategies vary a lot not only by choice of vendor but also by other deployment choices, thus requiring system-specific decision support.

5.8 Holistic Approaches

The different points to collect information or to influence decisions typically only provide artifacts to a more complete picture. In this section, a number of tools to assist in capturing a comprehensive picture of I/O activity throughout an HPC data center are discussed.
Increasingly, users and system researchers seek holistic or general-purpose approaches to improve performance and to orchestrate, control, and analyze systems. Tools to monitor system health come closest to offering such a holistic perspective, but these tools usually take the system perspective but do not relate telemetry to a user's workflow.
There are also a number of risk factors to consider when developing or procuring holistic products, in particular, because of phenomena such as "feature creep" and "vendor lock-in". In the case of feature creep, a software project will pick up on countless responsibilities beyond its original mission. Vendor lock-in occurs when mission-critical hardware and software components may not be easily isolated and integrated into new systems provided by competitors or developed in-house. Often a migration to more open ecosystems is complicated by a range of factors, which usually are a combination of time to solution requirements, trust and risk considerations towards open source solutions, personal preferences,and interoperability with existing organizational structures.
None the less, in support of scientific workflows holistic approaches can offer many advantages as they allow to streamline and co-design for a range of requirements. In addition, increasingly holistic approaches are combined from many smaller building blocks. In this section the discussion focuses on general-purpose to *monitoring and telemetry collection*, holistic approaches to *storage services* as well as *cloud-like HPC provisioning* which integrates both perspectives.

5.8.1 Monitoring and Telemetry Collectors

Collecting, aggregating and visualizing monitoring and telemetry information in distributed environments is routinely performed and typically attempts to provide a more holistic overview by including various services relevant for deployment. As a result, there is a wide spectrum of tools and services around to do so, often developed because other existing solutions did not take certain domain-specific requirements into account or because tapping into existing solutions was not easily possible. A closer look reveals

that tools on the visualization and interaction end often consolidate or build upon many smaller tools. An example is Nagios [Nagios Enterprise, 2019] which can be extended with collector modules for various services. On top of Nagios, tools like Ganglia [Massie et al., 2004] optimize the collection and presentation for clustered environments.
More recently, collectors in combination with flexible query functionality to assist higher-level tools and tasks such as monitoring, performance analysis, and performance prediction are becoming more popular. Two relevant information sources in this space are *collectd* and *Prometheus*.

collectd: As a daemon *collectd* [collectd, 2020] is used to periodically collect system statistics for various services and store them in a structured formats such as round-robin databases (RRD)[1] or comma seperated values (CSV). Information sources include the OS, user applications, log files or external devices and network services. Various third-party tools are building monitoring, alerting and performance analysis infrastructure on top of collectd. Collectd offers more than 160 plugins for various services, which to a large extent are contributed by the community.
For custom use cases and not yet supported services, collectd offers a plugin system which allows developers to provide a subset of the following callbacks depending on the functionality their plugin should provide: `init` for initialization, `read` to receive values, `write` to write values, `flush` when caching is supported, `shutdown` for cleanup. In addition, there is a logging interface `log` with various modes (ERR, WARN, NOTICE, INFO, DEBUG). These plugins then can be stacked, for example to perform aggregations such provided by calcNum, calcSum, calcAvg, calcMin, calcMax, calcStddev. A number of special plugins integrate interpreters to allow scripted collectors in higher-level languages such as Lua, Python, or shell script.

[1] Instead of accumulating an rapidly increasing amount of log data, RDD consolidates and eventually overwrites the oldest data in a round-robin fashion. While subject to configuration, it is common to find minute resolution statistics for the current day, hourly averages for the current month, but only daily averages for the entire year.

Prometheus: Prometheus describes itself as a systems monitoring and alerting toolkit. Originating in cloud environments, Prometheus [Prometheus Authors, 2019b] is becoming also very popular to monitor HPC sites. The collection architecture shares many similarities to collectd, and can also integrate data from collectd. Custom telemetry providers can use a special API to ingest data which is then stored in a structured form to allow querying data for monitoring, alerting and performance analysis. While primarily an *information source*, alerts can be used also as trigger points to automate actions, as such they can be sometimes treated as *decision points* useful from a workflow perspective. Prometheus officially supports client libraries in Go, Java/Scala, Python and Ruby. Countless third-party plugins offer additional client bindings in Bash, C/C++, Rust and many more. Prometheus comes with multi-dimensional data models to support time-series data for counters, gauges, histograms, and summaries and allows to associate data through adapters for a wide variety of instrumentation. While jobs and instances can be associated with collected data, custom fields to allow task or workflow granularity are not directly supported. This can, however, be addressed by conventions on how to name labels. This data model exposes its full potential in combination with Prometheus's query interface PromQL which third-party applications can use to export data. A native feature-limited web UI is maintained, but many third-party tools such as Grafana and other APIs are relying on Prometheus.

Total Knowledge of I/O (TOKIO): A telemetry collection framework specifically targeting I/O and storage in HPC environments is the TOKIO framework [Glenn K. Lockwood et al., 2018]. It brings together monitoring information from multiple sources throughout the data center, as illustrated in Figure 5.25. This is necessary because the different storage subsystems often provide their own, sometimes proprietary, mechanisms to query backend I/O monitoring data.
TOKIO is also notable because it combines user-level instrumentation with privileged monitoring information from I/O subsystems and vendor APIs through so-called connectors: On the application level TOKIO collects this data using Darshan. In addition, site-local services such as Slurm, Lustre, HPSS and inter-site services such as Globus and ESnet are considered.
The analysis and correlation of activity is mostly offline, but the benefit of continuously monitoring a data center over time demonstratedly can be used to detect performance regressions, for example, after software updates to the storage system [Lockwood et al., 2018]. TOKIO also offers tools to analyze and aggregate collected log records.

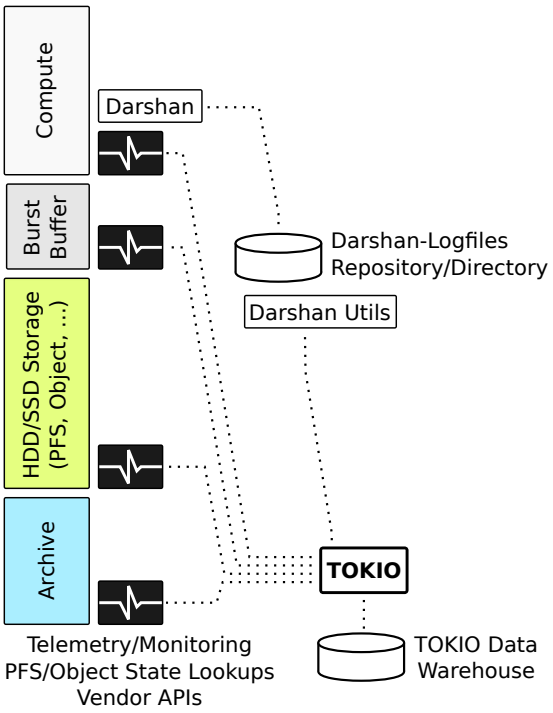


Figure 5.25: TOKIO takes a holistic approach to I/O activity capture throughout the data center. To do so, TOKIO collects data from different data sources, such as system and service logs, vendor APIs, PFS monitoring tools, and Darshan log files.

5.8.2 *Policy Systems on Top of Heterogenous Storage*

As the diversity in storage services deployed at a site increases, solutions to integrate multiple storage systems are being developed. Often these systems will be responsible to trigger large data movements between different storage tiers or even across sites. On which tier or service data is residing can have a large impact on data access performance, which makes staging and data placement important *decision points*. Typically, these high-level integration services will provide policy engines and quality of service features.

Site-Local Policy Engines: A variety of policy engines are part of existing multi-tier storage systems or as third-party solutions such as Starfish. As these often are proprietary it is hard to make assumptions about their integration. Starfish features a rich GUI to browse the unified namespace,

statistics but also to configure triggers and policies. An open source policy service specifically designed for POSIX filesystems with many advanced features to support Lustre is the *Robinhood* policy engine [Leibovici, 2015], which also establishes a policy system to generate statistics, enforce quotas or help with the creation of snapshots of the Lustre namespace or POSIX metadata. To do so, Robinhood is periodically scanning the file system and can be configured to issue alerts, which makes it an attractive information source for file system state information over time. Robinhood can integrate with HSM and thus be used for data staging. As such for slowly evolving *decision points*, modifications to the file system namespace can potentially be efficiently performed through systems like Robinhood. Robinhood can be easily extended through plugins. Some policy engines, such as iRODs are extending beyond a single site or system.

Integrated Rule-Oriented Data System (iRODS): A community built metadata-driven federated open source storage system, with a strong focus on supporting high-level global scientific and institutional workflows and data exchange, is iRODS [iRODS Consortium, 2019]. While originating in efforts for long-term archival for physics data, it is now heavily used by users especially in the life sciences but also many other domains.

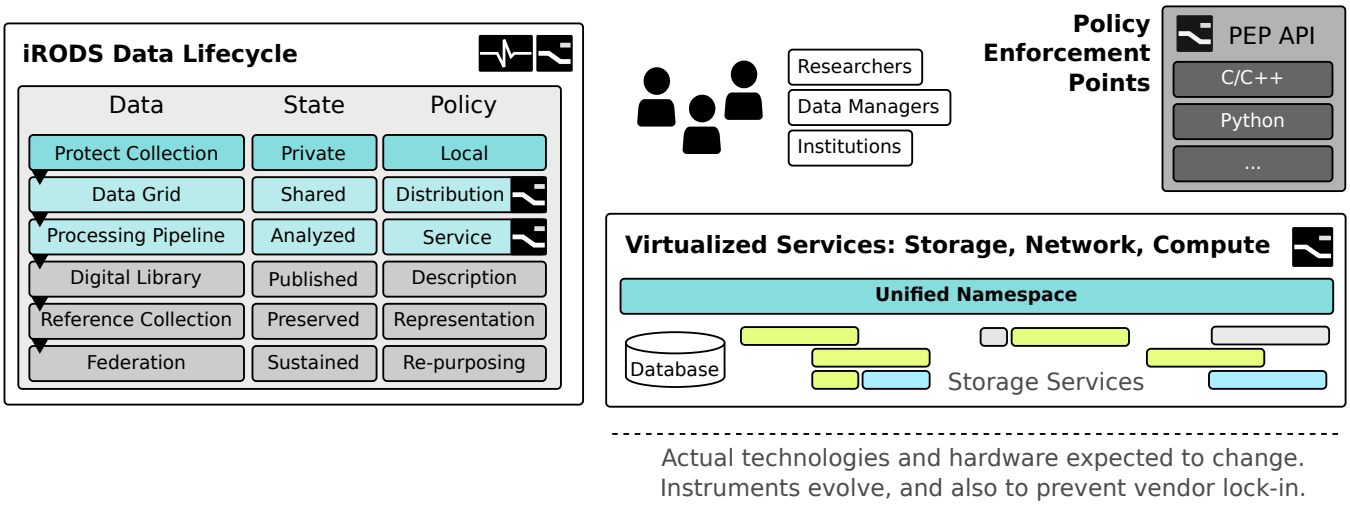


Figure 5.26: Illustration of the iRODS data lifecycle in relation to user roles and the virtualized perspective on actual storage and network resources. By defining policies in, for example, C++ or Python at so-called Policy Enforcement Points high-level scientific workflows can be supported by system automation while also ensuring metadata capture or compliance requirements.

As illustrated in Figure 5.26, iRODS establishes a unified namespace for research objects on top of existing storage services and systems. It deliberately takes a fully virtualized perspective onto storage, network, and data generation/collection systems, to accommodate changing science infrastructure but also to prevent vendor lock-in. While most ingested data today is file-based, adapters to support object storage and even RDMA for highest performance are in development [iRODS Consortium, 2019]. Different deployment patterns have been developed to help organizations get started without requiring to commit to adopting all assumptions made by iRODS. As such HPC systems, for example, can be easily integrated out-of-band by orienting on existing data-to-compute or compute-to-data deployment patterns and thus avoid sacrificing performance, for example by tapping into routine scans of storage resources and then ingest asynchronously.
The system does not aim to be a purely technical solution but instead anticipates user roles and cross-organizational access control. Roles might include researchers but also data managers, which are accommodated in

workflows to help mapping to existing institutional structures, but also to realize compliance frameworks such as the FAIR data principles [Wilkinson et al., 2016].
At the core of iRODS is an assumed data lifecycle around which metadata and data services are established. These are then controlled by a policy system to automate metadata capture on ingest, as well as data verification later on, to enforce compliance to scientific, privacy, or legal requirements. For this iRODs offers a policy rule language [Lampa, 2014] which might be used to control data routing, movement, synchronization as well as metadata capture, application and verification. These policies are then triggered automatically at so-called Policy Enforcement Points (PEPs) as iRODS initiates different system activities. Examples include the automatic extraction of metadata, the pre-processing of data as it is uploaded, but also the transfer of ownership of data as users join or leave an organization. In addition to allowing to chain these rules, provenance and change logs are kept for auditing.

5.8.3 Holistic and Co-Design for Memory and Storage

There also exist a number of holistic approaches to storage systems. It is important to realize that holistic in this context is a moving target, as some of the storage solutions discussed are building blocks in a larger product line which would cover a number of use-cases with varying degrees of automation. As most of these solutions, however, make no attempt at accounting for the data structures and meaning of data they are not covered here as holistic in regard to scientific workflows. Most of the systems which are listed as holistic approaches here are also a thin layer on top of many existing technologies. An exception to a degree might be DAOS, which however is still in an early development and not fully deployed at any production site yet.

Distributed Asynchronous Object Storage (DAOS): An end-to-end storage approach co-designed to accommodate scientific data on top of persistent memory technologies is DAOS [Intel et al., 2014]. As with other solutions that offer greater flexibility, DAOS takes a virtualized view onto storage resources through so-called pools, which act as reservations and thus achieve more predictable capacity and performance across multiple resources. It is assumed that a pool would roughly map to a project, so that this could be a decision point which might adjust pool configuration based on observed behavior. A DAOS system is designed to accommodate multiple users which share resources.
A DAOS architecture overview is illustrated in Figure 5.27 together with key technologies and potential *information sources* and *decision points* relevant from a workflow perspective highlighted. DAOS assumes scientific workflows and offers a number of leverage points to take advantage of how data is structured in comparison to block storage services.
For this, DAOS introduces the concepts of containers and also offers a data model library that provides primitives to map data to arrays, KeyValue-stores as well as multi-level KV stores. A container, for example, might be accessed through the HDF5 interface, but objects and metadata are not handled exclusively by the HDF5 library but also by the storage stack which allows optimizing dataset distribution on, for example, a per-dataset basis.
To achieve high performance, DAOS deliberately uses only NVRAM (here

3D XPoint) and SSDs although integration of other technologies is possible through SPDK and PMDK. Based on these technologies DAOS scales through so-called DAOS nodes (DN), on top of which storage resources are virtualized and only accessed through pools. DNs can be deployed either pooled or in a hyper-converged setup depending on the use case.
Interaction with clients and within DAOS uses RDMA through libfabric as a data plane and maintains a separate control plane which uses remote-procedure calls on top of gRPC. All communication is non-blocking, to promote interleaving compute and I/O. In many cases, this is expected to achieve zero-copy often bypassing RAM all-together.
For legacy applications, there remains POSIX access through a FUSE library or `libdfs`. No integration of HDDs is assumed, although there is a data mover service to integrate with external storage tiers such as Lustre or an archive. Depending on the size of the DAOS deployment a workflow might use this to stage resources before execution starts.

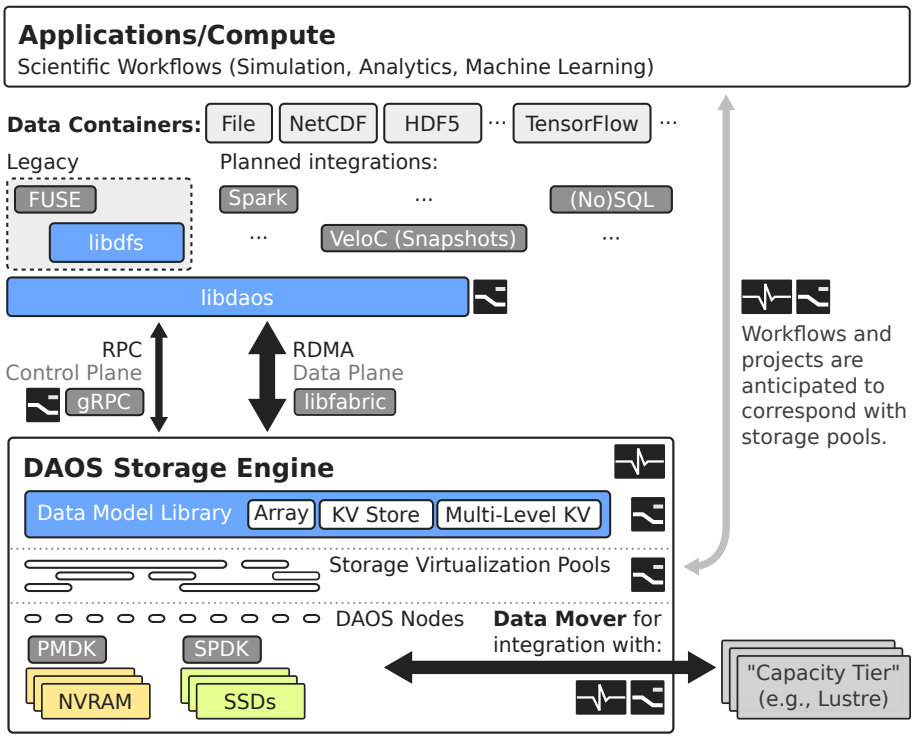


Figure 5.27: High-level overview of DAOS with key decision points and information sources added. Combination of information found in source code as well as various DAOS documentation. DAOS is still under development, so some of the suggested integrations might never materialize.

5.8.4 *Cloud-like HPC Provisioning*

OpenStack is an exception to many other projects discussed so far, as it is a solution to provision and manage clouds. As such it is a mix of countless smaller projects which provide building blocks for a wide variety of different services. OpenStack is mostly written in Python, and in many cases coordinates with already existing well-established interfaces, tools and low-level functionality offered by operating systems, storage services, and others. Different vendors offer their own flavors of OpenStack with minor variations. Services providing core functionality include:

- *Compute*: VMs, Containers or *serverless/Function as a Service* (FaaS)
- *Networking*: Software-defined networking, load balancing, name services

- *Storage*: Object storage, block storage and file storage, backups and image inventories for containers/VMs
- *Shared Services*: access control, resource inventory, allocation helpers
- *Management and Frontends*: *orchestration*, migration, monitoring, accounting and policies

The above list is not exhaustive, and OpenStack often uses names for different services that do not correspond directly to their application. A reduced logical overview of the OpenStack ecosystem is illustrated in Figure 5.28, which illustrates how many traditional services are combined together to provide the cloud experience.
Highlighted are *information sources* and *decision points* which spread across storage and orchestration services:

Information Sources: In principle, many of the previously discussed telemetry sources apply to cloud environments as well. But because many clouds are operating on a measured service model[2], that is the system is monitored for optimization and to bill customers based on their resource consumption, telemetry capture and reporting to a *(central) telemetry service* is implemented and activated in many cloud sub-services by default. Depending on the exposed granularity, this is an attractive information source also for workflow optimization. It should be noted, that it is often still appropriate to extend this high-level information with more fine-grained information from other layers when taking a workflow perspective. In the case of OpenStack, the telemetry service is called Ceilometer which features a collector interface and can then be coupled with notifications to trigger alarms or other services.

[2] The US National Institute of Standards and Technology defines five essential characteristics to define what constitutes a cloud: A cloud offers users *on-demand self-service*, *broad network access*, *resource pooling*, and *rapid elasticity* in a *measured service* environment [Mell and Grance, 2011]. The definition also discusses different service and deployment models.

Decision Points: Decision points within cloud deployments span the entire service landscape as illustrated in Figure 5.28. When taking a workflow and storage perspective, it becomes apparent that there are sometimes multiple places to enact an intervention. Ultimately, the orchestration service or the bare-metal service are responsible to allocate resources and thus are often able to directly influence the configuration. Adaptation, however, might often take place at a lower level, which might be more appropriately performed by a specific service.
In OpenStack, the workflow perspective is accommodated to an extent, for example, when using the secondary orchestration services such as *Heat*, or the specifically aimed at defining workflows within OpenStack *Mistral* service.
OpenStack *Mistral* is a high-level orchestration service in OpenStack to complement other orchestration services such as Heat, Senlin, and Blazar, which offer mostly provisioning automation instead of application workflows. Mistral does not take a holistic view directly but is developed with the embedding into the OpenStack ecosystem in mind which allows associating telemetry, tasks, and triggers far more effortlessly because many core concepts are identical across orchestration, monitoring, and management services.
Workflow definitions can use a combination of YAML and YAQL to achieve forking, joining, loops as well as workflow nesting. Two notable features are actions and task policies. Actions are a way to control what is to be done when a task starts, thus they become potential *decision points* for interventions. Policies are hooks offered by the task model to react to events such

as successful completion, errors, timeouts of tasks, but also allow to fine-tune behavior such as adding delays before and after a task. A heartbeat mechanism is used to detect if an action fails. Integration with event and notifications services allows to create webhooks. Repetitive actions can be cached, for example, when using OpenStacks FaaS solution Qinling.

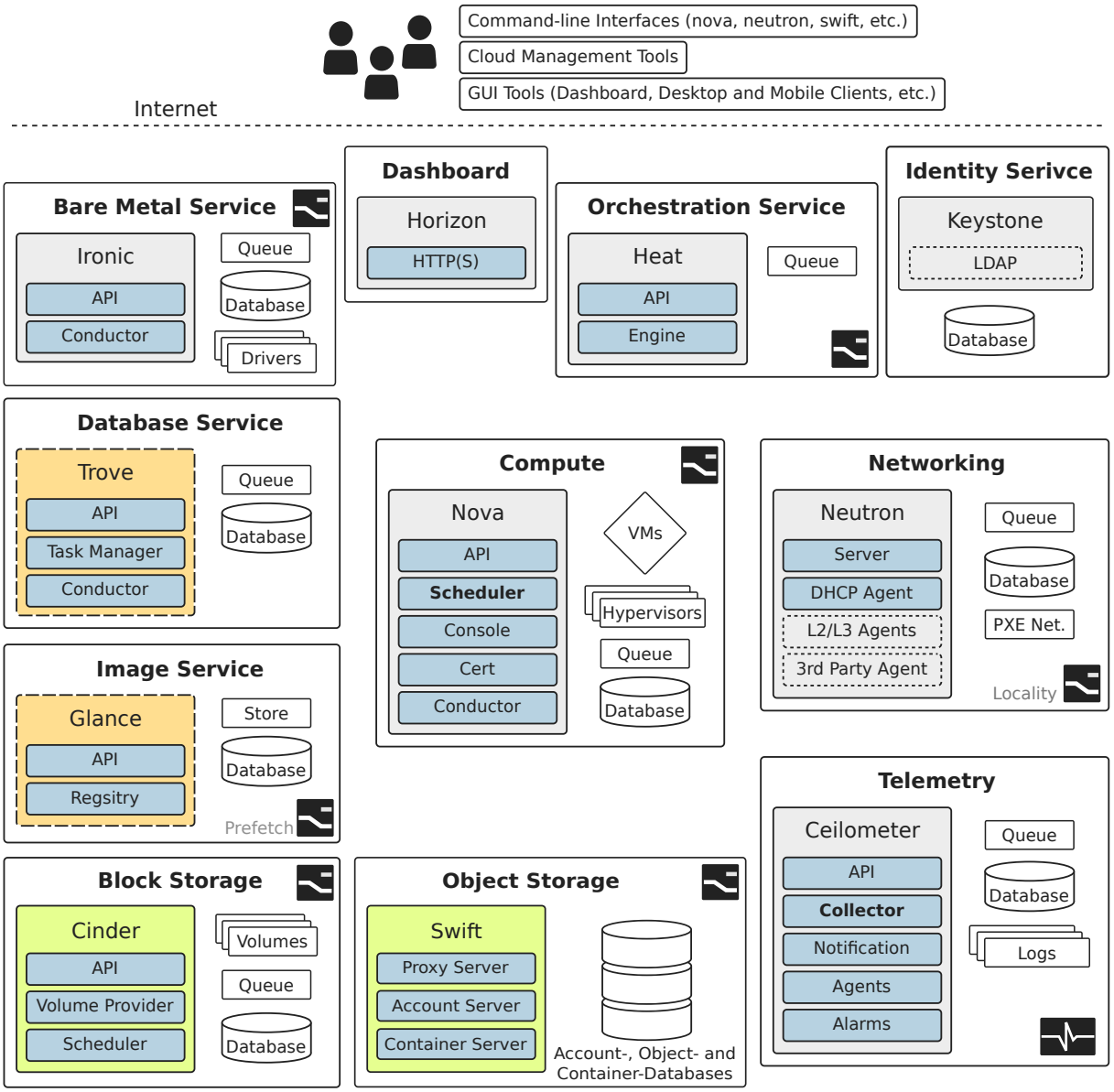


Figure 5.28: Overview of the logical relationships of OpenStack core services as they are combined to provide a cloud. Figure derived from [OpenStack, 2018]. There are decision points spread throughout the stack. While there is a global telemetry service it is mostly fine-tuned for accounting and billing, which often offer more detailed resource usage statistics than provided with many HPC sites.

Holistic Approaches Summary

Holistic approaches are still in their infancy and often take a micro-service-based approach which allows remixing specialized solutions which is in contrast to the traditional model followed by many workflow engines which usually aim to provide a one-stop solution.
Four types of holistic approaches that provide *information sources* or contain impactful *decision points* for workflow I/O optimization have been considered. *Telemetry services* are helpful to consolidate and streamline the collection of I/O performance implications. *Policy systems* for storage systems, expose hooks to manipulate higher-level data management decision. A relatively new breed of *co-designed* or *cloud-like* provisioning of storage

materializes better performance switching to virtualized architectures which offer more control to provide customized quality of service.

Chapter Summary

This chapter surveyed the storage path to identify *information sources* but also *decision points* spread throughout storage stacks to inform the APIs and services to enable workflow-aware storage systems that are introduced in the next chapter. When it comes to information sources an important observation to make is that it is typically only possible to collect artifacts that reveal only little information about a larger workflow and that one should take into account whether these artifacts are explicitly workflow-related or only can be implicitly linked to a workflow. Both are important because it is not feasible nor desirable to explicitly define every detail, especially as the underlying hardware and software technologies are constantly evolving.
Information sources and decision points are then grouped into different levels, corresponding roughly to some of the layers which commonly operate isolated of each other only communicating with their direct neighboring layers. In distributed systems, this poses some challenges because there are services that take a broader scope.
The survey starts with the *workflow and user level* which captures most of the high-level logical relationships but hides many undeclared inputs and outputs which are very important to gauge performance implications. These high-level relationships give rise to many decisions a workflow engine makes to map the logical workflow onto the physically available resources. As there is sometimes multiple layers even within the workflow level, for example, when workflows are nested or implementation details are "imported" from applications, ignorance of other orchestrating actors poses challenges most likely overcome by community standards to express workflow relationships.
On the *application-level*, most of these undeclared implications start to materialize, which then can be captured using various means of instrumentation and logging. A challenge for almost all subsequent layers is that often it is not easy to associate an activity with the workflow context and that it is usually not possible without introducing some overhead.
As some responsibility is delegated to the *middleware-level* or the operating system or *node level*, the line between node-, middleware- and application level optimizations is often blurry. An important caveat, however, is that any change on the middleware- or node- or service-level will affect many applications. As such targeting to optimize or adaptively swap how decisions are made at those levels, potentially benefits also applications and workflows which share similarities.
As it comes to the *distributed service level*, systems are taking a more global perspective, although typically within a narrow mandate which collects only information to monitor the service state and then base a service-exclusive decision on this. The logic for these, however, is spread throughout different clients and servers which participate in providing the service. The contextual information to relate activity back to a workflow is almost never propagated this far. Although efforts to correlate the activity from different services in offline analysis are being performed.
Such efforts attempt to take a *holistic perspective*, but typically do not directly consider decision-making in support of scientific workflows. First up are

abstractions and processing pipelines to collect and analyze monitoring and telemetry. Based on telemetry data collected across different services that do not share concepts explicitly, workflow activity can be correlated often leading to a better understanding of the system interactions.
Finally, there are co-design efforts that design storage systems specifically for HPC, analysis, and machine learning workloads which to a degree rely on virtualization. And then there are technologies originating from cloud computing and storage which were not directly targeting HPC but increasingly influence how HPC is designed. By means of virtualization, resource sharing becomes manageable, also because migrating a resource from one physical location to another can be achieved transparently if compatible technologies are being employed by users on higher-levels. From this perspective, clouds are a collection of services that will share more concepts and thus allow to consider optimizations that are not maintainable in traditional HPC deployments. Cloud-like or more specifically *service-oriented deployments*, also help with decision-making and workflow artifact collection when thriving for workflow-aware data handling.

6
Architecture for Workflow-Aware Decision Components

This chapter introduces an architecture proposal for different addons modernizing the storage stack to allow workflow-aware decision making. Somewhat simplified, it is assumed that automating an optimization opportunity in many cases entails a sequence of steps similar to the following:

1. *Discover on optimization opportunity for a given workflow.*
2. *Learn to automatically detect if a known opportunity is present.*
3. *Propagate this knowledge to decision components benefiting from it.*
4. *Learn to map an opportunity to the most appropriate action and act.*

To enable each of these steps a wide range of tooling and some interface extensions are required which are motivated in the following sections (compare Figure 6.1):

- *Section 6.1 starts with an overview of the architecture, the most important stakeholders, and a number of guiding use cases.*
- *General tooling requirements are briefly covered in Section 6.2.*
- *Utilizing advisory knowledge for desirable system behavior requires awareness of available actions. Therefore, an abstract decision component interface, the Decision API, is proposed and discussed in Section 6.3.*
- *How to represent workflow relevant knowledge and advisories by finding a compromise between telemetry and learning affinity is covered in Section 6.4.*
- *A message-oriented middleware, the Decision Aid Propagation Service (DAPS), tailored for the distribution of workflow knowledge to relevant decision components is proposed in Section 6.5.*
- *As only a few learning algorithms are tailored for HPC storage and system optimization, requirements for a learning environment friendly for HPC and workflow use-cases are discussed in Section 6.6.*
- *Finally, training of learning algorithms typically requires millions of training examples. A possible path to accelerate and pre-train decision components is offered by turning to system simulation which is discussed in Section 6.7.*

"There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies." – C.A.R. Hoare

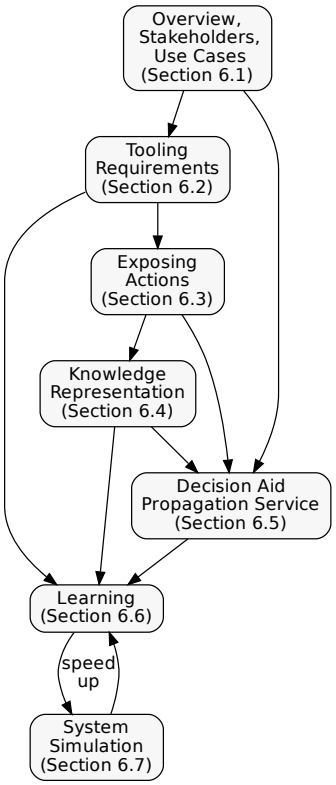


Figure 6.1: Relationship of the different sections in this chapter.

6.1 Overview and Building Blocks

Chapter 3 (Scientific Workflows in High Performance Computing) together with Chapter 5 (Information Sources and Decision Points) disassembled scientific workflows and analyzed artifacts that allow inferring I/O performance implications. This knowledge about workflow I/O behavior in turn can be useful for decision components. This chapter combines the information collected so far and compiles them into an architecture proposal that aims to allow automated workflow-aware decision-making along the storage pathway. Special care was taken to ensure that the proposed architecture can be integrated gradually so that applications and middleware can maintain compatibility to legacy applications.

Figure 6.3 attempts to give a visual overview of the overall architecture including the core stakeholders, tooling, interfaces, and involved systems which are also listed in Table 6.1:

Component Name	Description
❶ Stakeholders	Centered at the top are various *stakeholders* with different objectives which are discussed in detail in Section 6.1.3. Stakeholders include scientists, developers as well as I/O researchers. They all seek different insights and intervention opportunities in different parts of the larger system.
❷ Analysis and Learning Tools	The large black boxes at the center roughly group different sets of *tooling and auxiliary* such as aggregation helpers, knowledge representation, visualization helpers, and learning templates.
❸ Knowledge and Decision API	Two central components are the *Decision API* and a *Decision Aid Propagation Service* (DAPS) which streamlines the gathering and propagation of workflow observations and knowledge to interested decision components.
❹ System Simulation	Coupling *system simulations* with learning methods is likely essential to pre-train decision components before they are deployed in production.
Decision Components/Points Information Sources	Telemetry providers and decision components are scattered across the system when deployed, but one should think of them as a pool of plug-able *adaptive or neural heuristics* researchers have at their disposal.

Table 6.1: Description of core components to enable workflow-sensitive decision making.

6.1.1 Optimization Opportunities of Adaptive Heuristics

In Chapter 5 (Information Sources and Decision Points) storage-related *decision points* throughout the storage stack have been identified. I/O related workflow optimization will typically involve finding a trade-off between space and time utilization:

- *Time* related decisions: When to access data? When and how often to transform data (possibly exploiting locality)?
- *Space* related decisions: Where to store and how to distribute data? Which data representation and which transformation to use?

Fortunately, even seemingly expensive transformations can be worthwhile due to the asymmetry between compute and network/storage performance. Transformations, however, have to consider the time it takes to perform them, to reverse them, and the power required to perform either of them.

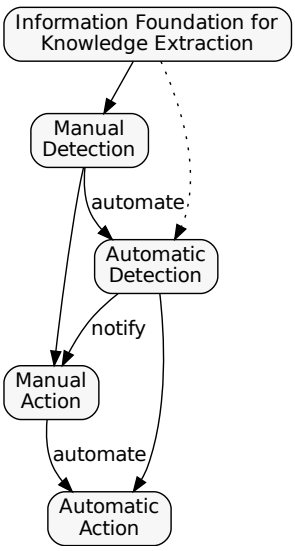


Figure 6.2: Automating workflow-sensitive interventions requires to 1) automate detecting an opportunity as well as 2) deciding on an appropriate action to pursue.

6.1.2 Workflow for the Development of Decision Aids

The auxiliary tools, the Decision API and the Message-Oriented Middleware are designed in support of a systematized workflow which is illustrated in Figure 6.2: Analysis of telemetry data and workflows gives rise to optimization opportunities which in many cases are assumed to be discovered manually. Before a decision component can act, it has to detect if the circumstances for optimization are met. In a second step, a mapping from different circumstances to possible actions can be learned. It should be noted that this attempt to systematize discovering and deploying optimizations may appear to be simple and straightforward, but currently would require intricate knowledge, experience, and intuition in system architecture, software development, and learning methods.

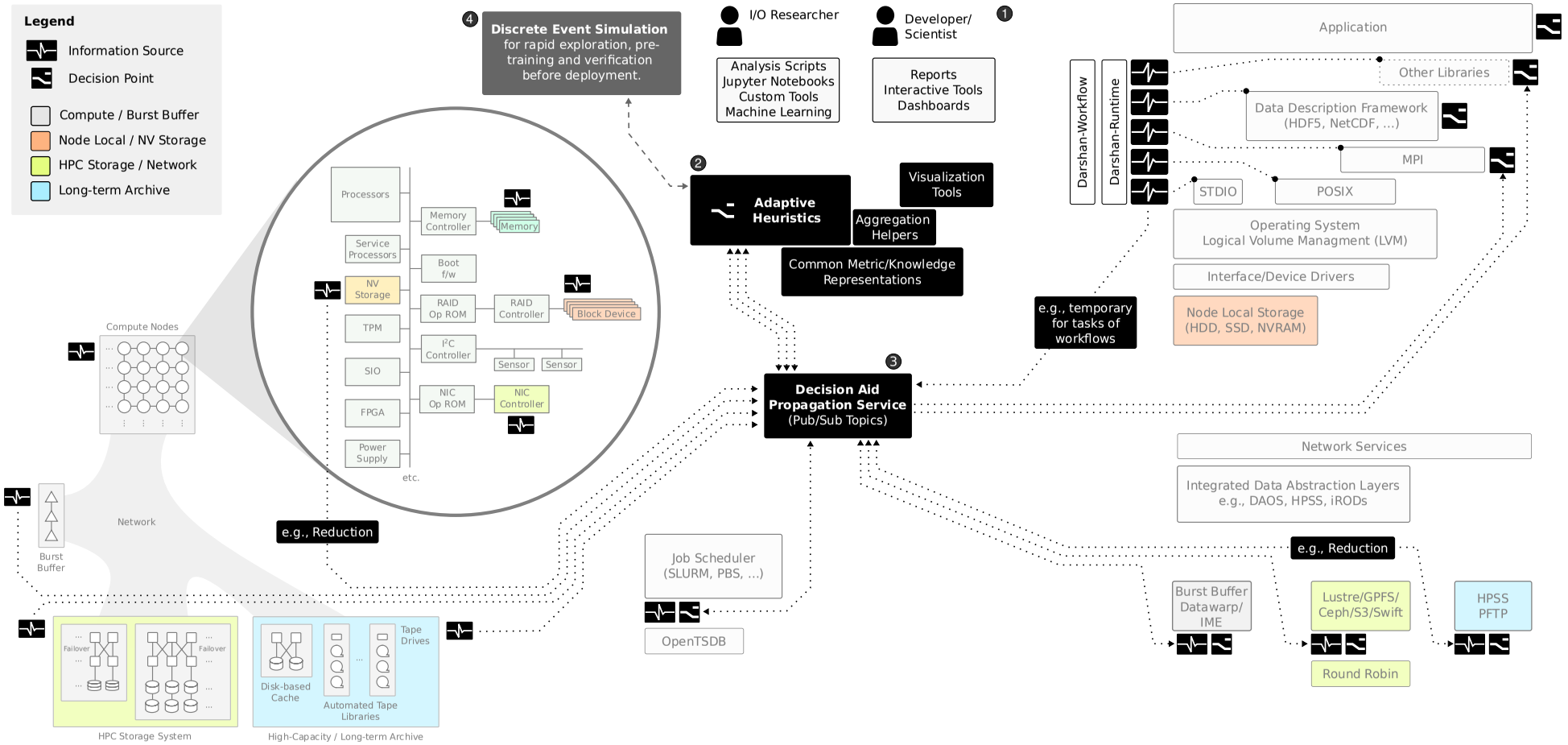


Figure 6.3: Overview of a workflow-aware storage system stack with proposed components and methods highlighted including references to the respective subsections. For details on the greyed-out parts of the graphic refer to Figure 5.7 in Chapter 5 (Information Sources and Decision Points).

6.1.3 *Stakeholders*

In this section, the different stakeholders which are expected to benefit from a more workflow-sensitive storage stack in data centers are briefly discussed. Figure 6.4 illustrates the relation of different stakeholders and technologies which are roughly associated with the following layers:

- Application Layer: Scientists, workflows, applications, and libraries.
- Control Layer: Developers of system libraries, operating systems, and operators of network services.
- Infrastructure Layer: Operators and vendors of computing, network, and storage hardware.

Table 6.2 lists the stakeholders by their role and gives a brief description regarding their expectations and what functionality they likely prioritize the most. Middleware developers are placed in between the application and the control layer because some middleware is closer to applications while other middleware is closer to the systems and services. Similarly, the site operating is placed in between the control and the infrastructure layer because they not only reconfigure and fine-tune existing systems but also operate and procure new infrastructure.

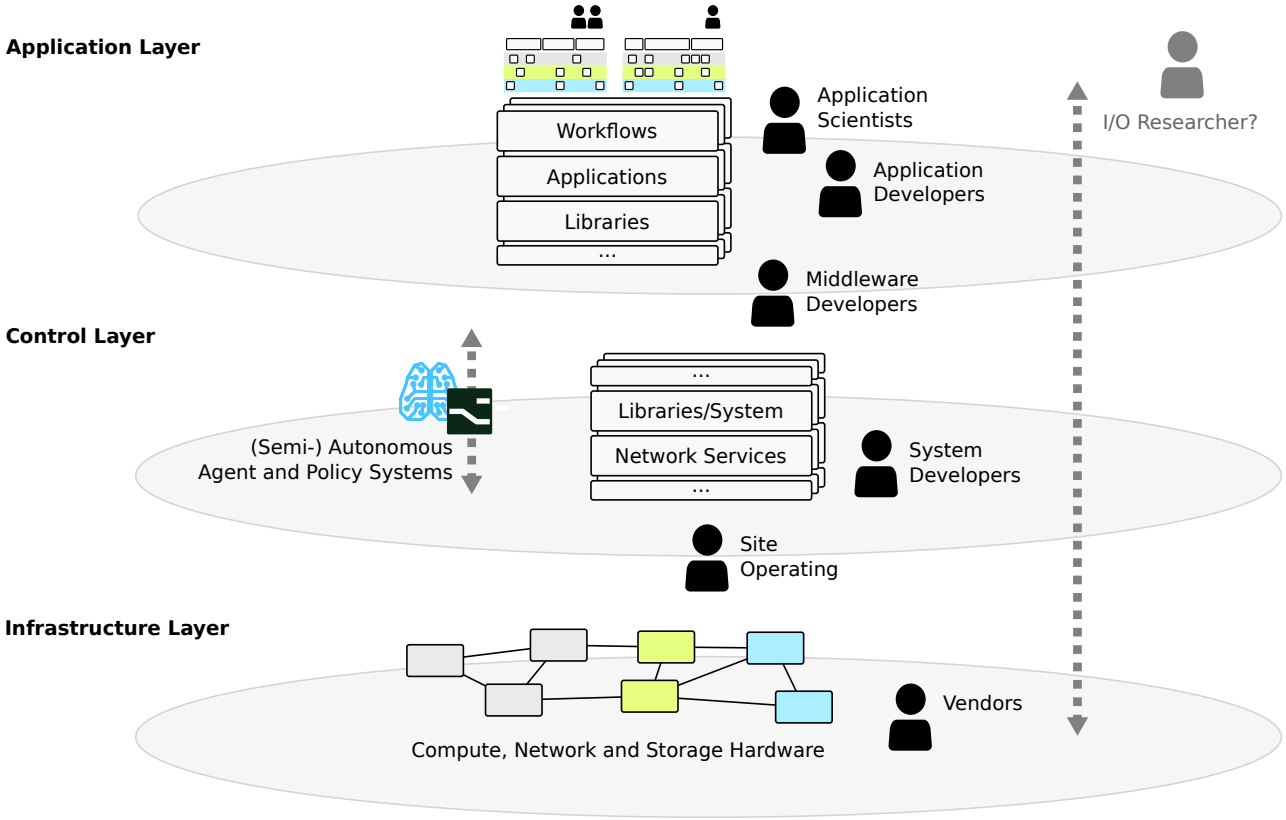


Figure 6.4: Stakeholders and the subsystem they care about the most. Each stakeholder may consult with or become an I/O researcher as troubleshooting or designing becomes necessary.

To an extent, automated technical systems are expected to be able to make increasingly complex decisions. Therefore it seems adequate to assume that agents and stakeholders may be not only humans but also technical systems. Still, the core stakeholders considered are Application Scientists, Application Developers, Middleware and System Developers, Site Operators, and Vendors. Any of these might be required to continuously or occasionally investigate I/O, in many cases across multiple layers in the software stack.

Role/Name	Description and Expectations
Application Scientist	Application scientists formulate the workflows and coordinate experiments and compute runs. They care about performance feedback for two reasons: 1. lowering turn around times yields useable results quicker 2. reduce cost when billed by compute hours and storage quota Offering hints for optimization might reduce cost. They would find a report reflecting the logical tasks of a workflow highlighting cost and parameter improvement opportunities helpful. While usually not involved with application or system-specific optimization, an application scientist may share additional roles such as being an application developer. Allowing users to label data interactively and influence decision components actively can help to empower users and prevent resentment with transparent optimization.
Application Developer	Application developers will expect storage systems and middleware to come with sane default behavior while offering fine-tuning options on demand. Reports can include more detail, such as profiling and tracing information for troubleshooting and debugging. Feedback should include tunables for middleware such as HDF5, MPI, or Lustre. An application developer will often be responsible for a limited number of applications. Often any optimization that improves performance will be considered acceptable, but this might neglect side effects and negative impact on other users.
Middleware/System Developer	Adding to the expectations of application developers, middleware and system developers will expect far more control over options to fine-tune underlying systems. A middleware, being used by different applications usually will have a multi-application perspective. Adopting and deprecating features will be done only after rigorous testing and evaluation to maintain compatibility. In addition, performance regressions have a larger impact as many users are affected. To select the right tuneables I/O middleware requires decision components to make decisions. This can include layout/fragmentation, schedule, tier/service selection decisions, which usually will be evaluated against performance.
Site Operator	Site operators require tools to enact policies and configuration of different systems. They have to moderate application behavior of different users and ensure stable system operation. Working with special privileges, they have more statistics to base decisions on. Policy decisions are evaluated by their impact on performance and total cost of ownership. Operators are consulted to troubleshoot middleware or even application problems.
Vendor	Vendors are often contracted to offer support for various subsystems, which might include researching default policies and configurations. Many proprietary systems will expose limited statistics and telemetry information to customers or operators via special APIs. When expecting vendors to integrate pluggable decision components, the underlying software framework has to come with a license compatible to commercial requirements.
I/O Researcher	Each of the previously listed stakeholders might find themselves in the role of an I/O Researcher. Usually, this will require taking a systems perspective including analyzing statistics and telemetry information. While there exist countless tools (compare Chapter 5), I/O researchers routinely will need to piece together a holistic view for themselves. A toolchain of common but customize-able analysis and visualization building blocks would be needed. For exploration of new strategies, system simulations would be helpful.

Table 6.2: A list of stakeholder with a brief description and their differing respective requirements.

6.1.4 Use Cases

There is an abundance of potential workflow related advisory products that would benefit existing systems. This section describes four guiding use cases relevant for the following architecture discussion in detail. A collection of additional candidates for advisory products useful in storage stacks, based on the information sources and decision components identified in Chapter 5 (Information Sources and Decision Points) includes:

- Detection, indication, and reporting of tasks that are issuing large amounts of random I/O as candidates for optimization.
- Advice on the most suitable storage system from a list of available systems in a heterogeneous data center.
- Recommendation of HDF5 chunking options. Useful, for example, with write-once-read-many (WORM) datasets and workflows where multiple follow-up tasks might access the data by using distinct access patterns.
- Detection of inadequate Lustre stripe configurations as well as offering better recommendations.
- Detection of reliance on POSIX features such as timestamps, for example, for locking/semaphore files or diagnostics. In addition, an alternative service or feature could be recommended if available.
- Detection of inappropriate use of MPI collective/independent I/O as well as overlapping reads on the same file in a parallel application.
- Recommendation of the optimal checkpoint frequency [Daly, 2006] based on observed reads/writes. (This requires explicit declaration or reliable recognition of checkpoint files; compare Section 7.3.3).

This list is not exhaustive and only includes a selection of example use cases especially relevant in current storage systems.

6.1.5 Use Case: I/O Aware Scheduling and Staging

Among the most impactful systems with a more global perspective is the workload scheduler. However, when considering the features offered by state of the art workload schedulers as discussed in Section 5.7.1 (Resource Management & Scheduling) and Section 3.3.2 (Comparison of Workflow Engines) the I/O perspective, and particularly storage I/O are only sparsely addressed.

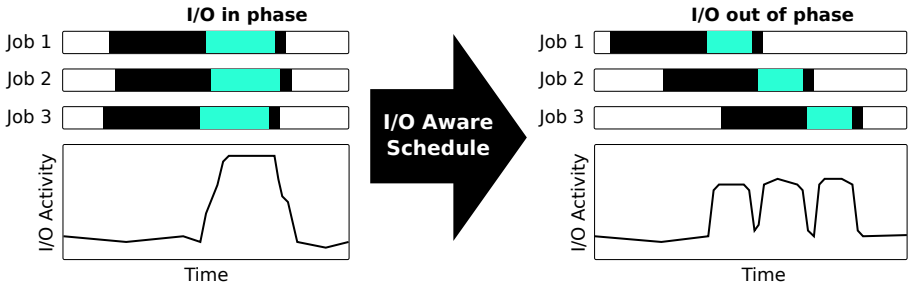


Figure 6.5: Sketch of I/O-aware scheduling using knowledge about workflows to potentially increase performance and reduce I/O time per job or allow procuring a scaled-down storage system.

In a scheduling scenario, knowledge about the I/O phases of a task, job, or pipeline offers interesting opportunities to adapt scheduling decisions. Figure 6.5 illustrates an example that assumes a number of independent jobs

that are submitted to a batch scheduling system. Assuming a workflow that experiences bursty I/O, this approach could help the scheduler spread jobs over time in order to reduce stress on the file system. To do so, the scheduler could ensure that different jobs are started out of phase in order to spread out I/O more evenly over time. This approach could allow operators to offer a more predictable quality of service or to reduce cost by scaling down system size since peak performance requirements could be relaxed. This use case is already useful without considering workflows in particular. But a workflow perspective offers to target situations where the impact of an optimization is multiplied. In addition, overhead can be controlled more easily, as looping and cyclic workflows make it straightforward to instrument a certain percentage of jobs. Knowledge about I/O phases also offers opportunities beyond fine-granular scheduling decisions, for example, to better plan when to allocate burst buffers, when to stage and unstage from cold to hot tiers and how to phase-shift jobs requesting shared resources.

6.1.6 Use Case: Workflow-Sensitive Striping/Representation

With data producers and consumers ignorant of each other, except for a common data format, producers usually determine the on-disk representation. In this scenario, a producer's best effort for lack of additional knowledge would be to write data into a serialization that offers the highest performance for writing data. For certain time-critical situations, this can be the most appropriate option to reach data volume and data frequency requirements. If a slightly asynchronous processing pipeline is used, this strategy usually is not the most cost-efficient, especially when multiple consumers are using a similar access pattern.
Figure 6.6 illustrates a simple workflow with a Producer P and two types of consumers A and B. The domain decomposition of the producer is split into four square regions, while consumer A reads vertical slices and consumer B operates on a subset of horizontal slices. The data is written only once, while it is consumed many times over. Here it is assumed 100 consumers would favor a representation optimal similar to consumer A , while only 5 consumers would favor a representation optimal for B.
Instead of letting the producer choose the representation, it would be possible to save on an amortized basis to store the dataset in a representation that caters to the requirements of consumer A as illustrated in Figure 6.7. Ideally, for P there would be a way to delegate the decision to a third party which is aware of the workflow.

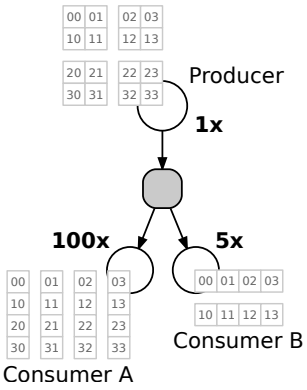


Figure 6.6: Example workflow with a single writer, and two readers with different magnitude and access pattern.

Figure 6.7: Sketch of workflow-aware choice of data representation and striping. Typically, the producer decides which on disk representation is used. Including workflow knowledge would favor a different representation which considers what would be beneficial for a common consumer.

While depicted as simple in this example, a decision like this might need to take into account many additional dimensions such as the network topology, available storage tiers or performance variability. This example looked at a single workflow and assumed complete knowledge. In practice, not all accesses can be anticipated or recorded so the scope of finding optimizations has to be limited. Examples could be aiming to find patterns on a user,

group, project, application or workflow level.

6.1.7 *Use Case: Decisions in I/O Middleware*

Along the I/O paths to storage systems countless decision components are active. In many cases these are embedded in I/O middleware such as DAOS [Intel et al., 2014], HDF5 [HDF Group, 2019], ADIOS [Liu et al., 2014; ADIOS2, 2018], Decaf [Peterka et al., 2018] or ESDM [ESiWACE, 2016]. As a result, I/O-related middleware is an attractive target to apply optimizations that benefit a large number of applications. The preferred intervention point for the previous use case, Section 6.1.6 (Use Case: Workflow-Sensitive Striping/Representation), too, would be in a data description library. Figure 6.8 illustrates how an I/O middleware might mediate between applications and the complexity of the underlying storage services. Advanced I/O middleware such as HDF5 might come with various internal heuristics and optimizations for different storage services while still relying on users to choose which mode to use.

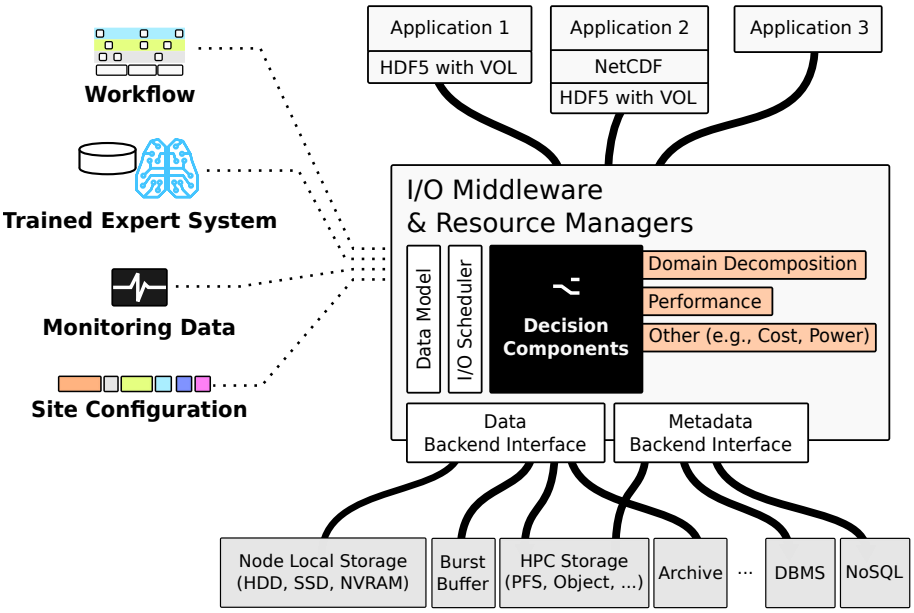


Figure 6.8: I/O middleware considering knowledge about workflow I/O to make data placement or transformation (e.g., enabling compression) decisions.

By considering knowledge about the different processing steps of long-living data in workflows, a number of decisions could be improved:

- Domain decomposition/transformation on write
- Target selection, for example, recommending service, client/server, OST
- Handling separation of data, for example, metadata vs. data
- Automatic population of data catalog services

In particular, applications with high degrees of similarity and minor variation in the structure of output data are attractive for this approach. For uncertainty quantification (UQ) and high-throughput computing (HTC) workflows, for example, a pipeline may be executed hundreds or thousands of times. Because doing so yields multiple observations per task and pipeline, we can be more confident in the expected behavior and slowly and potentially automatically adapt the system behavior to better accommodate this workload. The I/O middleware might need to consider information

from multiple sources, such the workflow, an expert system, monitoring or the site configuration, as is illustrated to the left side of Figure 6.8.

6.1.8 Use Case: Asynchronous I/O, Low-Level Devices, and Offloading

While still speculative, a number of changes and additions to existing compute and storage architectures are anticipated or promoted within the community or by vendors. Examples are affordable low-power compute chips and FPGAs which open the possibility to re-configurable networks and to perform computations and transformations in transit or at the edge. For storage systems, this, for example, might become relevant in *active storage* devices which also would include low-cost compute chips able to perform aggregations, compression, or encryption. Active storage in particular benefits from taking a workflow perspective as data locality and placement are determining how data is being reduced.
Similarly, special-purpose acceleration chips for neural networks promise to at least allow inference tasks along the storage path, raising questions if and how learned heuristics might be exploited in-band. That this is not necessarily a distant vision is supported by network- and MPI offloading and interfaces for asynchronous I/O. Analog to I/O middleware, usually the host system or independent firmware is driving the decisions that have to be made on how data is handled. As a result, this adds additional intervention points to influence behavior.
For data to be handled asynchronously like this, additional metadata has to be carried. In the case of MPI offloading, for example, the network firmware or hardware to an extent is able to unpack MPI communication to identify recipients or communicators. Similarly, active storage devices can perform useful computations such as computing an average as is illustrated in Figure 6.9, but this requires information about the stored data. This additional information can also be used to reshape serializations adaptively. An idling disk might use access statistics to replicate or to split and merge chunks of data. In addition, to depending on device access statistics, a storage device might as well consider workflow information either from a workflow management platform or from metadata stored along with the data.
In transit, workflow information may indicate the data is expected to be used rarely which might be a sufficient criterium to apply a slightly more expensive but better compression algorithm automatically and then relocate it to a cold storage tier.

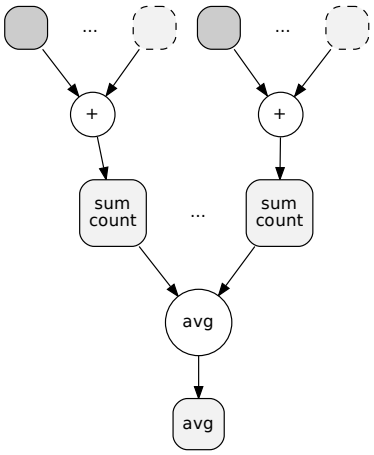


Figure 6.9: Workflow perspective for the calculation of an average as it might be performed by a cluster of active storage devices. Unlike more traditional post-processing approaches that perform all aggregations on a set of compute nodes, a task-based programming paradigm is helpful to distribute some computations closer to the physical location of data. This way only the intermediate results need to be transmitted in contrast to the entire dataset.

6.2 Tooling and Software Components

In support of the different use cases and stakeholders, a variety of complementary tooling and software services is required which are briefly introduced here. In particular this includes discovery tools to analyze workflow artifacts, a Decision API to declare decision points and allows easy enumeration, a Decision Aid Propagation Service (DAPS) to allow the efficient exchange of advisory products, and finally concrete or abstract implementations of decision aids for training with existing frameworks. An overview including more detailed descriptions for the different tooling requirements is listed in Table 6.3 with details in subsequent subsections. In Chapter 7 (Evaluation) selected examples and prototype implementations for each of the components are discussed in more detail.

Component Categorie	Description
Discovery Tools	A toolbox to analyze and visualize the workflow perspective. This includes post-processing tools to aggregate log and instrumentation data.
Abstract Decision API	Applying optimization strategies usually requires to change the code base of applications, services, libraries or even the operating systems. An abstraction for decision components is proposed, which assumes decision components are going to be modularized and easily exchangeable.
Decision Aid Propagation Service	In many cases, it would be desirable to make decisions which are based on non-local knowledge: A special message-oriented middleware is proposed which propagates different performance relevant signals to decision components throughout the storage stack. This simplifies integration with expert systems and provides flexibility for the future.
Learning Tools / Decision Aids	With exchangeable decision components, tools for their creation are needed. As manual optimization ought to be minimized means to train decision components by example or by automatic exploration via reinforcement learning or genetic algorithms should be considered.

Table 6.3: Core components to enable workflow-sensitive decision making.

In the following sections, each of those components will be described in more detail. The relationship between the different tools is illustrated in Figure 6.10. Developers, operators, and researchers, placed at the top, will typically investigate I/O and storage systems to either:

- Improve application/library performance: Here feedback is desired to verify if I/O bound problems are present. They should be relate-able to the business logic but allow diving into system metrics if necessary.
- Improve system performance: Operators and researchers often will use custom analysis scripts and tools, analysis increasingly also may include machine learning methods and Jupyter notebooks to document results.

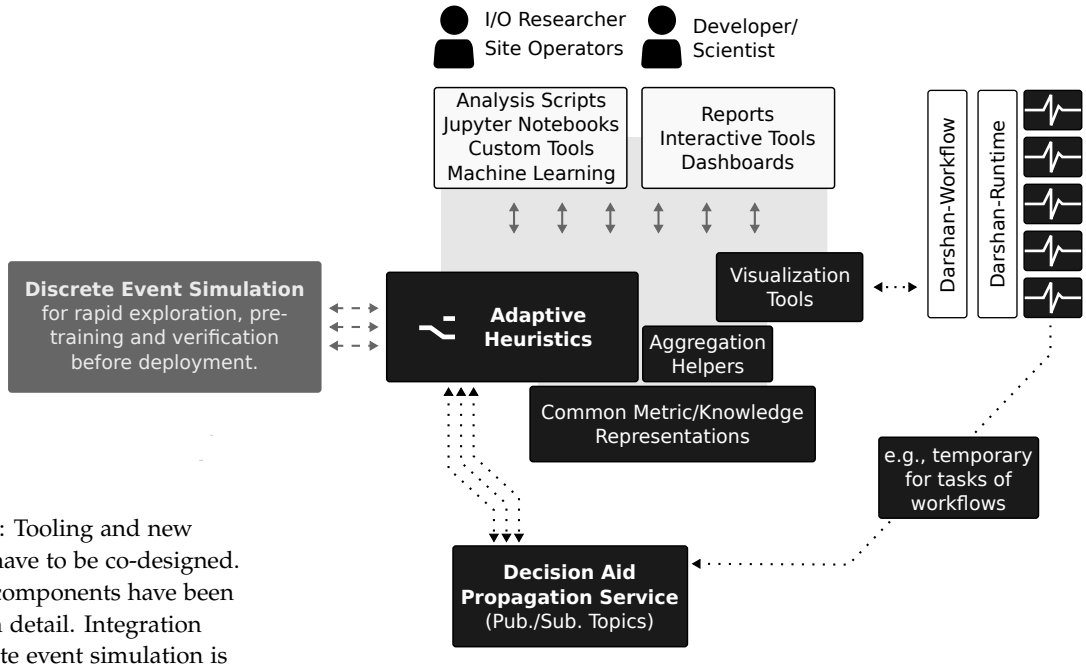


Figure 6.10: Tooling and new interfaces have to be co-designed. The black components have been explored in detail. Integration with discrete event simulation is anticipated to become important for training, but is well beyond the scope of the thesis.

6.2.1 *Discovery, Visualization and Analysis*

At the beginning of the optimization discovery process analysis tools and visualizations are important. But in particular instrumentation, aggregation, and visualization helpers are not strictly limited to discovering and automating optimizations, compare Section 6.1.2 (Workflow for the Development of Decision Aids). While many useful tools in this space already exist, they are usually not easily integrated outside of their respective software suite. Chapter 7 (Evaluation) showcases a number of prototype tools, which employ web-based technologies to maximize customizability and accessibility so that they can be used and modified by the community. Sharing similar technology, some of these were packaged as widgets in Jupyter notebooks, but can also be reused in monitoring dashboards used by the site operating, such as Grafana or Nagios.

6.2.2 *Integration Helpers & Message Oriented Middleware*

On the operational side a number of new independent interfaces and services are proposed to help with the integration of existing software while maintaining the flexibility to adapt to future architecture changes:

- A new interface, the *Decision Support API*, for middleware and libraries to explicitly expose possible decision points, actions, and tunables but also to request a decision for an exposed action, as a way to allow decision components to be easily swapped out, see Section 6.3 (Actions & Decision Points).
- A service, the *Decision Aid Propagation Service* (DAPS), to distribute workflow knowledge or system telemetry for decision components to resource managers, middleware, and applications, to base decisions on, see Section 6.5 (Decision Aid Propagation Service).

6.2.3 *Learning Tools and Decision Aids*

Compatibility to learning tools in support of an optimization feedback loop as illustrated in Figure 6.11 will be discussed in more detail in Section 6.6 (Learning & Decision Components). Considering interoperability with machine learning is mandated by a lack of experts, short supercomputer life-time, and the increasing complexity and heterogeneity of systems. While machine learning research, in particular in the field of deep learning, yielded very promising new approaches for classification, prediction, regression, pattern recognition or clustering, they are rarely deployed in storage systems.
This can be attributed to a lack of research adapting methods for HPC system optimization. Storage systems in particular suffer a high barrier of entry for practical reasons like limited availability of training data, but also because of the complex interactions of different factors such as the network topology or contention for shared resources.
On the one hand, it is necessary to collect and generalize machine learning approaches applicable to storage systems into reusable templates for decision aids. On the other hand, tools to reshape data and train data are necessary. A particular aspect to this is integration with discrete event simulation of systems to offer sufficient training experiences, see Section 6.7 (Integration with Synthetic Benchmarks & System Simulation).

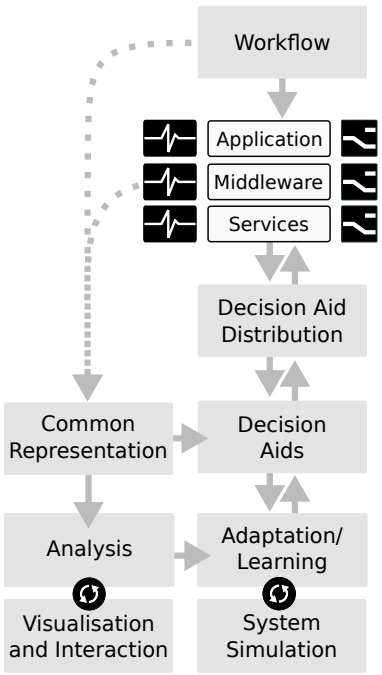


Figure 6.11: Illustration how different high-level tasks require tooling to allow for end-to-end approaches to learn and tune decision aids and utilize them to optimize workflows. A more detailed variant of this figure in relation to the tooling ecosystem is Figure 6.30 on page 147.

6.3 Actions & Decision Points

"If you automate a mess, you get an automated mess." – Rod Michael

With tools to discover opportunities to optimize systems to better accommodate workflows one of the largest challenges is:

- *How to turn a number of observations into useful action?*

Periodically and manually surveying the storage stack collecting optimization opportunities and potential decision points is no feasible solution considering the fast-changing nature and complexity of the software and hardware ecosystem surrounding HPC systems.
While the fully automatic discovery of opportunities and resulting actions might become a possibility one day, current technologies lack the cognitive depth to sufficiently reason and identify actions and decision points. Evolutionary approaches or brute-force may be an option to discover some decision points, for example by correlating conditional branching and subsequent function calls with performance measurements. In practice, the unfolding combinatorial space even within simple programs, let alone a complete software stack down to the storage systems, also render these approaches unfeasible.
Instead, decision points should be explicitly declared where applications or libraries can select among multiple actions. This way developers are offered a mechanism to declare intervention points together with a default decision that can be overruled. While this works best with high acceptance by the developer community for wide-spread adoption, it is already beneficial if only applied gradually in a few pivotal software libraries and services such as:

- I/O Middleware and Data Description Frameworks
- Resource Management Systems and Workflow Engines
- Storage Systems and Related Services (for example, Robinhood)

Some of these systems already offer facilities to influence internal behavior, compare Chapter 5 (Information Sources and Decision Points), through a variety of means such as hints, policies, callbacks/hooks, plugins or configuration files.
A user or developer who manually tunes a system often does so based on information that is inaccessible to the individual subsystems themselves. Often this information is not machine-accessible at all. As a result, many decision components currently consider only local knowledge, either local to a node, or at least to a subsystem of a supercomputer. However, when considering the current job allocation or the whole workflow, additional information needs to be fed to decision components. Chapter 5 (Information Sources and Decision Points) made clear that local situational knowledge is usually insufficient to make workflow sensitive decisions. This deficiency can be overcome by introducing the Decision Aid Propagation Service (DAPS), a message-oriented middleware that allows decision components to imitate extending the decision context to include outside knowledge a programmer would usually apply.
Besides opportunities where the action taken is only slowly changing, there are also actions benefiting from very short living information often because they are highly dependent on context. This includes opportunities that are

affected by the topology of the compute, network, and storage systems. For example, jobs submitted to a batch scheduling system may share the domain decomposition, so that the logical relationship of processing elements may be invariant: For example, the lowest or highest ranks of an application are often responsible for some coordination tasks. But the mapping to physical compute resources may vary considerably as the node list changes, especially in heterogeneous clusters featuring a variety of different node types. The result is that contention points vary from allocation to allocation. This is especially relevant for coupled processing workloads, such as multi-physics simulations as well as in situ workflows.
Application developers, ideally, would not be required to be concerned about many of the underlying complexities. This is especially true for application development when expecting to be used across multiple sites and potentially over decades. This among other factors contributes to the need to reconsider who, or which algorithms, should be responsible for making decisions in large distributed systems. Most generally, the goal is to take the burden to consider a multi-dimensional optimization problem away from the users. It is important, however, to allow users and operators to intervene if they do wish so.

6.3.1 Toward Abstract Decision Components

This section motivates the need and then introduces an interface proposal for abstract decision components. With optimal decisions depending on the workflow and on many deployment-specific details, decision components need to be easily adaptable and replaceable. This requires some generalization of decision components with stable interfaces.
Figure 6.12 depicts a proposal for an abstract decision component interface useful for I/O handling. As illustrated, I/O calls most often pass a pointer or reference to a *data buffer* along with a number of *arguments* that instruct the middleware or system library on how to handle the request. To keep decision components as simple as possible, the complexity of obtaining additional information, for example about the workflow, is delegated to and handled by a special service, the Decision Aid Propagation Service (DAPS), which is introduced in Section 6.5 (Decision Aid Propagation Service).

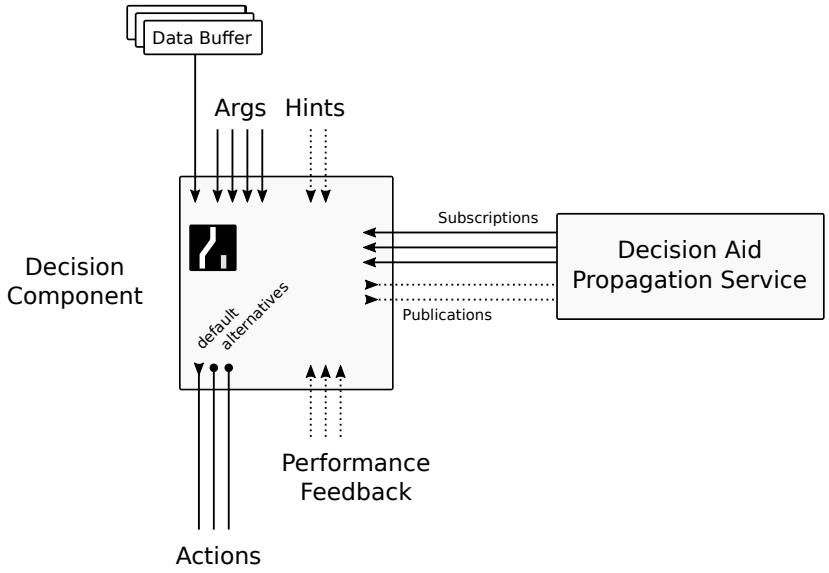


Figure 6.12: Schematic of an abstraction for a generic decision component suitable for a storage perspective. Storage APIs typically reference a buffer for data, along with additional arguments describing the data shape or controlling how the request is handled. In many cases, an I/O middleware might have multiple possible paths of action for which a decision component would provide a mapping.

Some I/O interfaces also accept additional *hints*, such as POSIX-`fadvise` or HDF5 via property lists. The combination of arguments and hints then determine which different branches (here denoted *actions*) can be taken to process the request. Various modern storage systems (for example, DAOS), as well as data description frameworks (HDF5), offer a variety of complex data structures beyond byte streams such as array-like (n-dimensional), tree-structured, relational data or key/value pairs.
Depending on the workflow and the bottlenecks for a given system under a particular allocation, different combinations of those might yield the best performance. Unfortunately, topology or workflow context is usually not accessible to many decision components on the storage pathway. As such, which options apply is either on the discretion of the calling applications or depends on a configuration of a privileged system. The result is that large parts of a request cannot be controlled on a case-to-case basis which would consider more global context. One way to do so is by subscribing to a message-oriented middleware that offers a variety of topics that can be used as decision aids.
Finally, each action may be associated with *performance feedback* for different input parameters of a request. A decision component at runtime would not perform analysis of all available performance records but instead would consume an aggregated value that serves as a *decision aid*. How such a decision aid might be represented is discussed in Section 6.4 (Knowledge Representation & Decision Aids). This value could be the expected performance for a given parameter combination. A decision component in this context then would map various inputs onto the most appropriate action. This is a task for which methods from the field of machine learning, and neural networks in particular, are well-equipped for already. A more in-depth discussion on how training these decision components follows in Section 6.6 (Learning & Decision Components).

6.3.2 *Distinguishing Semantics for Read and Write Paths*

When it comes to making decisions, it is useful to distinguish the *read* and *write pathways* to avoid applying optimizations that can render data inaccessible later on. This section discusses two particularly important exemplary scenarios where requests to two different storage services Storage *A* and Storage *B* are optimized for different access semantics. A middleware seeking to adaptively choose the most appropriate storage target for a given application might delegate this decision to a decision component.
This decision component would issue recommendations on where to store the different pieces of data. The problem here is that there is no well-defined responsibility on how to ensure coherence on where data is living. To ensure data is not silently corrupted, as data is moved, or as recommendations evolve as the system changes, modifications which can not recover if a recommendation is at fault should be applied conservatively:

- *Safe*: Changes that can be stored in metadata of the I/O system in charge of holding the data are typically safe because reconstruction does not depend on the availability of the decision aid.

- *Potentially Unsafe*: Changes that are only to be obtained via a decision aid carry the risk of preventing reconstruction. Data integrity should not depend on volatile information as may occur when the decision aid is

unavailable or evolves over time as the system changes.

Consider the following two failure cases also illustrated in Figure 6.13 as well as an allowable optimization attempt on the read path:

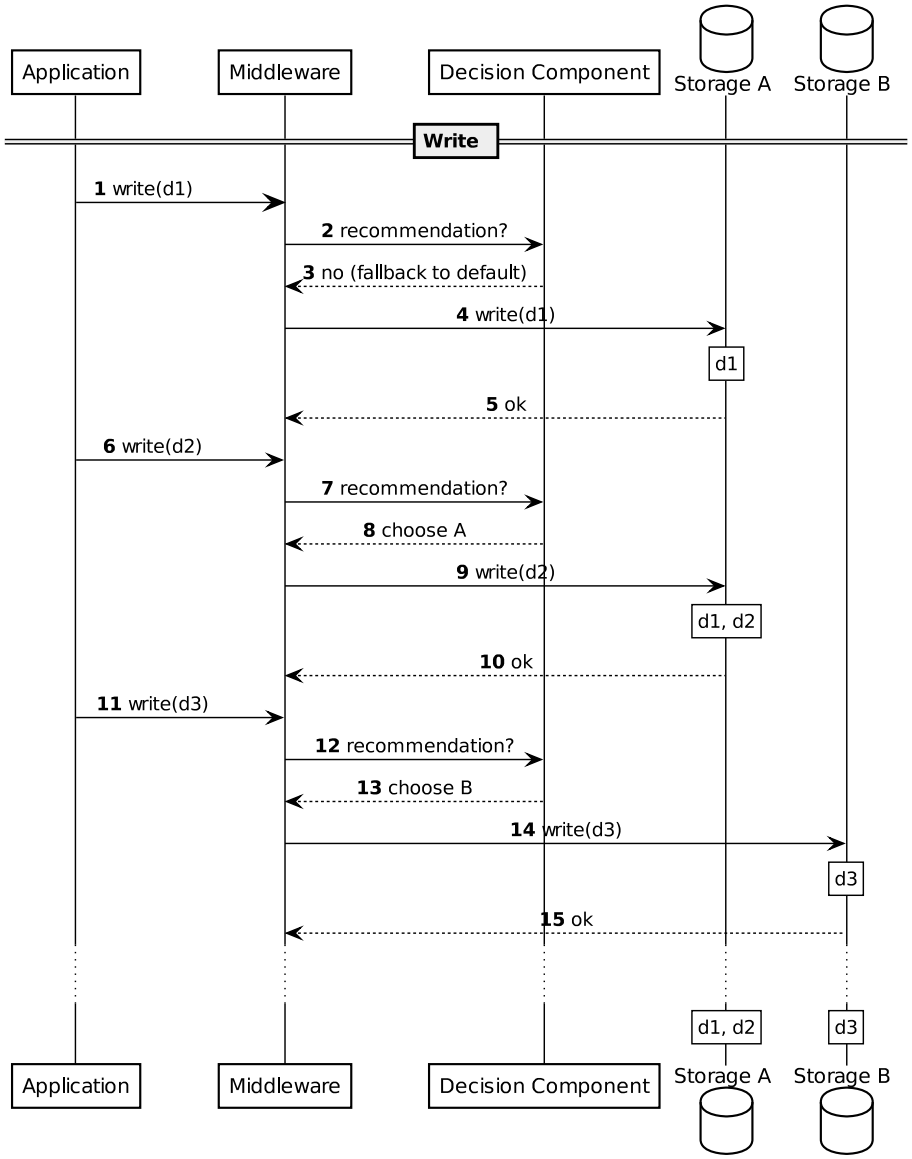


Figure 6.13: Possible recommendation optimized write requests, which might fail in the future if no graceful degradation in case of failing decision services are established. In this scenario, data can be distributed across two storage targets A and B which is aided by the recommendation service.

Failure Case: Read (Service Unavailable): Services in distributed service can become temporarily unavailable. As the application requests to read data (see Mark 1, Figure 6.14), the middleware would request a recommendation, but this time the service does not respond so that the reconstruction fails. When using the recommendation services it is therefore advisable to not depend on it. Phrased differently, decision components should not be abused as a lookup service, if no graceful fallback is established.

Failure Case: Read (Decision Evolution): Especially, when working with legacy systems, it may be tempting to achieving workflow adaptivity by trying to coordinate and orchestrate multiple decisions. Over time, the systems, as well as learned patterns, may change and result in different recommendations. In such a case, the application would request data (see mark 3, Figure 6.15), and obtain a recommendation that the decision component returns. But because

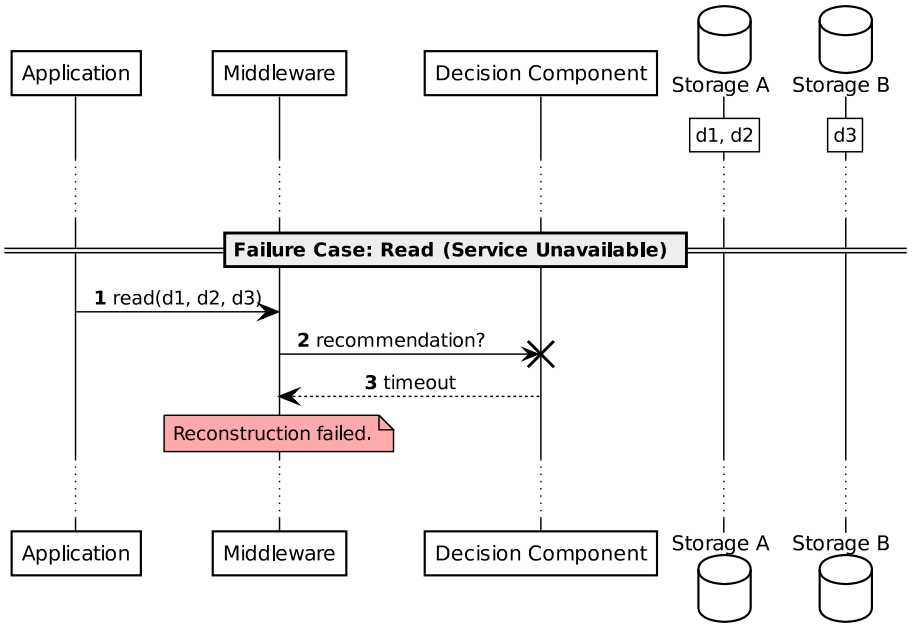


Figure 6.14: Illustration of the Service-Unavailable failure case on the read path in continuation of Figure 6.13. If reconstructing the data relies on the availability of the recommendation service or the decision aid (see mark 2 of this figure) then reconstruction can fail. Thus a fallback should be established on the read path even though it might be less favorable from a performance perspective.

the decision component/aid evolved, it might indicate that storage system *B* is the most suitable available tier, but if the data was actually stored to *A* the follow-up request points to the wrong storage service so that the reconstruction fails. As a result, when considering a decision aid for similar situations, it should be seen as a heuristic that might fail, but if it does not potentially allows to save time or effort.

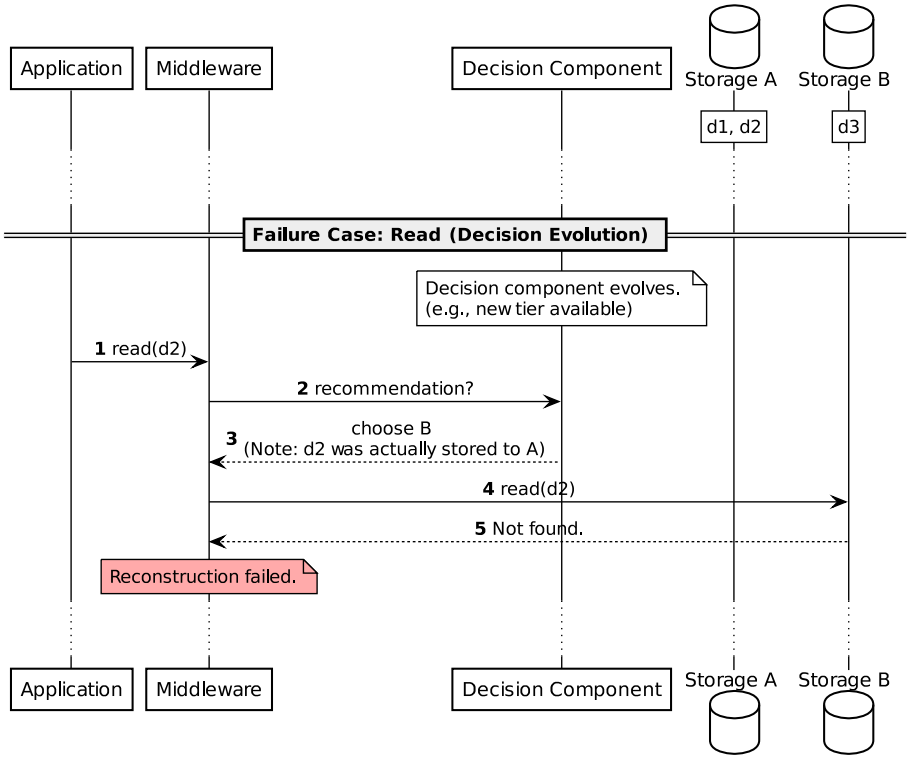


Figure 6.15: Illustration of the Decision-Evolution failure case on the read path in continuation of Figure 6.13. If reconstructing the data relies on recommendations not to change (see mark 2 of this figure) then reconstruction can fail if the decision component is adaptive. Similar to the Service-Unavailable failure case, a fallback should be established on the read path which might be less favorable from a performance perspective. Running into this failure might be used as a trigger to replicate or move data to match the evolved recommendation.

Allowable Read-Path Optimizations: This does not mean that performance improvements cannot be achieved also on the read path. But those optimizations should only be considered if a fallback is ensured, which relies on a persistency mechanism of the underlying storage system instead of an additional service. Examples for safe decision aids might provide cus-

tomized index structures or a reference to a transformed copy of data that is optimized for the access pattern, for example, using tracing information. In this case, if the recommendation issued fails due to an error or a timeout, it can be disabled or ignored for subsequent requests.

6.3.3 Patterns for Semantic Reasoning

The variety of features and dynamics observed in storage systems give rise to a number of additional rules of thumb on how and when to seek decision aids. To an extent, the implementation of the decision library can rely on these rules to guide a strategy to probe and apply different optimization opportunities. As storage optimizations are multivariate by nature, no such rule set can be expected to hold universally, even less so considering anticipated additions to the landscape of storage technologies.
Among these anticipated changes are storage systems that offer building blocks to address different access semantics. Traditionally, different tiers might have been optimized to handle different data sizes as well as distinguish between metadata and data. Various modern storage systems such as DAOS as well as data description frameworks such as HDF5 offer a variety of complex data structures beyond byte streams including n-dimensional fields, tree structures, relational databases, or key/value pairs. In the context of HPC, structured data is very common, with access semantics varying considerably across domains and subfields. A read or write operation usually will only correspond to a subset of a much larger data set, with the result that how data is chunked and spread out across a distributed storage system has a profound impact on performance. As a result, modern I/O middleware offers many different options to represent data internally and often also offers to keep different indexes. HDF5, for example, offers various features so users can explicitly fine-tune behavior:

- *Data segmentation*: Chunking and data distribution can profoundly impact performance. A particular challenge is to negotiate a balance between compute intensity and storage/memory bottlenecks.
- *Array of Structs vs. Struct of Arrays*: Whiles technically a transformation as well, an optimization in HDF5, for example, would typically rely on using either multiple variables or register a custom composite datatype.
- *Compression and other transformations*: A particular type of compression, in this case, may be a domain-specific compression algorithm, which might be learned using generative neural networks.

Most often a middleware or a filesystem will map the high-level data format onto a combination of internal data structures which yield a good trade-off for a variety of use cases. Because different application domains will favor different underlying representations, a common decision might be to pick which of these representations should be used when considering the overall workflow.
With these examples in mind, it is possible to construct context-sensitive heuristics, which are discussed in more detail in the following. It should be mentioned, however, that changing circumstances such as the specific site or the workflow at hand, might give rise to a different set of rules. This reinforces the need to seek heuristics with at least limited self-learning capabilities. It is, therefore, useful to consider the following three factors and their resulting consequences:

Local vs. Global vs. Remote vs. Central Knowledge holds implications about the communication required to access decision aids.

- *Local Knowledge*: Access to the decision aid does not require queries to components outside of the execution context. Obviously, local knowledge is limited and will often take not the whole workflow into account. Slowly changing or static knowledge, however, can be cached locally and be distributed at the job start requiring no further interaction with remote resources (see also Static vs. Dynamic Knowledge down below).
- *Global/Remote Knowledge*: Access to the decision aid requires to wait for a request which uses, for example, the network.[1]
- *Central Knowledge*: Remote knowledge might in some cases be aggregated in a central service. A central component can be problematic in so far that it might introduce a bottleneck.

[1] Remote knowledge can be associated with a range of different time penalties. A performance analysis that reveals how the network topology results in different latency domains is included as part of Section 7.4 (Component Assessment: Decision Aid Propagation Service) using a newly developed benchmark to evaluate the suitability of ZeroMQ to serve as a transport for the Decision Aid Propagation Service (DAPS).

Static vs. Dynamic Knowledge holds implication on the time frame a decision stays valid. This is relative to the systems and decision components that are analyzed. Somewhat simplified, static knowledge can be cached and remains valid and consequently introduces only a one-time latency penalty. Dynamic knowledge, on the other hand, will need to be constantly updated over a period of time when decisions might be made. Often this can result in a latency penalty, although a variety of situations allow fetching the information asynchronously incurring no relevant delay.

- *Static*: A decision aid that is invariant within the execution context. Different time scales apply. For example static for job allocation, information which can be propagated on job start maybe be considered static by subsequently spawned processes when there is no need to request an update for the decision aid. More formally, a decision component *DC* can be considered static when for two different timesteps $t_0 \neq t_n$ the decision component is concluding with the same recommendation:

$$\forall t_n : DC(request, t_0) = DC(request, t_n)$$

- *Dynamic*: A decision aid is dynamic if it reacts to changing circumstances such as the network contention within the execution context. A decision component can be considered dynamic when for two different timesteps $t_0 \neq t_n$ the recommendation changes:

$$\exists t_n : DC(request, t_0) \neq DC(request, t_n)$$

Context-Free vs. Context-Dependent Knowledge From a workflow perspective, context-free/context-dependent could be if the relationship of producers and consumers matters for the recommendation. Most workflow related decision aids are assumed to be context-dependent, such as that a decision aid is valid for a specific task but not another.

- *Context-Free*: A decision aid that is the same independently of, for example, the job allocation. Thus formally, a decision component *DC* would be considered context-free for a *request* in two different contexts $ctx_a \neq ctx_b$:

$$\forall ctx_b : DC(request, ctx_a) = DC(request, ctx_b)$$

- *Context-Dependent*: A decision aid that will yield different recommendations dependent on, for example, the nodelist. A decision component *DC* would be considered context-dependent if:

$$\exists ctx_b : DC(request, ctx_a) \neq DC(request, ctx_b)$$

6.3.4 Access Type and Transfer Size Effect on Recommendation

Another important, but highly site or system-specific factor, considers the relationship between access/transfer sizes and latency of I/O requests. While a user would typically issue a request of a certain access size, the discussion here is reduced to transfer sizes because most sophisticated I/O middleware and many operating systems will reorganize requests before they are dispatched. Coalescing, for example, is routinely performed for small continuous requests across different layers. Similarly, storage devices typically only offer block-wise access semantics, which often requires breaking up requests exceeding the block size. In summary, a request size issued in one layer does not necessarily correspond to the request size being used in an underlying layer of the storage stack during fulfillment. With the addition of high-bandwidth memory (HBM), wider buses and additional interconnect technologies this situation can be expected to further manifest. From a node perspective for many of the current systems Table 6.4 illustrates some recommendations when grouping them by access type and access size.
A determining factor to decide if a component should seek recommendations is the latency penalty which renders querying additional decision aids for small I/O unfeasible. The discussion here focuses on heuristics to decide which kinds of recommendations are appropriate in different scenarios. The resulting *thresholds*, and possibly additional bins, would differ considerably from system to system and even in between different layers.
This is another reason why it is important to develop self-learning approaches that adapt and fine-tune these and similar rule sets for easy consumption by decision components.

		Access/Transfer Size	
		small $<$ threshold	threshold $\leq$ large
Access Type	read	avoid dynamic global be conservative about static global	be conservative about dynamic global be conservative about static global consider additional indexes
	write	be conservative about dynamic global be conservative about static global self-describing transformations	consider dynamic global consider static global self-describing transformations

Table 6.4: Limiting candidates when considering decision aids by applying the rules from Section 6.3.3 (Patterns for Semantic Reasoning) for access type and transfer size.

The discussion also changes whether synchronous or asynchronous requests are being considered, where more expensive decision components can be acceptable in the asynchronous case. For all relevant factors, the argument about when to seek a recommendation boils has to be made by considering the amortized cost of obtaining and executing a recommendation:

$$\sum optimized(request)_{\text{cost}} + recommendation_{\text{cost}} \overset{?}{<} \sum request_{\text{cost}}$$

If a baseline is to be established, it will often be useful to use elapsed time as a cost metric which will be obtained for the time it takes to acquire a recommendation and execute the optimized request in comparison to the default execution. Cost metrics are not limited to time to solution though and could ultimately include total cost of ownership, power requirements or the effort for users to actually adopt the technology into their applications.

6.3.5 *Scaling Considerations*

As was discussed in Section 6.3.3 (Patterns for Semantic Reasoning) and Section 6.3.4 (Access Type and Transfer Size Effect on Recommendation) probing for a decision can introduce non-negligible overhead as well as bottlenecks if a decision component relies on information that need to be requested from remote resources. For this reason, this section discusses sources of latency as well as the mitigation of potential bottlenecks when considering central decision services.

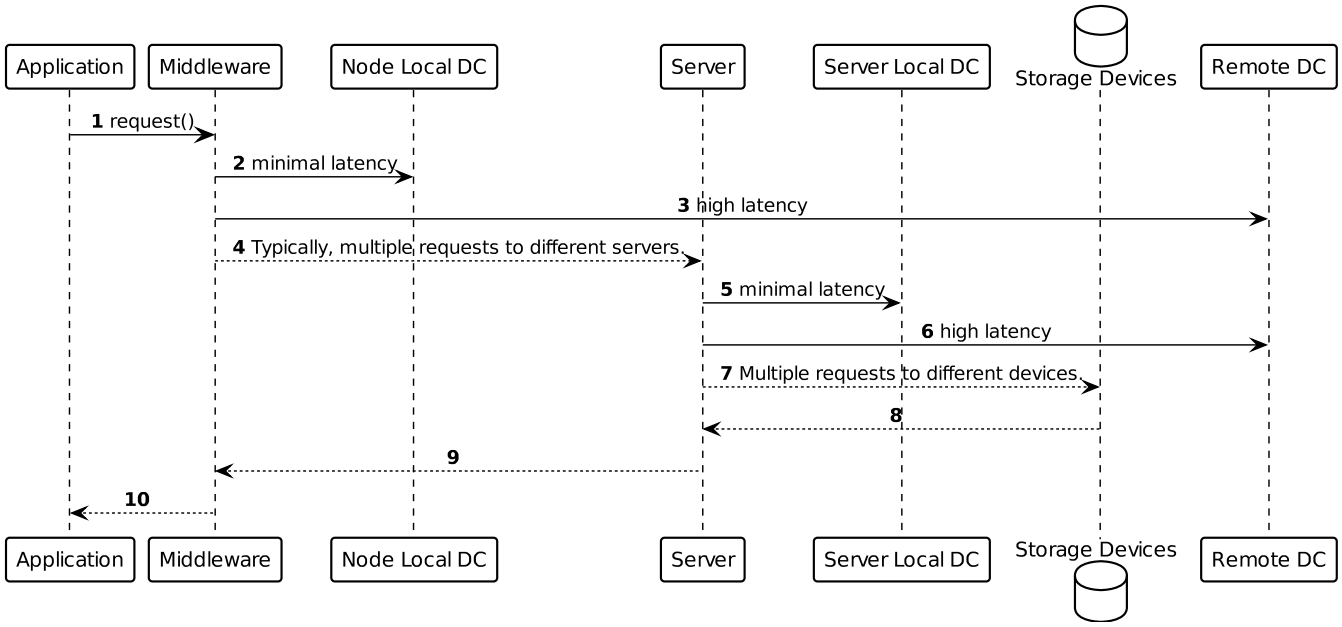


Figure 6.16: UML diagram indicating latencies induced by adding local and remote decision components to the storage pathway.

Causes of Latency: Latency can get introduced on multiple levels whenever a decision aid is called to give a recommendation. In the most optimistic case, knowledge is already present locally and no information from a remote decision aid needs to be requested. To minimize this case, either the decision library or the Decision Aid Propagation Services (DAPS) are expected to maintain caches in an effort to minimize remote queries.
Figure 6.16 indicates latency in a sequence diagram as a request traverses the storage pathway. A request is issued by an application and typically will be handled by various middleware. While additional latency might be imposed by this middleware, the example here disregards latencies independent of the introduction of obtaining a decision.
Information that can be obtained from their local decision service (2 or 5) will incur only minimal latencies, this applies to compute nodes as well as to server nodes. Potentially high latencies occur when a remote decision component is consulted (3 or 6).

Hierarchy of Central Decision Services and Caching: One challenge for decision support is the desire to take a holistic perspective onto the larger

system. To do so, information potentially coming from many sources needs to be collected, analyzed, and propagated again. The most informed decisions would be achieved by one central control service, but in practice, this is rarely feasible, nor would it be desirable because it introduces bottlenecks that potentially inhibit the scalability of the larger system.
Where possible, decision services should be as distributed as possible while avoiding central decision points. This can be achieved by adopting hierarchical approaches, where the provider of a decision aid is strictly limited to collect and aggregate the information necessary to yield the desired advisory. Similarly, decisions that would improve overall performance only slightly might not qualify to escalate a decision to the higher central decision services. Instead, lower-level decision services that are limited to a project, a workflow or only a job/task should be considered when delegating responsibility and sharing information. It is important, to emphasize, that slower-changing decision aids still might incorporate the bird's view perspective obtained on the highest level which is then passed and cached to local decision components. As this holds the implication that decision aids have a lifetime, there is a certain risk that this information is not the most recent, but in many cases when considering workflow knowledge the assumption is that once patterns emerge they will not change rapidly for a given system. Here the introduction of caches allows considering decision aids that take a holistic perspective if they tend to be static. Ultimately, this also helps to minimize latency, as it gives incentives to prefer local over remote decision aids.

6.3.6 Decision Support Library

The discussion so far motivated the need to establish a new kind of interface, a *Decision Support API*, and also established an abstract blueprint for decision components for I/O requests in Section 6.3.1 (Toward Abstract Decision Components). This section extends the discussion on this abstract interface and proposes a first concrete API as well as internal behavior to address the requirements imposed by HPC workflow environments. The discussion is broken up into the following parts:

- First the *Decision Support API* which allows applications and middleware developers to delegate decision-making is introduced in Section 6.3.7 (Decision Support Library Public API).

- Then core components and internal behavior such as callbacks and caches are covered in Section 6.3.8 (Decision Support Library Internals).

- Finally, functionality that is delegated to external libraries and tools is discussed in Section 6.3.8 (Exported Functionality to Services Outside of the Decision Support Library:).

6.3.7 Decision Support Library Public API

This section introduces the Decision API. The application and middleware facing API including data types that users of the decision support library can interface with are illustrated in Figure 6.17. Most notable are the *decision points* which have to be registered along with a *request context* and a number of possible *actions*. Optionally, it is also possible to report performance,

which the actual implementation of the decision component may use but is also free to disregard.

Decision Points: While decision points are registered with the decision library they are primarily an abstract declaration to allow easy discovery across complex software trees and exchanging different implementations of concrete decision components. A decision point at least requires the request context and a number of possible actions. Each decision point may be associated with a cache that retains recently calculated recommendations.

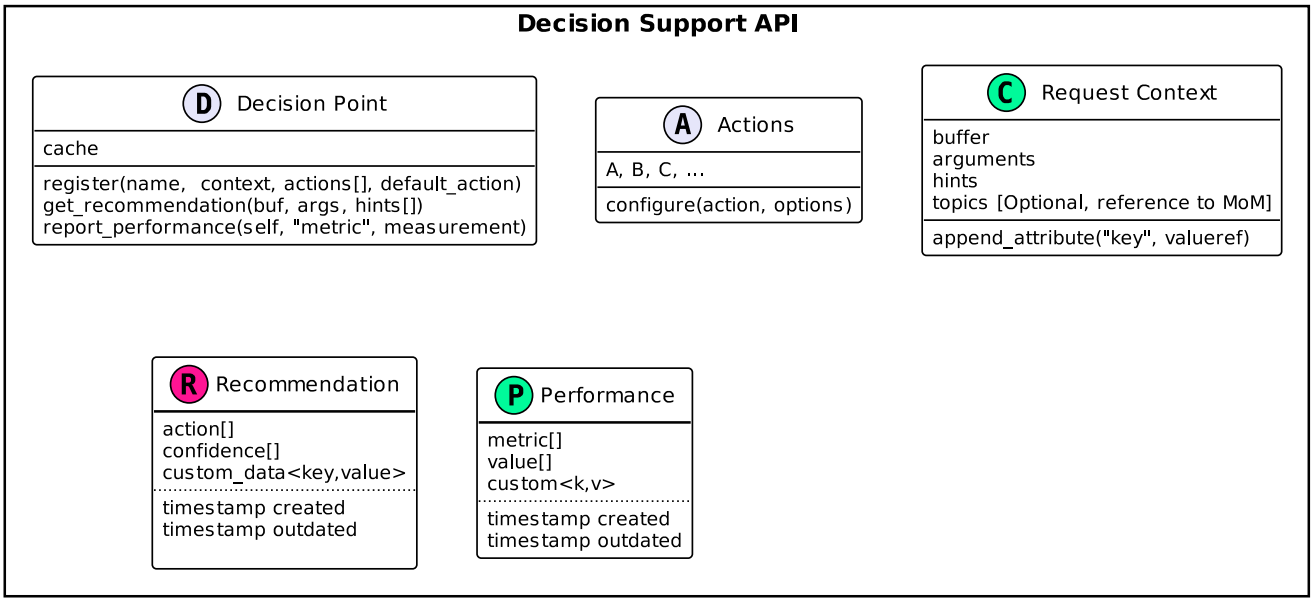


Figure 6.17: Abstract API for the Decision Support Library and topic integration with the Decision Aid Propagation Service (DAPS). The core concept is the decision point for which users provide possible actions as well as a description of the I/O request context in expectance of being able delegate the decision.

Request Context: The request context explicitly exposes relevant information about, for example, the environment in which the decision is to be made. As this might easily contain thousands of different factors users are encouraged to declare information they would consider themselves to make a similar decision. The request context should at least consists of a reference to the data buffer, the request type, and arguments. To improve the decision, hints, performance feedback, and topics holding decision aids can be provided which can be added to the request context.

Actions: Actions are declared, so a decision component can provide all suitable parameters required to populate a recommendation. They are especially helpful for machine learning-based decision components that attempt to map from the request context to the most appropriate action. Similar to a *decision point* a *request context* has to be declared.

Topics: Topics are only briefly introduced here, as they are covered in more detail in Section 6.5 (Decision Aid Propagation Service). Simply put, a topic in this context refers to an agreed resource identifier through which a decision aid can be accessed. Depending on the topic, this information is fetched and updated from a cache or from a remote resource. Topics are provided and handled by DAPS and not by the decision support library.

Performance Feedback: Feedback can be optionally provided by the application or middleware which is utilizing the decision support library. This

information can then be used to train and improve the actual decision components. Performance feedback may also be published to DAPS, but mainly to consolidate how decision aids are accessed and propagated.

Recommendation: The Decision Support API returns a recommendation when an application or a middleware calls `get_recommendation()`. Usually, this will be a list of recommendations along with a confidence value. Optionally, additional metadata may be provided by a decision which applications/middleware can use to refine the decision. A recommendation is also associated with timestamps of their calculation and a due date to help cache eviction for short-lived recommendations.

Example: To demonstrate how an application or middleware might interface with the decision support library when registering a decision point Listing 6.1 provides an example in pseudocode. As an alternative to using a library interface, pragmas could also be an option, especially if decisions are used which do not require runtime information.
In any case, first a *request context* is defined along with a number of helpful topics. The actual decision component might not necessarily require these topics but this offers a mechanism for developers of applications and middleware to suggest relevant topics that offer valuable cues for developers of decision components. In a second step the available actions are defined, which are then passed together with the request context to `register_decision_point()`.

```
# Declare/register a decision point
actions = [A, B]
topics = ["/workflow/<id>/total_bytes", "/node/cpu/utilization"]    # Optional
ctx_def = (buf, args, hints, topics)
dp_ref = register_decision_point("name", ctx_def, actions)
```

Listing 6.1: Pseudocode of the registration of a decision point with the decision support library.

To get a recommendation, applications and middleware would then issue a call to `get_recommendation()`. Listing 6.2 illustrates in pseudocode how the calling application or library might handle the resulting recommendation.

```
# Populate request context
ctx = (buf, args, hints, topics)
rec = get_recommendation(dp_ref, ctx)

# Time and execute action
start_perf()    # (Optional, see last three lines)
switch(rec[0])  # assume rec[0] is the highest ranking recommendation
{
        case B:
        doB()
        break

        case A:
        default:
        doA()
        break
}

# (Optional: Report performance to improve decision over time.)
perf = stop_perf()
report_performance(dp_ref, "metric_name", perf)
```

Listing 6.2: Pseudocode of populating the request context, requesting a recommendation and utilizing it, and afterwards optional reporting of performance feedback.

6.3.8 Decision Support Library Internals

This section discusses the internals of the decision library. The decision library's core mediates between consumers and producers of decisions aids.

To balance efficiency and convenience for users and operators the following three features are of particular importance:

- Effortless integration of information and decision aids that need to be obtained via the network.
- Caching of recommendations which are unlikely to change during the execution context but which are requested multiple times.
- Exchangeability of trained decision components based on context, for example, the current workflow.

Figure 6.18 illustrates the interactions from declaration of a decision point down to the invocation of a decision aid as well as the retrieval of outside information the Decision Aid Propagation Service (DAPS).

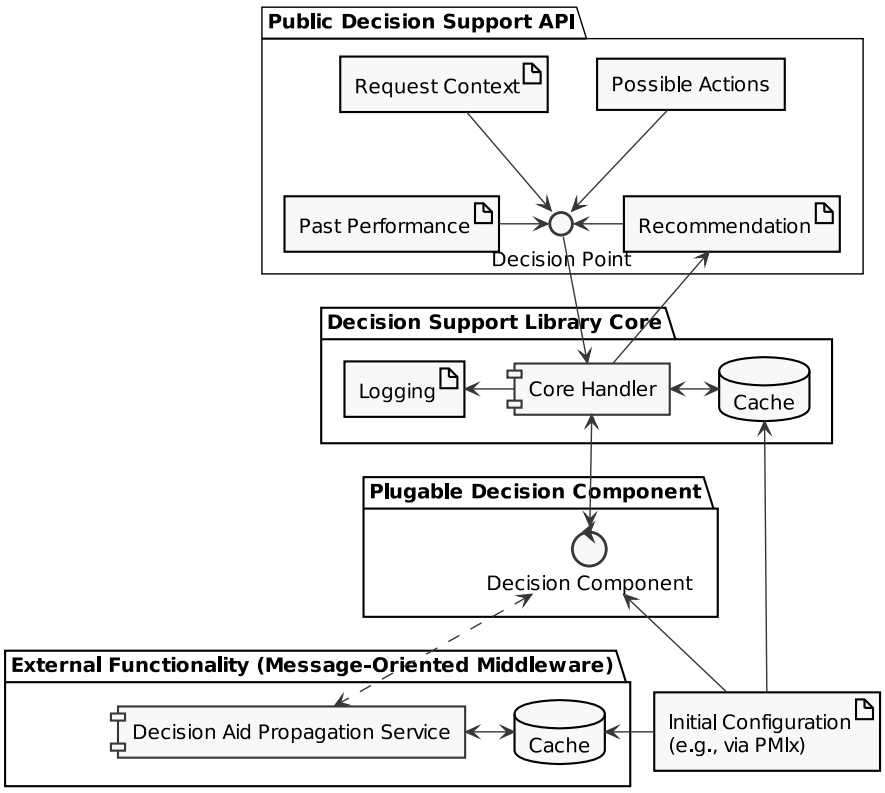


Figure 6.18: Use Case Diagram describing how a recommendation is generated and reused utilizing multiple cache layers, for already calculated decision as well as for already fetched slowly updating decision aids.

Decision Aid Invocation: The main responsibility of the decision support library is to invoke the right decision component based on the context and return with a suitable recommendation. In general, two different modes for the decision component invocation are anticipated:

- Invocation through topic only, which is most adequate for decisions that are the result of many aggregated information. From the application/middleware perspective this would typically mean consuming an externally computed recommendation.
- Invocation via callback/plugin, which is assumed to typically calculate a decision as if it were part of the application/middleware. It should be noted that part of the application might still be physically spread out due to the distributed nature and multitude of participating computational agents in HPC applications and services. A plugin/callback might also simply wrap fetching a recommendation from a DAPS topic.

Which of the two modes is more appropriate depends on the use case and also how much effort can be spent by application or middleware developers to fine-tune. By providing these two modes, it is possible to preserve flexibility and convenience while allowing to invest in sophisticated optimization of the decision process if necessary.
Besides software engineering considerations, this design is also supported by the two types of decisions in relation to an execution context: static and dynamic decisions. For a static decision aid, recommendations can be precomputed, and they would not change for the workflow within the given execution context. Instead of gathering and calculating the decision each time the Decision Aid Propagation Service (DAPS) might be used to request the presumably already determined/calculated decision. One common deployment for static decision components is thus expected to get an initial payload with precomputed recommendations or decision aids which can be accessed through a recommendation cache which is discussed later in more detail. A possible mechanism to propagate this information to the individual nodes in a job allocation on job start is, for example, PMI(x).
For dynamic decisions that can be made by different actors, the decision component could even be compiled directly into the application for minimal overhead. However, in many cases, this is not desirable nor feasible as software trees already confront operators and vendors with combinatorial challenges. This has the downside of removing flexibility as the application or middleware might need to be recompiled if the decision aid changes.
There is also a hybrid of these two modes, where DAPS is used to fetch different decision aids or parameters and learned weights from a decision aid based on the workflow. This way the decision aid can react to local circumstances while including a workflow perspective that is accommodated in, for example, the learned weights. This becomes useful, for example, when I/O middleware has to decide for a storage tier. A general-purpose classifier that reliably provides correct tier recommendations across different applications and workflows is nearly impossible to train or program. But within a specific workflow, the ambiguity of different factors is typically dramatically reduced to the point that machine learning algorithms provide very reasonable performance.[2] A decision component, in this case, would be a plugin that is propagated at job start, which wraps the computations necessary to give a recommendation. The nature of this plugin is not further constrained, it might contain a simple heuristic, a set of policies, or a neural network. While generally to be used with care, implementations of decision components are free to communicate over the network, for example, to pull in live information about a subsystem's health. As such, decision points could also be repurposed as mechanisms to add very rudimentary instrumentation or to trigger a change of an actual instrumentation's behavior.
By parsing for or logging the active decision points during compilation or execution it is later possible to present an overview of available decision points throughout the storage stack, as far as privileges permit it.

[2] That this is the case is also demonstrated in the assessment of developing different workflow-specific decision aids which use machine learning techniques to recognize I/O behavior or cluster filetypes based on I/O access patterns in Section 7.3 (Component Assessments: Decision Aids).

Recommendation Cache: An optional recommendation cache is introduced to reduce recommendation return latency by avoiding network communication or recomputation of decision aids that are only slowly changing. This is useful because some decisions, once found, are expected to remain valid for a certain time, which creates opportunities to minimize overhead without negatively impacting user experience.

At the center of these optimizations is the recommendation cache, which is populated on initialization and updated as recommendations are requested. This way, static recommendations as well as the necessary decision components can be efficiently obtained via PMIx at job start. Figure 6.19 illustrates the role of caches for the decision support library and DAPS in relation to the applications and I/O middleware. An application issues a call to an I/O middleware, which delegates a decision to the decision support library, here referred to as LibDC. LibDC will first attempt to find a cached recommendation. If the decision is not valid anymore the responsible decision component is consulted which might recompute the recommendation or use DAPS to fetch a remote decision aid to reach a recommendation. DAPS just like LibDC features a built-in cache that instead of final recommendations caches topics and updates transparently depending on the topic invalidation times. A decision component might feature internal caches as well, but because a decision component may be an arbitrarily complex program this is omitted in Figure 6.19.

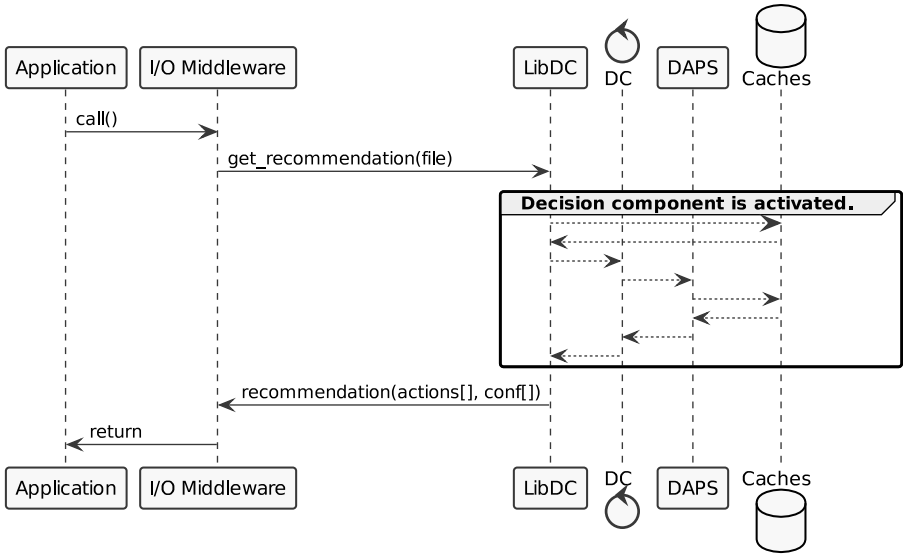


Figure 6.19: Requesting a recommendation is only performed if the decision component is activated. The decision support library will then try to use a cached recommendation or compute a new one. The interactions within the activated decision component block are only exemplary and are highly dependent on the decision component in action.

In general, decision components are encouraged to be conservative about the resources required to obtain a decision. Yet, for many workflow related decisions, even an expensive decision can be worthwhile when the benefit is amplified if enough tasks in a workflow benefit from it. In addition, it seems inadequate to expect decision components to be always fully optimized, especially, if a decision is expected to become static for a given workflow. In other cases, a qualified decision might require considering a large amount of data to ultimately reach a recommendation. This is the second reason why the decision support library itself is equipped with a built-in cache that preserves computed recommendations until their due date. For applications and middleware that request many decisions, various workflow-dependent strategies for cache eviction become appropriate, possibly defaulting to least recently used (LRU).

Exported Functionality to Services Outside of the Decision Support Library: Not all functionally exposed by the decision support library should be implemented as part of the library itself. In particular, this applies to the specific decision components and to the acquisition of decision aids which is best realized as an independent service because the decision support library is only one of multiple plausible clients. This independent service is called

DAPS, which is discussed in Section 6.5 (Decision Aid Propagation Service). Decision components/aids should be exported and implemented as plugins because they often will be site, application, and workflow specific. All of the earlier discussed stakeholders might see fit to maintain their own optimized decision components. Vendors might provide default decision components adjusted to match the features of their products. Site operators would adapt and fine-tune decision components to adapt to the specifics of their site and to enforce different policies. Application and middleware developers can adapt and switch decision components based on common workflows, while scientists can modify or train decision components to match their specific workflows. This fine-tuning might be exposed through user-friendly dashboards which might be provided by, for example, the site operating. This might also help data labeling efforts.

6.4 Knowledge Representation & Decision Aids

"Standards are always out of date. That's what makes them standards." – Alan Bennett

Before describing the design of the Decision Aid Propagation Service (DAPS) introduced next in Section 6.5 (Decision Aid Propagation Service), it is useful to consider the different kinds of information that are necessary for workflow-aware decision-making in storage systems. In this section, it will be motivated why seeking *blueprints* or even a *standard* for different decision aids increases interoperability across workflows even though workflows themselves may be relatively specialized. Establishing somewhat reliable knowledge representations from a storage perspective in the context of scientific workflows is useful for at least the following reasons:

- Similar systems expose similar information. This often requires glue code to reshape data. For common data types, there should be abstractions, for special cases diverging from standards always remains in option.
- Aggregation becomes easier across multiple levels when using established performance counters. Many different metrics share similarities about how they can be reduced, thus saving users the need to define these operations.
- Learning tools, just like aggregation tools, are easier to reuse when knowledge representation is somewhat stable. Otherwise, a well-working trained network would need to be retrained. This should be avoided if not absolutely necessary.
- During propagation, a well-defined interface (not necessarily a standard for the carried data) allows the transport to optimize network traffic and latency. For example, in a publisher/consumer setting, knowledge propagation can be planned to minimize sending messages containing the same information.

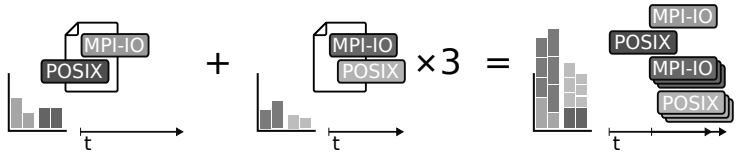
6.4.1 Contrast to Established Monitoring/Telemetry Representations

For monitoring data, there are a number of existing efforts to homogenize common monitoring data for aggregation and analysis [OpenTelemetry, 2020]. The case for workflow decision aids, however, is different as information sometimes flow bi-directional or has to be updated as new information is reduced into the decision aid, which at least from a conceptual level has to be reflected in the data representations.

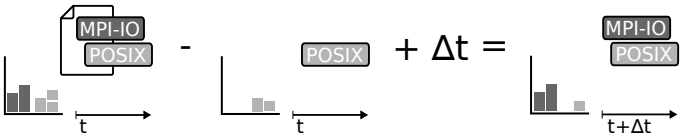
The usefulness of stable representations and aggregators as an important abstraction to ease integration with other tools has also been identified in other projects [Ganglia Developers, 2019; Nagios Enterprise, 2019; Open Grid Computing Inc., 2019]. The OVIS project, for example, explicitly names abstractions for analysis to shed light on the system dynamics as a core goal of the project and therefor invested into the development of the Light Distributed Metric Service (LDMS) [Agelastos et al., 2014].
More recently, approaches originating in cloud-like deployments [Prometheus Authors, 2019b] increasingly enjoy popularity also in HPC environments. Prometheus, for example, also integrates easily with dashboard tools like Grafana [Grafana Labs, 2019]. Similarly, collectd [collectd, 2020] is a daemon that introduces abstractions for telemetry collection and to an extent aggregation which is widely used in distributed deployments. Finally, the diversity of different approaches motivates consolidation efforts such OpenTracing and OpenTelemetry [OpenTracing, 2018; OpenTelemetry, 2020].
Consequently, this section seeks not to come up with better representations for monitoring data, but instead adds considerations to the discussion for a new type of data product derived in part from telemetry and monitoring data which will be referred to as *decision aids*. While decision aids may remain simple in many cases, optimizing an I/O workflow decision can require the composition of many workflow artifacts that may require sophisticated operations to associate, subtract, and project activity.

Algebra for Performance Modeling and Workload Extrapolation: A recurring requirement for I/O optimization in the context of scientific workflows requires the combination of performance artifacts from different applications and processes. This is necessary to analyze and attribute I/O pressure imposed onto various systems on the one hand but is further useful to compile perform extrapolations for usage in scheduling and resource allocation scenarios.
Figure 6.20 a) illustrates the merging of different I/O statistics from different, for example, Darshan logs, to create a combined statistic. Imagine, for example, needing to determine the I/O cost of scaling up a workflow. This may be achieved by measuring individual tasks and then multiplying each respective task by their frequency of execution.

Figure 6.20: Examples for the definition of two particular useful operations for performance understanding, modeling and extrapolation of scientific workflows.



(a) Combine I/O activity from multiple log records, for example, obtained from multiple workflow task, to assess total I/O cost for a workflow.



(b) Apply substractions to remove unwanted activity and rebase in time by applying a time delta, for example, for use in I/O-aware scheduling.

Similarly, one can use recorded activity to create performance extrapolations which are useful to inform scheduling decisions to become I/O-aware. Figure 6.20 b) shows an example where activity recorded for a task or a

number of tasks is modified by removing some activity and applying a time delta Δt to rebase when the activity takes. This report algebra for performance analysis, modeling, and extrapolation is implemented as a prototype for Darshan log records in Python using the ability to overload operations. Internally, these operations are redirected to methods, thus allowing a wide range of operations to be performed on top of report objects.

6.4.2 *Representing Decision Aids*

This section discusses important factors to consider when representing decision aids. In particular, the section motivates why decision aids should be *timestamped*, vaguely *standardized*, augmented by *confidence information*, and as far as affordable *explicitly-defined* while allowing to *fallback on schemaless formats* if performance is not critical.

Monitoring data is often aggregated because it is created at frequencies which make reporting raw events unpractical. Unlike monitoring and telemetry data, however, many decision aids for workflows and storage systems may not need to be updated at high frequencies. Although some decision products informing about, for example, network contention would typically be only useful if obtained recently. As a result decision aids should include *creation* as well as *outdated* timestamps.

While decision aids and monitoring data share some commonalities, there is also a number of differences. Monitoring data will often be based on well-established metrics, for decision aids, these do not exist yet, and in other contexts, they are specific to a particular task, workflow, or site, thus requiring a more flexible representation. In particular, decision aids and recommendations generated by a decision component need to communicate uncertainty or confidence. Often a decision aid might produce multiple candidates for a decision. Thus, to allow the consuming decision component to make a final verdict, the information returned by a decision aid in many cases will be an array of actions. Both, recommended actions and confidence are therefore depicted as arrays, denoted by square brackets in Figures 6.21 and 6.22.

Requirements and priorities vary from site to site and over time also within a site. It is not untypical to see different workloads during working days in comparison to weekends. For sites servicing multiple different scientific domains, workloads can vary considerably from domain to domain. Even within a particular scientific domain, a number of different workloads are observed with different methods or processing phases showing certain characteristic behavior. In Section 6.3 (Actions & Decision Points) a generalization of decision components was discussed because it allows to easily swap out decision components that have been optimized for one use case or another. The same reasoning extends to decision aids used by a decision component.

This requires, besides defining interfaces to query for a recommendation, to define which information should be considered to reach a decision. As decision aids in this form do not exist today, there is little prior experience in the context of HPC and scientific workflows to draw from when it comes to standardizing such representations. However, to optimize representations, it is necessary to understand the most common modes and information that are being consumed from a decision aid.

In combination, the spectrum of different workflows, the differences across

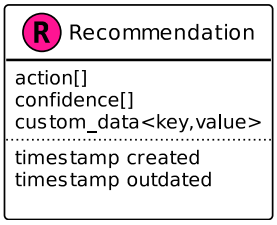


Figure 6.21: Recommendation datatype as used by the decision support library.

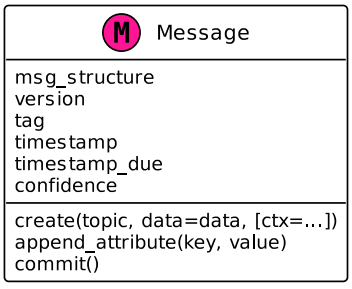


Figure 6.22: Message datatype as used by the Decision Aid Propagation Service (DAPS).

sites, and the changing nature of supercomputers require a decision support framework for workflow I/O to allow compromising between conflicting requirements. Other research on intelligent adaptive systems balances, for example, the need for flexibility and performance by providing an abstract communication layer that allows users to provide explicit data descriptions that offer the supporting software and hardware infrastructure the necessary information to break apart data into independent streams.
There are also additional incentives to explicitly define representations as they allow transparent optimization by the workflow I/O decision support framework in the following cases:

- When an initial payload is prepacked and shipped on, for example, job start, additional decision aids can be afforded when assigned only a limited budget.
- When a local cache is used to remember the decision history, it is less likely to get evicted if they are timestamped. These caches might not necessarily be at a node-local level but could be within the rack/switch, I/O node, etc.
- When requesting information from remote places, less strain is put on the network if representations are efficient.

Many of these requirements can be achieved transparently to consumers and producers of decision aids by introducing a special middleware optimized for the propagation of decision aids.

6.5 Decision Aid Propagation Service

Section 6.3 (Actions & Decision Points) and Section 6.4 (Knowledge Representation & Decision Aids) discussed how a decision aid is computed and in which way information required to reach it might be represented from a workflow I/O perspective. This section adds a discussion on how to bring different information to interested decision components. The challenge here is to remain efficient while not compromising convenience for application and middleware developers as well as site operators.
To propagate relevant monitoring data and decision support products a message-oriented middleware, the Decision Aid Propagation Service (DAPS), is proposed. Figure 6.23 illustrates how the service is expected to integrate with existing applications, middleware while not making strong assumptions about the underlying transport. Similar approaches are also used in some multi-agent distributed systems in the context of intelligent adaptive systems such as the Robot Operating System (ROS) [Quigley et al., 2009]. The boundary conditions and semantics for an HPC site, however, differ considerably from those in a robot, granting freedoms not present in the other and vice versa. At its core the middleware proposed here provides a publish and subscription service for *decision aids* or raw telemetry offering the following functionality:

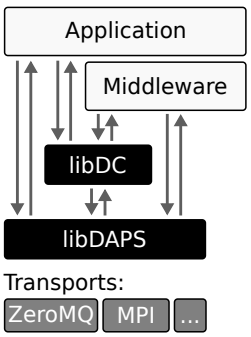


Figure 6.23: Relationship between applications and middleware to libDC and libDAPS. No strict assumption about the underlying transport is made as new network standards and technologies are constantly emerging.

- A *publish/subscribe service* for developers and operators to quickly integrate and recombine decision products.
- A *registry for contexts/topics* which allow well-defined messages in addition to schema-less messages to allow for efficiency but also for aggregation of decision products by the middleware.

A variety of projects such as ZeroMQ or RabbitMQ provide basic building blocks to balance different trade-offs when designing such middleware. [Stanford-Clark and Nipper, 1999] developed Message Queue Telemetry Transport (MQTT) for low-bandwidth, high-latency asynchronous environments. Yet, a number of HPC and workflow related challenges cannot be optimally addressed using these libraries and frameworks. For example, HPC applications typically already establish a transport layer between nodes using MPI. ZeroMQ and RabbitMQ both rely on a separate transport, which might waste precious resources. Mercury [Soumagne et al., 2013] for example, implements remote procedure calls (RPCs) on top of various transports commonly found in HPC. In addition, a number of unconventional features are required which extend what is typically not found in already existing message-oriented middleware:

- Caches for slow-changing topics and support for decision aid lifetime.
- Uncertainty information for decision aids that offer predictions.
- Aggregators for topics which are combined from multiple sources.
- HPC topology-aware broadcasting for minimized network communication and efficient topic or decision aid aggregation.

The efficient implementation of this middleware is beyond the scope of this thesis, especially as such an API remains an active area of research that is best developed iteratively in a dialog with the community of stakeholders. The remainder of the discussion will, therefore, assume that a proof of concept implementation wraps established transports and offers an API tailored for use by the decision support library, introduced earlier in Section 6.3 (Actions & Decision Points), I/O middleware, and applications. As such DAPS is considered part of the control layer introduced in Section 6.1.3 (Stakeholders). This section focuses on storage and workflow relevant example topics, resulting in semantics and key requirements of DAPS.

6.5.1 Example Topics

Topics in the context of the Decision Aid Propagation Service (DAPS) are best explained by considering a number of examples relevant for storage and workflow-aware decision-making. Listing 6.3 introduces a number of example topics as well as a number of relevant attributes such as the lifetime or the frequency at which the topic is updated. In addition, it is indicated if a topic has an aggregator implemented. From top to bottom, the listing illustrates complex I/O advisories such as offering a runtime prediction or I/O phases, down to job statistics or node-local telemetry.

```
1  Topic Name                                                  Aggr.   Freq.  Lifetime
2  ----------                                                  -----   -----  --------
3  /workflow/<workflow-id>/task/<task-id>/runtime_prediction   false   n/a    inf
4  /workflow/<workflow-id>/task/<task-id>/io_phases            false   n/a    inf
5
6  /workflow/<workflow-id>/task/<task-id>/bytes_total          true    n/a    inf
7  /workflow/<workflow-id>/task/<task-id>/operations           true    n/a    inf
8  /workflow/<workflow-id>/task/<task-id>/throughput           false   n/a    inf
9
10 /job/..                                                     true    n/a    inf
11 /job/<job-id>/runtime_prediction                            false   n/a    inf
12 /job/<job-id>/io_phases                                     false   n/a    inf
13
14 /node/cpu/utilization                                       true    1 Hz   1 sec
15 /node/cpu/cores                                             true    1 Hz   1 sec
16 /node/network/utilization                                   true    1 Hz   1 sec
```

Listing 6.3: Topics and contexts are arranged in a hierarchical namespaces which is especially useful for the workflow related topics to query information based on workflow- or task-id. While intended for workflow knowledge here, the abstraction appears to be useful for non workflow decision products as well.

A topic `/workflow/<workflow-id>/task/<task-id>/runtime_prediction` would then return the most recent runtime estimate in seconds. A scheduler or an anomaly detector can use this information to improve resource utilization or to notify users. A runtime prediction could be generated by considering prior executions or a mathematical model. Similarly, the same applies for `/workflow/<workflow-id>/task/<task-id>/io_phases`. The I/O phase might be derived from profiling or tracing information as they can be obtained with, for example, Darshan. Here it is assumed, that many such complex advisories cannot be aggregated, while other information such as the bytes written or the number of operations can be.
This approach could also be used to expose local information, such as the `/node/*` namespace. As no expensive communication is necessary, this example indicates a high refresh frequency and a low lifetime. Additional node-local topics can be imagined, it should be noted, however, that it may be more appropriate to obtain this information through other existing telemetry providers.
A decision aid is not constrained to be number or textual input, it could also be a program, a mathematical function that describes some model, or the weights and parameters for an entire neural network.

Topic Versioning and Tagging: Topics should be versioned and potentially tagged, as different consumers might rely on a topic but would like to experiment with improvements. While no effort is made to consolidate and reserve identifiers in the namespace, a combination of random token and version number should be sufficient to identify conflicts and notify users or operators. Versioning is further useful to encourage the evolution of decision aids, while also encouraging to explicitly define their form which is helpful with optimizing the propagation to consumers.
As HPC sites differ considerably, only an abstract communication interface is specified. Different sites might vary the underlying implementation of a topic provider, as is common also for many existing services. Which information is exposed can also vary depending on available privileges and available technologies. Access control will not be discussed in detail, but it should be noted that the namespaced approach in combination with message tagging allows realizing both role-based access control and message-level access granularity.

6.5.2 *Decision Aid Propagation Service API*

This section introduces an API proposal for the Decision Aid Propagation Service (DAPS). Figure 6.24 offers a UML class diagram describing the message and topic datatypes including methods to interact with them. The basic cornerstones of the middleware are *topics* to which *messages* adhering to a certain format can be published which are then forwarded to subscribers:

Topics: Identifiers for decision aids or other publishable/subscribable contexts. For a given topic it is generally assumed that only a few publishers will populate a specific topic, while many different nodes (for example, all nodes of a job) might subscribe to a topic. Topics should feature a version identifier, to allow upgrading the API and message types. Topics also allow indicating update frequencies.

Messages: Bits of information that are published to a topic. While message payloads can be schemaless, it is recommended to specify a schema for better performance. Messages carry a timestamp to evaluate recentness, as well as a payload which depends on the topic. Typically, these messages would be of a well-defined form as discussed in Section 6.4 (Knowledge Representation & Decision Aids). This way a decision component can disregard information it deems outdated, and fall back to its default behavior.

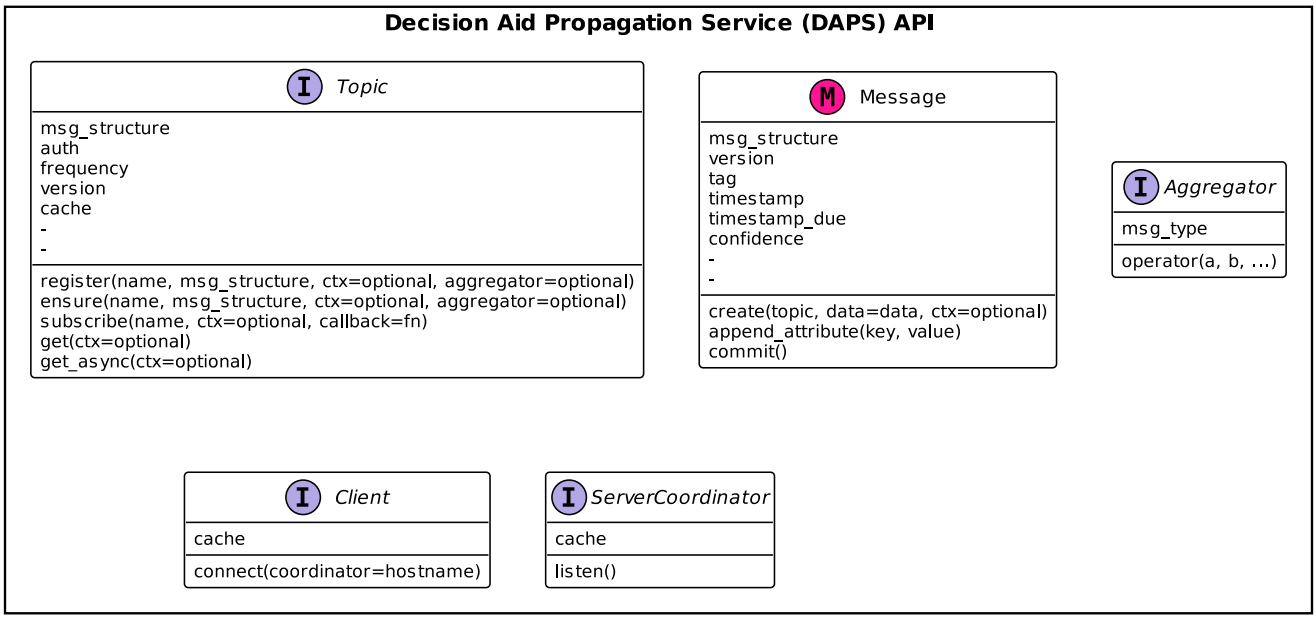


Figure 6.24: API of the Decision Aid Propagation Service (DAPS) to efficiently and conveniently propagate decision aids throughout the storage stack. Information is encoded into messages which are distributed through topics to topic subscribers. Typically, a consuming library might register as a client to a single coordinator, but large allocations may be divided into hierarchical partitions. Well-defined aggregations allow reducing information before transmission if clients specify which information they require in detail.

Coordinators and Gateways: In an effort to avoid requiring site-wide central services, DAPS clients are joining propagation networks by specifying a gateway/coordinator. Different propagation networks might be connected by subscribing to one network and republishing to a local group. As such a client might connect to multiple networks. Site-wide services to an extent might be subjected to quality of service (QoS), for example, featuring rate limits at which subscribers can expect to be notified or at which publishers are allowed to push updates.

Aggregators: The Decision Aid Propagation Service (DAPS) should offer support for aggregations. In part this was already motivated Section 6.4 (Knowledge Representation & Decision Aids) where well-defined topic aggregation helps to handle complex data and metrics more easily in different contexts. In addition, there are performance considerations that are discussed in Section 6.5.4 (Decision Aid Propagation Service (DAPS) Core Functionality) which help to reduce traffic induced by publishers and subscribers to DAPS on the network.

6.5.3 *Semantics and Example*

This section discusses the behavior of the Decision Aid Propagation Service (DAPS) in more detail and also illustrates an example. One of the motivations for this service is to abstract away the complexity of the transport and potential aggregations away from producers and consumers. Figure 6.25 illustrates the relationship of publishers and subscribers on top an arbitrary transport. A publisher, connected as a DAPS client will announce a topic by registering it with DAPS.

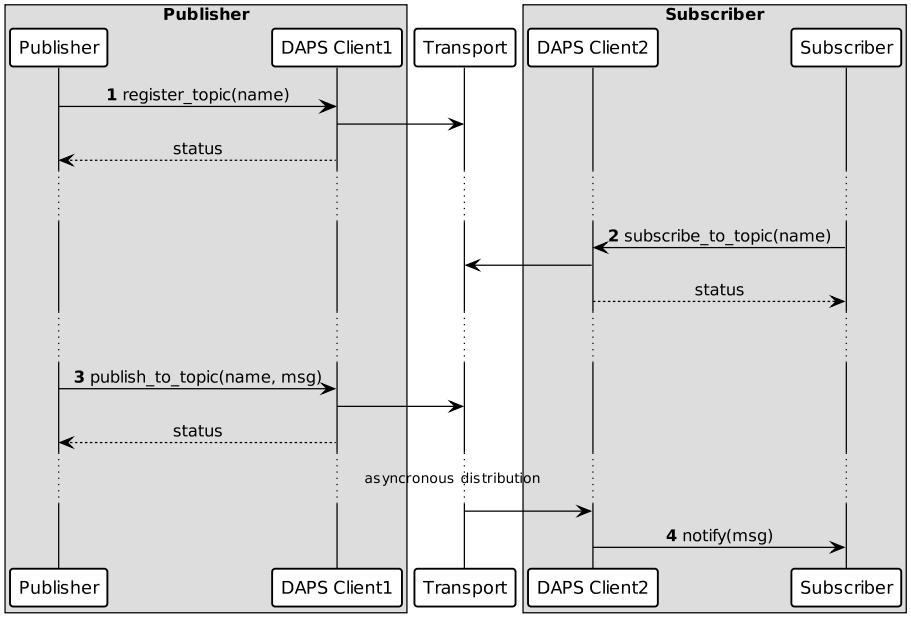


Figure 6.25: Subscriber/Publisher interaction on top of an abstract transport. Different implementation for message-oriented middleware follow different strategies with implications for latency and convenience.

In many cases, topics are expected to be served only by a single publisher, for example, a post-mortem analyzer of workflow statistics might infrequently update a decision aid associated with a workflow, a task of a workflow, or a data product of a task which might be held in DAPS's caches for a while. Listing 6.4 gives an example in pseudocode of such a publisher.

Listing 6.4: Pseudocode for a producer of a stable topic. The publisher first connects to a DAPS coordinator and registers as a publisher to a topic. In a separate step some anaylsis might yield decision aids which are then published to this topic including contextual constraints.

```
1  daps = daps_connect(coordinating_hostname);
2
3  topic = daps.ensure_topic(name='/workflow/<workflow_id>/TopicA', msg_type=MsgTypeA,
       authentication)
4
5
6  for workflow_id in registered_workflows:
7  result = perform_postmortem_analysis(workflow_id)
8
9  msg = new Msg(type=MsgTypeA, {valueA: result[0], valueB: result[1]});
10 msg.append_attribute(custom_propA, result[3])
11
12 daps.topic_publish(topic, ctx=workflow_id, msg)
```

A publisher of a dynamic decision aid on the other hand might be activated with the start of a job or task, where it starts publishing the latest advisory such as, for example, the network contention or the fill level of the file system. As far as publishers are concerned the APIs to publish a topic remain identical independent of the decision aid being static or dynamic. Subscribers on the contrary are expected to choose among multiple mechanisms to obtain the contents of a topic. In particular, synchronous and asynchronous invocations are offered, as well as mechanisms to register callbacks for event-driven subscribers as illustrated in Listing 6.5.

Listing 6.5: Pseudocode demonstrating multiple ways to recevie messages from a topic.

```
1  daps = daps_connect(coordinating_hostname);
2
3  ctx = {}
4  ctx['workflow_id'] = environment.get('WORKFLOW_ID')
5
6  topic = daps.subscribe(name='/workflow/<workflow_id>/TopicA', ctx)
7
8  // sync
9  msg = topic.get(topic)
10
11 // async
12 request = topic.get_async(topic)
13 if request.status() == 'finished':
14 msg = request.get()
```

```
15 // callback
16 def fn(msg):
17 print(msg)
18
19 topic.register_callback(fn)
```

As such a number of different specimens for clients of DAPS are anticipated: Providers and consumers might be pure, in that they are either only publishing or only consuming decision aids. In later examples, consuming producers will be discussed, which will take a decision aid and use it to generate an aggregation or a value-added secondary or tertiary decision aid. Similarly, common network functionality such as DAPS clients to act as forwarders, gateways or coordinators.

6.5.4 Decision Aid Propagation Service (DAPS) Core Functionality

Depending on the mode of a decision aid, and especially if it is static or dynamic, different strategies to propagate information are appropriate. As already discussed along with the decision support library proposal in Section 6.3.8 (Decision Support Library Internals) to make a decision while introducing only a minimum of overhead, caching and prefetching decision aids is important. This section, therefore, discusses how existing building blocks for message-oriented middleware might be leveraged and how distinguishing *static* and *dynamic* decision aids encourage using different delivery strategies.

Base on Top of Established Network Middleware: There are a variety of existing solutions for message-oriented middleware developed with different use cases in mind. The existing middleware solutions, however, do not address the special requirements necessary to propagate and aggregate decision aids as outlined in this section. Decision aids, often being complex products of different workflow artifacts and as a result potentially workflow-specific require reconfigurable aggregations. In addition, most message-oriented middleware is developed with cloud-like services in mind. The special topological peculiarities of HPC systems are usually not addressed. RabbitMQ, for example, aims to be convenient to use and comes packed with features and tunables for a variety of use-cases, but relies on central exchange services for distribution. For especially latency-sensitive use cases, ZeroMQ offers the low-level building blocks but distribution strategies have to be implemented on a case to case basis.
Most established communication solutions in HPC like the Message Passing Interface (MPI) or Process-Management Interface (PMI) address a different use case too. Efforts like Mercury offer remote-procedure calls (RPC) on top of a variety of technologies, but typical producer/consumer or publish/subscribe service primitives are not included. Although, PMI offers a fan out feature that would notify only a single task per node, which is delegated the responsibility to notify the remaining tasks on the same node.

Caching: Initially, caching on current stacks would be kept on a per-node level for frequently requested topics. In general, a lot of workflow advisories are anticipated to be updated sparsely as soon as enough observations for a stable recommendation have been collected.

Propagation of a Static Decision Aid: Minimal interference and overhead with the actual application can be achieved when only using static decision

aids. Figure 6.26 illustrates a sequence diagram outlining a number of steps which help to limit overhead for static decision aids. Upon submission of a new job, the scheduler can start gathering information relevant for at least the following use cases:

- Information relevant to improve the scheduling decision such as the I/O phases or a runtime prediction.
- Information relevant to improve application and library behavior at runtime such as past communication patterns.
- Information that impacts the behavior or performance of subsequent tasks and consumers such as a recommendation that influences the shape of output data.

As the job typically will remain waiting in a queue, it is possible to compile an initial payload that contains the relevant decision aids. Ideally, these advisories are then shipped together with the job start avoiding queries to a central service have to issue when a decision has to be made. In fact with PMIx, a technology which also is tightly integrated with some scheduling systems, this can substantially reduce load on the decision services at runtime. For decision aids that are assumed to change only slowly or not at all relative to job/task lifespan, this allows to effectively serve otherwise remote decision aids as local ones.

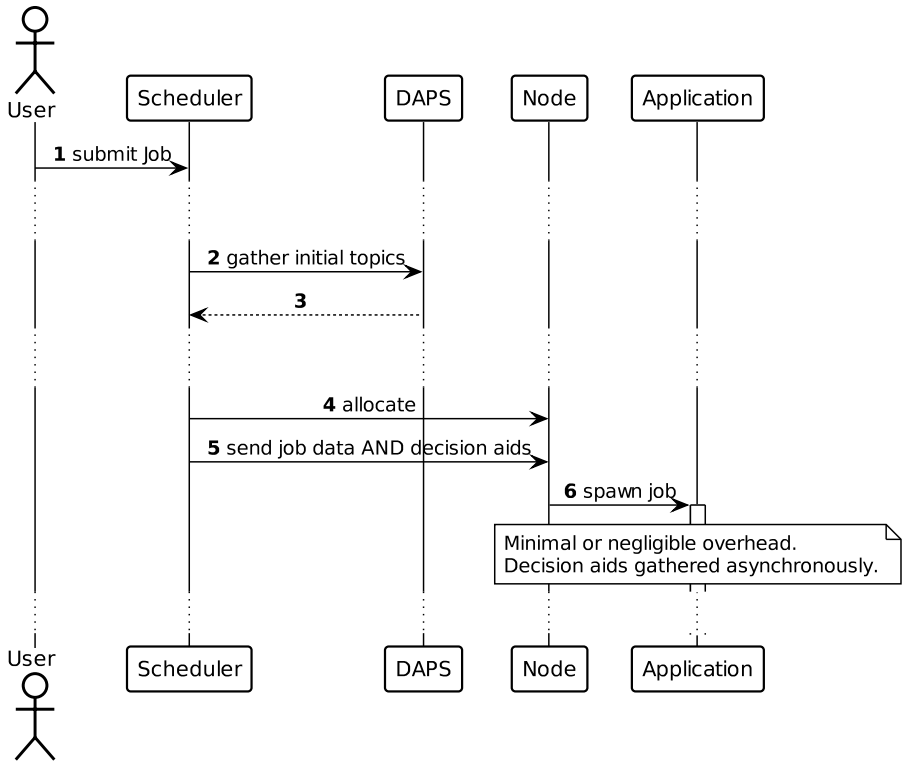


Figure 6.26: Obtaining static decision aids using the Decision Aid Propagation Service (DAPS) with minimal runtime impact.

Propagation of a Dynamic Decision Aid: For dynamic decision aids, it still makes sense to pack an initial payload which instantiates and pre-populates caches to a degree but as the decision aids are expected to change during the allocation, additional requests will be issued to update with more recent information. Figure 6.27 illustrated the process for dynamic decision aids in a sequence diagram. Again after job submission, relevant topics and information are gathered and packed to be transmitted via PMIx on job start. As soon as the application is spawned, dynamic decision aids can

start regularly updating their topics. Typically, this is expected to be performed asynchronously to avoid latency introduced by the decision process. If a library or an application deems it is necessary to require the most up-to-date information they can still utilize the blocking calls, although this is generally discouraged.

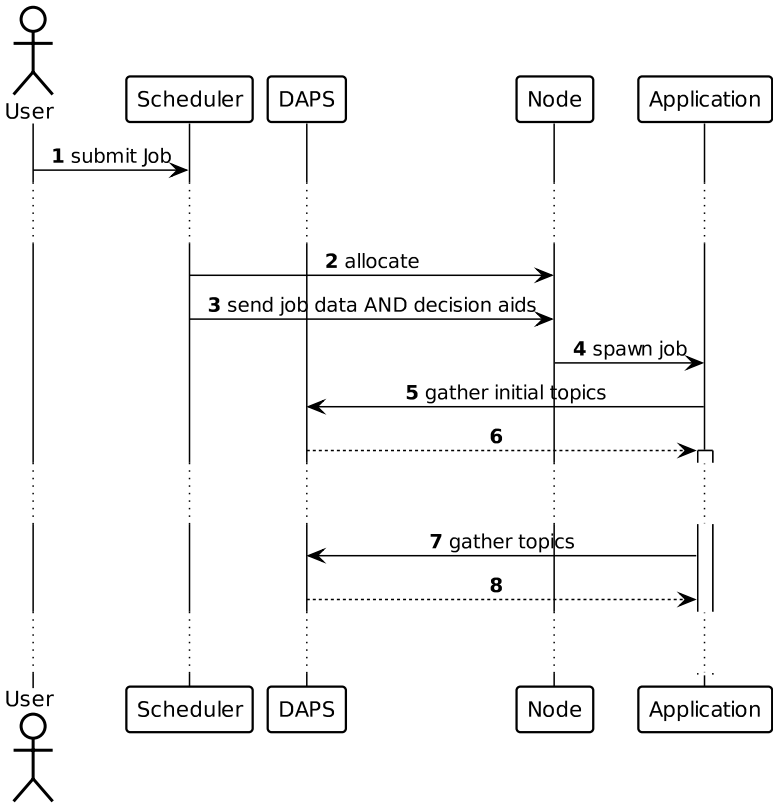


Figure 6.27: Obtaining dynamic decision aids using the Decision Aid Propagation Service (DAPS).

6.5.5 *DAPS Scalability Considerations in HPC Environments*

This subsection is concerned with scalability considerations specific to a message-oriented middleware such as the Decision Aid Propagation Service (DAPS) in an HPC environment. The discussion focuses on two key objectives in particular:

- Minimize decision aid traffic to minimize interference with application-, network-, and storage I/O.
- Minimize message counts and aggregation overhead to reduce waiting time spent by decision aid consumers.

For many storage-related workflow use cases it can be anticipated that interference with application I/O will not become an issue because recommendations are not updated at high frequencies. For more dynamic workflows reacting to performance variation and from an operating perspective, however, these optimizations are important. Four design considerations for efficient utilization of communication infrastructure in HPC data centers are especially impactful and thus influenced the interface introduced earlier:

- Use a publish/subscribe interface explicitly to allow other optimizers to take advantage of, for example, topology information or caches.
- Use multicast messages to avoid unnecessary one-to-one communication.

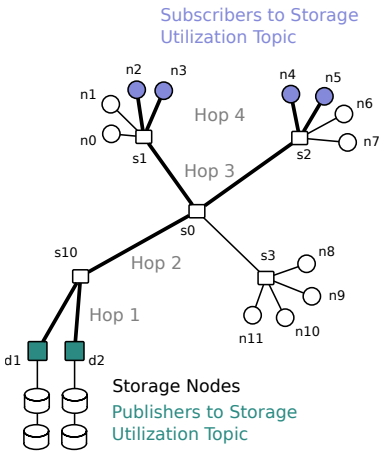


Figure 6.28: Example on how to avoid messages, by being topology-aware and capable of multicasting to avoid unnecessary connections and messages.

- Aggregate/reduce decision aids or telemetry along the path or in transit.
- Utilize existing/already open connection, for example, within an MPI application use MPI as a transport.

To give a simple example, consider the use case of a storage-related decision aid illustrated in Figure 6.28 taking the physical perspective. Here two storage nodes (dark green) publish a topic that might indicate fill levels or a prediction about anticipated busyness in the future. Highlighted in light green are potential consumers who announced their interest in receiving updates as the publishers publish recent data. This use case allows for two optimizations that take advantage of the topology and message routing that is able to perform limited interpretation/aggregation of decision aids. Instead of establishing many point-to-point communications multicast relaying allows to significantly reduce traffic. This is shown in Table 6.5, whereby counting messages it is easy to see, that especially for high-frequency data naive approaches would induce a large amount of avoidable traffic despite relatively low node counts.

Table 6.5: Comparison of message counts and hop counts for two different propagation strategies using a point-to-point propagation schema and an aggregated multi-task propagation schema for a simple example to illustrate the need for topology awareness of the Decision Aid Propagation Service (DAPS).

	Message Counts	
Scenario	Worst Case (single-trip)	Optimal (aggregators)
Hop 1	$4+4$	$1+1$
Hop 2	8	1
Hop 3	$4+4$	$1+1$
Hop 4	$2+2+2+2$	$1+1+1+1$
Total	**32**	**9**

6.6 *Learning & Decision Components*

"If people never did silly things, nothing intelligent would ever get done." – Ludwig Wittgenstein

To take full advantage of the proposed architecture the integration of self-learning approaches is an important aspect. In fact, semi-autonomous agents are included in the stakeholder diagram introduced earlier as a special kind of non-human stakeholder, as once deployed interference with these agents needs to be taken into consideration. This section discusses how the proposed architecture was designed to cater toward their integration but also where fundamental challenges remain. Key drivers to consider learned decision components and decision aids include the following:

- the fast-evolving nature of compute and storage systems;
- the increasing complexity and diversity which varies from site to site;
- the diversity of workflows and applications which results in different sites offering different services depending on their user base;
- the lack of experts to manually tune or develop tailor-made solutions.

With one goal being to mimic parts of the optimizations process that are currently being made manually, the expectation is that some of these optimizations can be found and applied automatically using the capabilities of machine learning algorithms. This does not mean that traditional optimization and established heuristics are expected to cease in importance, but in a variety of cases, general-purpose solutions are mainly deployed because

current systems are not able to adaptively swap to more appropriate heuristics. With some modern existing but more importantly prospected systems, the combination of hardware acceleration and algorithmic breakthroughs in the last decade allows to now considers previously undeployable complex machine learning models. It is important, however, to consider which current learning techniques are well enough understood to be useful in HPC storage systems without sacrificing reliability, performance, and data consistency characteristics.
As discussed in Section 2.6 (Control and Machine Learning Techniques) new algorithms and more efficient hardware allow to consider using deep neural networks as *universal function approximaters* [Hornik et al., 1989; Leshno et al., 1993] useful to generate mappings and discovering structure when presented with training examples. Thus when defining the abstract decision point interface, a core requirement was to incorporate straightforward compatibility to these kinds of machine learning methods. A challenge here is that machine learning as a field has been rapidly evolving recently with some of the new and exciting capabilities and algorithms not always resembling well-understood and established problem statements. Nonetheless, already now it can be anticipated that novel approaches will constantly require integration. Consequently, the discussion in this section focuses on balancing the integration of well-working existing approaches while attempting to future-proof interfaces for compatibility to upcoming capabilities. As this is an emerging area of research with blurry boundaries, this section focuses on the requirements necessary to accommodate and develop what might be referred to as *adaptive or neural heuristics* in compute and storage stacks. The section breaks down into the following aspects:

- Section 6.6.1 (Integration) starts with a high-level integration overview. It is focused on avoiding to restrict the interfaces between decision points and machine learning to today's landscape of machine learning approaches, but already assumes the use of system simulation to speed up training and fine-tuning these learned decision components.
- Before it is possible to use these methods, it is necessary to formulate HPC storage-related decisions as problem statements aligned to existing machine learning methods. For this, the notion of *adaptive or neural heuristics* for storage systems and workflows is introduced in Section 6.6.2 (Adaptive and Neural Heuristics for Storage Systems and Workflows).
- There are also various challenges, limitations, and problems to consider with state of the art machine learning methods when deployed in support of scientific workflows and storage systems. Section 6.6.3 (Challenges for Adaptive- and Neural Heuristics) discusses challenges related to storage systems and workflows are identified and discussed with potential mitigations to limit where these approaches have to be ruled out entirely. Many of the already plausible deployment models will primarily make use of supervised learning methods but research prototypes sometimes demonstrate remarkable capabilities of self-exploring and unsupervised approaches. This section also considers where end-to-end approaches as well as approaches that integrate reinforcement learning and genetic algorithms provide value also to storage system optimization from a workflow perspective. These use cases are a core motivation for seeking integration with simulation software even though existing surrogate system models sometimes are lacking or still cost-prohibitive.

6.6.1 *Integration*

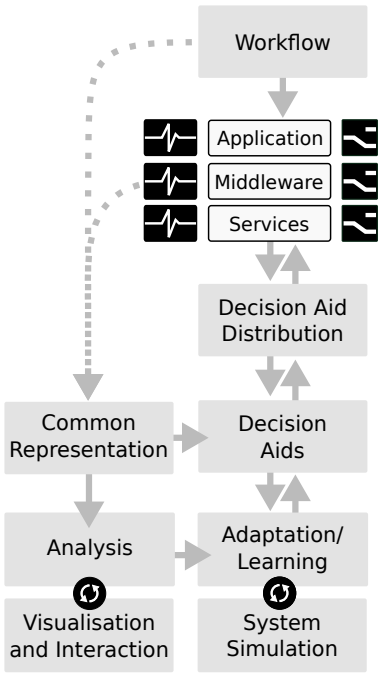


Figure 6.29: Illustration how different high-level tasks require tooling to allow for end-to-end approaches to learn and tune decision aids and utilize them to optimize workflows. A more detailed variant of this figure in relation to the tooling ecosystem is Figure 6.30 on page 147.

This subsection provides an overview of how the previously introduced building blocks (Decision Points, Decision Aids, DAPS, etc.) are combined to realize a continuous optimization cycle. The coarse architecture is adequate for a variety of different control and decision-making scenarios and is inspired by established methods from control theory such as PID controllers, as introduced in more detail in Section 2.6.1 (Control Theory and Feedback Loops). Unlike a PID controller, however, the approach here extends beyond the assumption of applying corrective action against a set value while limiting oversteering but instead aims to extract value from the feedback loop to fine-tune or train I/O and workflow specific feature detectors and decision components. In addition, for many workflow and storage-related optimizations, different deployment models can be adequate depending on a balance between whether information is acquired online/offline, and whether interventions are applied in- or out-of-band all while considering the locality of data transformations and decision computation.

Revisiting Figure 6.11 (here included again as Figure 6.29) the feedback loop starts with a workflow definition, for which telemetry can be collected, and for which decision points have been determined, for example, using the *Decision Support API*. As workflow artifacts accumulate they are converted into compatible representations to allow their combination. These can be directly made available as decision aids, or be further analyzed to derive *secondary* or *tertiary decision aids*. Often this analysis will be guided manually by a human in the loop, for example, to pick from candidates for information and learning approaches. For methods that require many training examples, or where interference in a production environment is not desired, the learning methods may be pre-trained with system simulation or with experimental system testbeds. As such a system requires many components that share common concepts, it was necessary to prototype components across the storage and workflow automation stack. This more detailed perspective is illustrated in Figure 6.30 in which black components indicate components for which sometimes multiple prototypes with different objectives have been developed to inform this research. The white boxes are used to indicate different technologies from existing software ecosystems.

From a functional perspective, Figure 6.30 illustrates the process of finding, tuning, and propagating (learned) decision aids in support of scientific workflows in relation to the wide range of tools that become necessary to realize continuous feedback.

The process begins with the execution of a *workflow* ❶. This will spawn individual applications, which often use middleware and other existing services and systems. The execution creates various artifacts which reveal details and performance characteristics. To collect these artifacts the Darshan-Workflow-Collector was developed. Information from various sources are converted into a *common representation* ❷ to allow easier reuse in *analysis, visualisation* ❸ and *adaptation/learning* ❹ scenarios. The results of analysis and training are used to create *decision aids* ❺ which will usually inform a heuristic or decision component somewhere in the software stack. To streamline the *distribution of decision aids* ❻ and to allow feedback loops, the Decision API and the Decision Aid Propagation Service (DAPS) are provided.

This perspective also illustrates how five, currently independently developed, core families of software need to be accounted for to benefit from research activities, existing software ecosystems, and user communities which are actively working toward consolidation and advancement within their respective fields, namely:

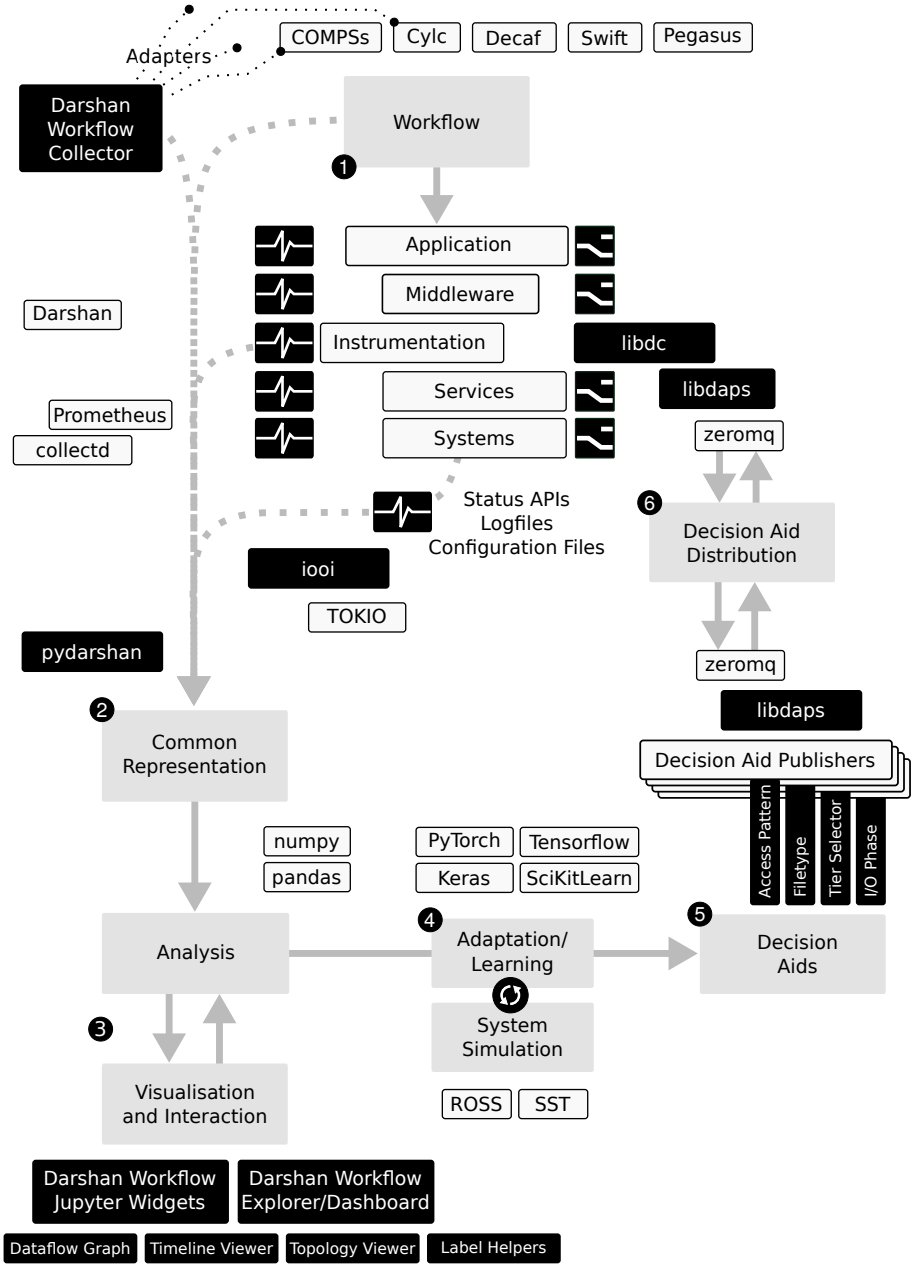


Figure 6.30: Illustration of the relationship between various tools and services to the workflow (grey boxes) to find, tune, and propagate decision aids. Components marked in black denote parts which required exploration in prototypes to inform the architecture and developing the system integration workflow depicted.

Workflow Engines: Workflow engines in the context of learning, are mostly implementing functionality to execute and distribute tasks across available resources. Research in this context considers machine learning, for example, for runtime predictions and anomaly detection to inform scheduling decisions and early detection of rogue tasks. As surveyed in Chapter 3 (Scientific Workflows in High Performance Computing) many different communities have developed their own workflow engines, although approaches to consolidate workflow definitions are gaining momentum. In the feedback loop for continuous workflow-aware decision aid generation, the current diversity is addressed by allowing adapters for different workflow engines.

Middleware and Services: Middleware and services are typically designed with stability in mind and thus, in the context of storage systems, often favor deterministic and predictable behavior to allow for data recovery but also to improve performance by avoiding metadata requests. None the less, machine learning such as found in recommender systems can be useful. for example, to populate caches before an actual request is made. Coupling such a system with patterns learned from workflow knowledge allows improving prediction quality. While software libraries, middleware, and services provide great utility by bundling common functionality, Chapter 5 (Information Sources and Decision Points) reveals that information sources and decision points are typically not expressed in a machine-friendly form and thus at the moment pass on opportunities presented by workflow-specific optimizations.
In the feedback loop for continuous workflow-aware decision aid generation, the challenge to enumerate and discover decision points and actions is addressed by introducing the Decision Support API as well as the contextual tokens which can be populated to ease association.

Telemetry and Monitoring: While workflow engines, middleware, and services integrate with various telemetry and monitoring services many obtained metrics are either not directly related, or if they are, a token for artifact association is missing. In some cases, artifact association can be achieved through correlation in time, in other contexts tokens between different services share similarities, but data is not clean enough for bitwise matchings. Machine learning can help to filter down to likely candidates or associate telemetry with the workflow task-data dependency graph.
Not limited to workflow and storage applications, telemetry data is routinely combined with machine learning methods for *classification* and pattern recognition tasks, as well as in conjunction with *regression* to predict, correct, or fill in missing values.
While monitoring and telemetry capture systems often feature modular designs, they are currently often deployed in privileged operational modes which do not allow to use of advanced machine learning-driven aggregators on a per-application or workflow basis. Instead of introducing yet another telemetry or monitoring system, the feedback loop for continuous workflow-aware decision aid generation relies on established instrumentation and collection tools such as Darshan, collectd, or Prometheus.

Analysis and Learning: A key consideration when developing learning methods for storage systems and workflow systems is to reduce friction from data transformations when relying on existing analysis and machine learning frameworks and tools. Here the willingness by developers and increasingly also corporations to open source their frameworks has resulted in a self-perpetuating improvement to the software ecosystem surrounding the data science and machine learning communities. While the previous three software families evolved over the course of decades coupled and constrained by the requirements of production systems, analysis frameworks and machine learning frameworks, in particular, matured relatively independent in large parts in corporate contexts.
Many users of machine learning methods will base their analysis and training on frameworks such as PyTorch, Tensorflow, Keras, or scikit-learn, with large parts of these tools developed in Python Often new approaches will be made accessible by researchers and educators in the form of toy examples

packaged for exploration in Jupyter notebooks. While such experimental setups often would not be suitable for deployment, they can considerably reduce the time from method discovery to use in production. Some machine learning frameworks allow to export to more efficient programming languages, accelerated binaries, or to prune networks after training to fit into constrained power/performance/latency envelopes [ONXX, 2019]. Little consolidation in workflow engines, middleware, and telemetry has taken place to be compatible with such exported machine learning models.
In the feedback loop for continuous workflow-aware decision aid generation, this is addressed prototypically by the development of PyDarshan to collect and transform efficient binary representation as used by multi-level instrumentation such as Darshan into Numpy arrays or Pandas dataframes. The output of many system utilities is wrapped by custom helpers performing similar transformations. As the products and calculations that have to be performed to generate decision aids that use learning methods sometimes can be performed out-of-band but other times need to be performed in-band, a new service, the Decision Aid Propagation Service (DAPS), is introduced. The spectrum of decision aids thus might span from small messages carrying only computed results in out-of-band scenarios to entire sets of hyperparameters and weights for a neural network in in-band scenarios.

System Simulation: Other domains such as robotics or game AI systems have used complex simulation to dramatically speed up training already in the past [Jaderberg et al., 2018b; Silver et al., 2018]. Many hardware and software changes are too costly to develop and manufacture just for testing purposes. Small-scale and high-level interactions and emerging behavior can be explored using system simulation. System simulation can also be used to generate artificial training data. Most HPC system simulation frameworks and most learning frameworks originally were not developed with compatibility to the other in mind. A more in-depth analysis of how to integrate system simulation such as discrete event simulation (DES) is discussed later in Section 6.7.2 (Discrete Event Simulation of HPC Systems).

6.6.2 *Adaptive and Neural Heuristics for Storage Systems and Workflows*

As adaptive or neural heuristics are not established concepts in storage systems, this section starts with a high-level characterization, before formulating combined problem statements compatible to common machine learning methodologies but concretized to accommodate information sources and decision points in storage and workflow systems.
A useful starting point for a formal definition to borrow from was phrased by Tom Mitchell in 1997, effectively characterizing machine learning as programs that favorably evolve as they gain experience.[3] This particular notion of experience on the one hand is friendly towards computers as it does not require reasoning about or understanding of the underlying problem. On the other hand approaches for *multi-task learning* (MTL) are an active area of research that only recently are more frequently combined also with deep learning. Yet, especially for workflow-aware decision-making in storage systems, a similar form of programs or heuristics that learn with experience and generalize to similar tasks are being sought.
For the following discussion, it is, therefore, useful to use *adaptive or neural heuristics* as a term to communicate that these are deliberately deployed as a compromise which has limitations but still provides utility, especially when

[3] A computer program **A** is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measure by **P**, improves with experience **E**. – Tom Mitchell, 1997

Put informally: *Algorithms* that *improve* on some *task* with *experience*. – Kilian Weinberger, 2018

approaching system optimization by minimizing amortized costs:

- *Adaptive* as in reacting to context which takes into account the *workflow* as well as the compute, network, and storage *resources and their topology*. The workflow exposes access patterns, temporal relationships, and access frequency. The system topology exposes bottlenecks, while data management seeks strategies to optimize handling and placement of data.
- *Neural*, carrying multiple implications, but in particular offering: 1) trainable by example and not explicitly programmed, thus minimizing manual tuning while maximizing accessibility to different stakeholders 2) convergent against an objective that can be specific for sites, systems or workflows. Not all decision points should adopt neural networks as the method of choice.
- *Heuristic* as an approximate strategy but thus very compact and applicable if constrained by latency or limited available resources for decision computation. This is appropriate under the assumption that for many problems it will not be feasible to seek solutions that truly find an optimum but instead nudge decisions in a favorable direction. Especially with UQ workflows, even small gains can get amplified.

In contrast to existing machine learning approaches employed in storage and workflow contexts, algorithms that build up experience over time to optimize workflow storage I/O are not very proliferated in current storage systems, as discussed in Chapter 4 (Related Work). While marketing material occasionally likes to convey systems as smart or intelligent, storage systems rarely offer features that would justify the attribute when related to current research in machine learning and artificial intelligence (AI).
Besides a general lack of well-understood algorithms, there is also a variety of engineering challenges, such as non-determinism and overhead, which complicates considering wide-spread adoption of machine learning methods in storage stacks. These concerns are discussed separately in more detail in Section 6.6.3 (Challenges for Adaptive- and Neural Heuristics). This section will instead explore where existing machine learning algorithms can provide value. Often the bounderies for a particular method are not clearly define as is illustrated in Figure 6.31. Finally, other domains applying machine learning benefited from well-defined tasks and standardized training bodies to compare algorithms against, this section will also discuss where these tasks and datasets might be found for workflow I/O and storage-related optimizations.

Being universal function approximators, many neural approaches are intersecting with broad application across machine learning tasks.

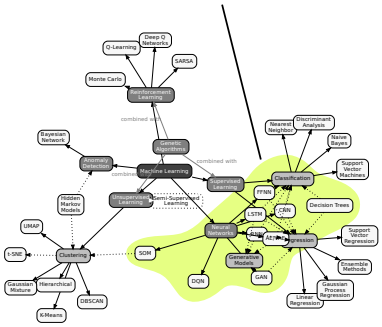


Figure 6.31: Relationships and overview of various methods in machine learning which illustrates that many of the following outlined task statements can be achieved utilizing one of many different ML architectures with different trade-offs. A larger readable variant of this figure is attached in the appendix, see Figure A.1 on page 244.

Task Formulations and Problem Statements: Earlier, machine learning was formulated as algorithms that can improve with respect to a Task T. With this definition in mind, this paragraph will concretize tasks for which machine learning approaches are available and where matching tasks are found in workflow and storage stacks. Using [Goodfellow et al., 2016] as a basis for machine learning tasks which in the following will be considered as relatively established and well understood, applications for storage and workflow optimization can be found for:

Classification tasks take an input and assign a category, for example, using a single function: $f : \mathbb{R}^n \mapsto \{1, ..., k\}$. When dealing with missing values instead of learning 2^n individual classifiers, it can be beneficial to phrase the task as learning a joint probability distribution thus learning only a single

function with a few structural constraints. Both scenarios occur also for decisions needed for storage and workflow optimizations. Examples include the classification of access patterns or predicting a file type.

Regression tasks take an input and map it to another value $\mathbb{R}^n \mapsto \mathbb{R}$ or a set of values $\mathbb{R}^n \mapsto \mathbb{R}^a$. Regression and prediction methods based on, for example, recurrent neural networks (RNN) or Long-Short Term Memory (LSTM) when employed for time series data, can be useful to predict task or request latencies, throughput, system loads, or compression rates.

Transcription tasks, for example, $\mathbb{R}^n \mapsto$ `[A-Za-z ]+` take an unstructured input and map it to a sequence of, for example, characters. This has widespread applications in optical character recognition but also offers useful applications in storage and workflow optimization. In particular, it allows turning complex measurements such as I/O traces into symbolic representations, which are often more efficient to handle and transmit.

Machine translation tasks are closely related to the previous task and are often implemented as Sequence-to-Sequence [Sutskever et al., 2014] mappers such as $(x_1, ..., x_T) \mapsto (y_1, ..., y_T)$. In natural language processing or robotics, this finds application in translating between languages or in turning a sequence of instructions into actions. For system and workflow optimization similar translation processes are necessary as telemetry is collected from different sources that use different formats. Similarly, trace replay could benefit from approaches that map an I/O trace back to a working code sequence of instructions needed to read/write from different I/O interfaces. If these approaches are useful substitutes to translate already machine-friendly sequences that typically carry not a lot of ambiguity is unclear.

Structured output tasks $\mathbb{R}^n \mapsto \{graph, segmentation, caption, ...\}$ share similarities to transcription but allow for arbitrary relations. Mapped is an unstructured input to a structured output such as a graph, segmentation/annotation or a caption (a valid sentence). This task might be particularly useful to reconstruct workflow graphs, or to annotate parts of a trace or a workflow as critical regions, for example, featuring an I/O phase.[4]

[4] Section 7.3.2 (Decision Aid: Complex I/O Phase Characterization and Segmentation) demonstrates how this can be used to find and characterize access patterns in I/O traces (in this case using DXT traces).

Synthesis or sampling tasks $\mathbb{R}^n \mapsto \{\mathbb{R}^a, \mathbb{R}^{a \times b}, graph, ...\}$ are used to generate complex outputs, such as synthetic inputs. This is well established for landscape generation, for example, for games or for generative art. From a storage perspective, this is sometimes indirectly considered for use in domain-specific lossy compressors which learn complex constraints inherent to the data which should be stored. Attractive architectures for this are VAEs and GANs which have been demonstrated to learn what could be interpreted as general concepts. [Radford et al., 2016], for example, showcase a network that was trained to classify men, men with glasses, and women. Rewiring the network for image synthesis allowed to compute $man\_with\_glasses - man + woman$ resulting in an image showing a *woman with glasses*. For storage and workflows optimization this can also be useful for workload generators, but also to expand symbolic representations back into, for example, telemetry inputs. In addition, synthesis can be used to introspect learned representations.

Anomaly detection tasks $\mathbb{R}^n \mapsto \{normal, anomalous\}$ can be used to detect outliers such as fraudulent transactions or faulty sensors. [Chandola et al., 2009] notes in a survey that anomaly detectors have a history of being adapted to a particular application domain. This notion of specialized

anomaly detection also applies to scientific workflows: By taking a workflow perspective, anomaly detection can become more nuanced by alerting only about task executions that are anomalous with respect to the workflow instead of with respect to all task executions of a system.

Denoising and *Imputation* tasks $\mathbb{R}^n \mapsto \mathbb{R}^n$ can be used to map from corrupted or noisy data to clean samples or to fill in missing values. For workflow and storage systems denoising can help to preprocess complex telemetry data, such as measurements of storage systems that may be subject to I/O variability. Imputation can be used in scheduling scenarios, for example, where based on an a observed sequence of tasks for a learned workflow, task context, or subsequent tasks, as well as their characteristics, might be predicted even if direct interaction with a workflow engine is not possible.

Clustering and unsupervised approaches help to identify groups that share properties and which can then be addressed independently.[5] Similarly, *semi-supervised learning* (SSL) tasks attempt to learn a representation, that has $\{unlabeled\}\ P(x) \mapsto P(y, x)$ approach $\{labeled\}\ P(X, y) \mapsto P(y, x)$. These approaches are useful to automatically grow a labeled dataset from only a small set of labeled data, a situation often found when classifying storage telemetry and workflow artifacts. In addition, the assumption of workflows makes such an approach especially attractive, as for UQ workflows, for example, we already know that unlabeled artifacts are very likely to share similarities to hand-labeled ones.

[5] Section 7.3.3 (Decision Aid: File Type Clustering and Classification) explores and verifies the viability of this concept on I/O telemetry collected with Darshan to automatically group and classify input and output files.

End-to-end approaches often found in combination with Autoencoders (AE) and adversarial networks encourage the learning algorithm to find internal representations which can be used to denoise input data, compress a complex input or to generate artificial data points, which at the same time can be used to offer introspection into the neural network. Decision aids for workflow and storage might benefit from such approaches as minimal site- or application-specific representations might be learned this way. In combination with multi-task learning these representations might even generalize across sites and inform the process of designing instrumentation.

With this spectrum of applications and problem statements in mind, it is now possible to consider a basic framework that allows training solutions to workflow- and storage-specific tasks programmed by iteratively presenting examples. Machine learning practitioners have identified a variety of building blocks that are found in most learning techniques. Common machine learning notations for reference are provided in Figures 6.32 and 6.34. Their relationship when combined into an algorithm that learns is illustrated in Figure 6.33. Highlighted in blue, red, and yellow are target areas which will require domain adaptation for storage and workflow optimization:

- The *data-generation process* (blue), in the context of scientific workflows and storage systems, for the most part, is concerned with telemetry capture which needs to be combined with workflow and system topology information. Reconsidering the overview of common ML tasks, there is also potential to learn some of these combined intermediate representations. In addition, to develop generalizable solutions applicable to multiple systems, research into the data sampling process is needed, because many machine learning algorithms respond unfavorably to highly correlated features or samples that are not drawn randomly.

$$\hat{\theta}_m : \text{model (parameters)}$$
$$x \mapsto y : f(x; \theta_m) = y$$

(a) Model: The model parameters describe the model and include architecture, weights, and hyperparameters.

$$\hat{\theta}_{m+1} = \hat{\theta}_m + \gamma \nabla \theta_m$$

(b) Training: For training the model is iteratively updated by distributing the error in relation to provided examples (x, y) sampled from a data generating distribution p_{data}. The learning rate γ is used to scale how much the gradient affects the update of the model parameters θ_m.

Figure 6.32: High-level building blocks for many machine learning algorithms *model parameters* and *training step*. Included here because their structure influences how to phrase a learning problem adapted to storage and workflow optimization. How these building blocks of learning relate to each other is illustrated in Figure 6.33.

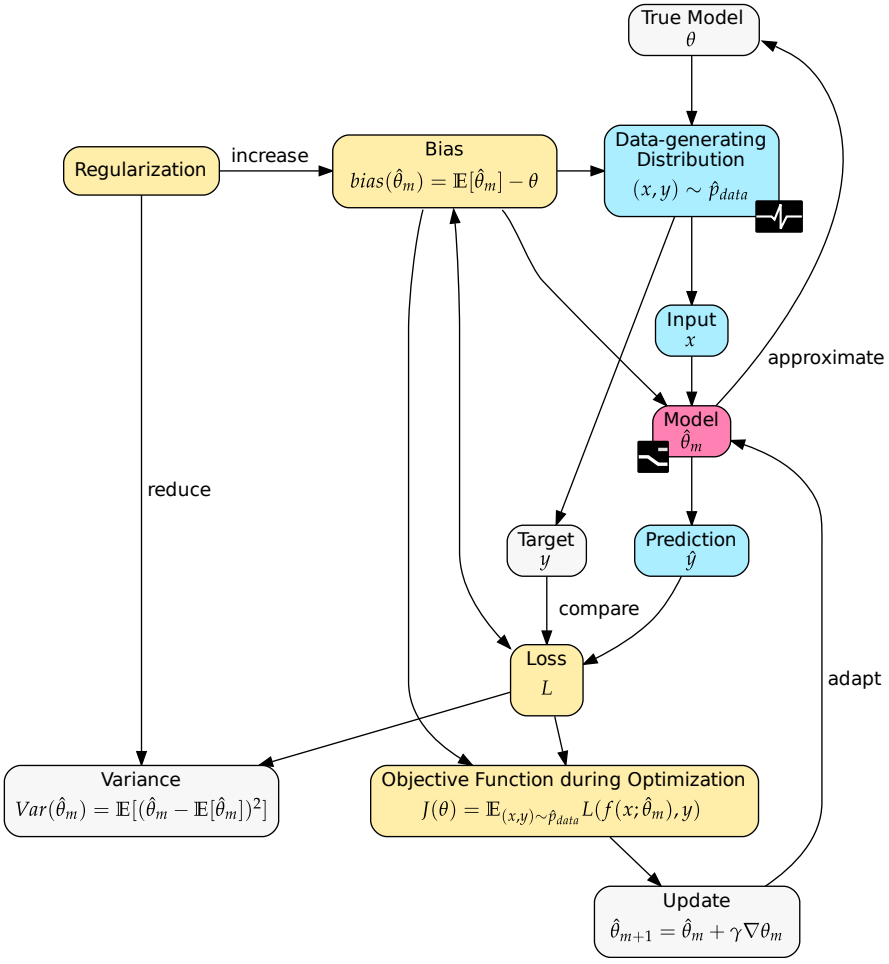


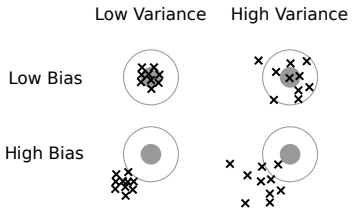
Figure 6.33: Illustration of the model learning process for many neural network-based approaches. Three parts in this view are especially worthwhile targets in support of specialized networks with application for workflow-aware storage optimization. The first is concerned with telemetry capture and versatile intermediate representations (blue), the second with model architecture (red), and the third with systems- or workflow-related regularization methods (orange).

- *Model architectures* (red) require exploration because of constrained performance envelopes when used for real-time decision-making on the one hand, but also because of the opportunity to exploit knowledge about structural relationships that exist in instrumentation data, in workflows as well as in system topologies. CNNs, for example, work so well for image data because they take advantage of the fact that there actually exists a causal meaning in localized features, where *max-pooling* allows summarizing over large regions without requiring a lot of resources. Similar optimization opportunities might also exist for storage and workflow data. Two additional considerations that might occasionally conflict are further architectures that are easy to prune into less costly models in contrast to architectures which allow for easy introspection.

- *Regularization methods* (yellow) and learning strategies, too, can be expected to benefit from domain adaptation. In particular, introducing factors such as throughput, data locality, and energy consumption into the loss function might yield learning algorithms which are more suitable to workflow and storage optimization. At the same time, it can be attractive to avoid departing too far from the more general approaches and instead opt for preconditioning of data and reformulating problem statements reducing the effort of incorporating future research in this area. There is also a hardware acceleration argument to this, as it would be less likely to find more exotic primitives implemented in hardware, although this is most likely only relevant during training and not during inference.

High-level objective is to reduce the expected loss over the training set:
$J(\theta) = \mathbb{E}_{(x,y)\sim\hat{p}_{data}} L(f(x;\theta), y)$

(a) Objective-, cost- and loss functions: To guide the model update into a favorable direction performance measures are used for the optimization process. Loss functions score how right or wrong predictions are. Simply counting is often referred to as optimizing the 0-1-Loss. As this is intractable for large training sets, surrogates with similar effect are employed such as the Negative Log-Likelihood (NNL) in combination with Softmax. The objective function can incorperate more application specific performance metrics but often does not.



$Bias(\hat{\theta}_m) = \mathbb{E}[\hat{\theta}_m] - \theta$
$Var(\hat{\theta}_m) = \mathbb{E}[(\hat{\theta}_m - \mathbb{E}[\hat{\theta}_m])^2]$

(b) Bias and Variance: Intuitive interpretations sometimes characterize the bias as the "error rate on the training set" and variance as "how much worse the model performs on the validation set than on the training set".

(c) Regularization: Methods to reduce variance at the expense of increasing bias. The resulting models cope better with unknown data as overfitting is reduced. Many regularization methods introduce or modify penalty.

Figure 6.34: High-level building blocks (continuation) for many machine learning algorithms *objective function*, *bias*, *variance*, and *regularization* included for reference. How these building blocks of learning relate to each other is illustrated in Figure 6.33.

With these three leverage points in mind, implementers of adaptive or neural storage and workflow tuners can benefit from existing machine learning research by catering towards these concepts.

Toward Toy Problems and Datasets for HPC Workflow-Related Storage Stacks: Unfortunately, a majority of research on specialized neural networks and machine learning is directed to other areas such as machine vision, speech, and robotic applications. But if the outlined tasks map so easily to common problems also found in storage and workflow optimization, why are these methods not more proliferated also in storage stacks?
One reason might be the general lack of clearly defined HPC task statements and datasets to test on.[6] To an extent, this can be attributed to the limited accessibility to HPC systems in general and storage systems in particular. It may be also attributed to a perceived overhead with questionable returns in compiling such. For example, it might be argued that the limited life-time and fast pace of innovation would quickly render any defined task and dataset combination out of date. In addition, it might be necessary to change or simplify the problem statement to such an extent that that there are at least no short-term benefits.
Having often started from initially intractable tasks, the field of machine learning today offers countless examples, that show that this should not discourage the storage and workflow community from seeking these task statements and investing into the curation of datasets to go with them. Training bodies such as MNIST, CIFAR, and ImageNet have helped the field to optimize and advance architectures and track progress over decades. Publications even today routinely include, for example, MNIST performance to allow comparison even though there exist highly efficient solutions to the particular underlying task.

[6] This extends to computer systems in general. An extensive list of popular datasets curated at Wikipedia, lists none related to computer system optimization or fault prediction [Wikipedia contributors, 2020]. Even Kaggle with more than 48.000+ datasets has few datasets related to systems, storage, or workflow optimization. While there is a disk failure dataset from 2016 [Backblaze, 2016], more recent raw data is only available from Backblaze directly [Backblaze, 2020], invisible to a broader audience.

6.6.3 Challenges for Adaptive- and Neural Heuristics

This section discusses challenges and potential limitations when considering adaptive and self-learning approaches for workflow and storage optimization. While currently, two core challenges are in the exploration of suitable architectures and the definition of task and problem statements for workflow and storage optimization, there are a number of less obvious challenges resulting from seemingly conflicting dynamics exhibited by storage systems on the one hand and learning systems on the other. In particular these may concern *overhead*, *reliability*, *explainability* and *generalizability*. While these add to challenges in storage systems research, workflow automation research, and machine learning research, the discussion will only focus on the following aspects which are at the intersection of these research areas:

- *Overhead and Cost:* A more complex decision-making process requires to utilize resources. This on the one hand requires determining when and where the decision should be computed, and the resource investment needs to materialize reasonable gains to be recouped in an amortized perspective. There are also tangential costs to consider, such as incurred by software changes, instrumentation overhead, or the curation of training data, and of course the cost to train a network. Can this still be feasible for use with scientific workflows and high-performance storage systems?

- *Reliability and Loss of Determinism:* Storage systems employ various sophisticated strategies just to achieve reliability and performance, precisely returning the data they were entrusted to persist. In many situations, the highest performance is realized by relying on deterministic design principles that allow avoiding costly lookups or metadata operations. Neural networks disregard both of these principles, they excel when approximations are sufficient and they potentially evolve. Finally, when deployed as autonomous agents, there is potential for interference and even oscillating behavior, which ultimately becomes harder to understand than already existing systems. With storage systems often realizing reliability and performance by exploiting determinism, where do probabilistic qualities of adaptive approaches provide value?
- *Explainability:* While closely related to reliability, explainability is also sought from a research and system optimization perspective. Beyond preventing unexpected behavior, understanding neural networks used for system optimization might expose relevant patterns useful for both users and developers of scientific applications and infrastructure. However, storage systems and their performance are already multi-variate and hard to predict. Machine learning approaches add complexity, for example because their performance also depends on a mix of hyperparameters which are often not well understood. How should adaptive and neural heuristics be constructed to allow introspection or to limit interference?
- *Generalization:* Without sufficient amounts of data or with a faulty training process, machine learning methods are known to overfit. Usually, this is considered an undesirable property. For workflows this is not necessarily as severe, in fact, by training workflow-specific decision components there are even many situations where some degree of overfitting is working to our advantage. At the same time, also for workflow optimization methods are sought which should generalize across workflows and even compute sites. Do state of the art machine learning approaches suffice to provide value to workflow storage optimization, or are more capable and general solutions necessarily needed? How can adaptive and neural heuristics be constructed which promise better generalization?

Overhead & Cost Considerations: When considering adaptive and neural heuristics for storage and workflow optimization it is important to consider how these methods compare to established and classic approaches. This is the case because workflow decisions can be latency sensitive as is illustrated in Figure 6.35. Neural approaches introduce additional complexity but promise to offer relief from manual tuning where the problem contains elements that can be plausibly phrased as the approximation of a function, in the simplest case, for example, as classification or regression tasks.
In comparison to classic heuristics, current neural heuristics typically can be considered more complex and expensive from a computational and also from a memory perspective. Model complexity can be broken down into multiple closely entangled factors:

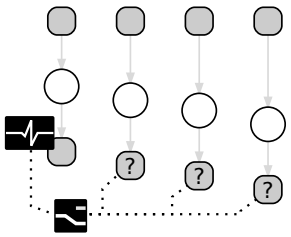


Figure 6.35: Example for a latency sensitive workflow optimization opportunity, where multiple pipelines can benefit from measurements from earlier pipelines.

- Computational model complexity is usually measured in the number of fused-multiple-add operations, which too is referred to as FLOPs, but is not to be directly confused with FLOPS in HPC. Current well-performing neural networks are often in the order of several billion FLOPs.

- Parameter, and thus memory complexity, is dominated by the number of weights and biases of a neural network that encode what the network has learned. Because connectivity between layers is often high, a neural network will routinely feature millions of parameters.

- Training complexity, roughly defined as overall training cost, is driven by hyper-parameters. Hyper-parameter search is used to find the best bias-variance trade off to prevent the model from overfitting. To do so, many networks with different hyper-parameter combinations are trained, often using grid search, randomized search, or genetic algorithms.

- Finally, there is the problem of potentially requiring large amounts of training data D. This is also referred to as the "curse of dimensionality" which states that to approximate a continuous function $f : \mathbb{R}^d \mapsto \mathbb{R}$ with accuracy ϵ in the order of $O(\epsilon^{-d})$ samples have to be considered.

While these factors might appear to make neural heuristics unfeasible in comparison to classical approaches, it should be considered that many decisions need only to be made once for an execution and that the induced latency for model predictions even for very complex models often do not exceed milliseconds to seconds.
Consider, for example, the following approach using a generative adversarial network to generate 256x256 video footage which maintains coherence across up to 48 predicted frames within only about 150 ms on a single TPU core [Clark et al., 2019]. To appreciate the complexity of this task it is useful to realize that the network learns to generate a coherent sequence of images constrained by the dynamics of recorded 3D phenomena based only on the information which were projected onto a 2D sensor. While this might seem unrelated to workflow I/O on first sight, predicting temporally and spatially correlated sequences may also provide utility for workflow-aware I/O scheduling. Instead of considering RGB color pictures, the different channels might code for a workflow allocation mask as well as network and storage utilization for up to 65,536 compute and storage nodes. The prediction would offer expected future utilization under the allocation, for which the scheduler can check for conflicts with other scheduled allocations.
Such timeframes become possible because many of the computations can be performed in parallel and thus be accelerated. Current deployments rarely offer such accelerators near the data path which could calculate such decision aids. However, even more expensive decisions may be recouped from an amortized perspective when considering the cost of transfer and storage latencies, even more so, if repetitively reused in a workflow.
Besides hardware acceleration, machine learning in the past benefited from algorithmic breakthroughs. As new more capable architectures are being discovered, currently unfeasible methods might fit into allowable decision cost-latency envelopes in the future. At the same time, research concerned with the pruning of neural networks [LeCun et al., 1989; Anwar et al., 2015; Molchanov et al., 2017] or approaches which are using parameter sharing such as in Dropout [Srivastava et al., 2014] allow to reduce computational as well as memory requirements.
As for the requirement of training data, systems instrumentation allows collecting extensive amounts of training data. A problem here is that training data might not be labeled, which in part can be addressed by using synthetic benchmarks to automatically produce large amounts of labeled data.

Similarly, using a combination of workflow engines and generated workflows, large amounts of synthetic task-data relationships can be generated. Nonetheless, the community should invest in the curation of training bodies and the definition of learning tasks such I/O motifs for recognition, for example, from workflow I/O workload generators. This is because other research on neural networks demonstratedly has benefited from such in multiple ways in the past: Not only did the research community engage in initially unsolved problems but once an architecture could demonstrate to effectively learn a problem of a certain class, researchers were able to come up with less expensive models to achieve the same task.

Neural heuristics also engage humans in the hardware and software integration process as well as in the training and tuning process. With the introduction of the Decision API and the Decision Aid Propagation Service (DAPS) the burden of integration is lessened and becomes an infrequent exercise only necessary when interfaces change.[7]

As for the training and hyper-parameter challenge a cost comparison should consider the amount of expert time invested into engineering a special-purpose solution in contrast to the time necessary for collecting and labeling sufficient training data which in many cases might also be performed by regular users. As for hyper-parameter optimization, there is research on so-called AutoML systems which would perform parameter sweeps automatically, or genetically explore different architectures. While this can require substantial computational resources, it may still be more cost-efficient than employing an expert.

As a consequence, intangible problems today should not necessarily prevent the workflow I/O community from defining toy problems that appear desirable for storage and workflow optimization. Although these toy problems are sometimes disregarded as too detached from practical deployments, machine learning research benefited from their definition in the past as it increased problem understanding to the point that today various practical applications can be traced back as having evolved from solutions designed for solving such toy problems. Similarly, attempting to solve a problem with an architecture that appears to be too expensive at the time still provides value because often it can be pruned and optimized once it was established that the architecture can solve a task.

Finally, adaptive heuristics will not need to leverage neural networks but more appropriately might be implemented as hybrid models. Mixing traditional heuristics with small neural approaches would not only limit overhead but also constrain unexpected behavior.

Reliability, Performance and the Clash with Determinism: The dynamics of neural networks on the one hand and storage systems on the other seemingly feature opposing dynamics. Storage systems typically establish reliability and performance by exploiting stable physical systems and deterministic algorithms, while neural networks work best under the assumption that probabilistic approximations and evolution over time are acceptable.

So while these different dynamics do limit and prevent considering neural heuristics in some functions of storage systems, there are also various functions that can be augmented using neural approaches. As these boundaries are not well established in storage and workflow research, it is important to consider how to specify and check correctness, reliability, and performance while accounting for trade-offs and side-effects offered by neural methods.

[7] The Decision API and the Decision Aid Propagation Service (DAPS) were introduced to decouple the dissenting pace of evolving middleware and hardware, but also because the publish/subscribe dynamics allow to better optimize decision calculation and propagation.

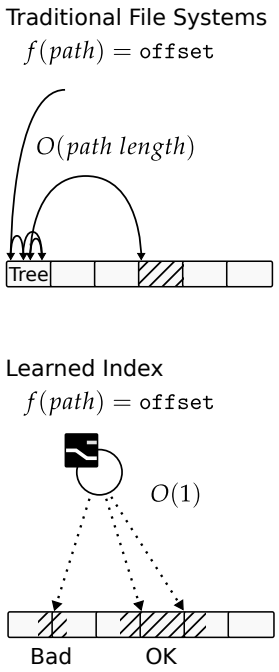


Figure 6.36: Tree based hierarchical lookups in comparison to an examplary learned index structures. Lookups can become faster but occasionally predicted offsets return by a learned index structure may be invalid. If this is a rare event, they still offer opportunity to speed up I/O operations.

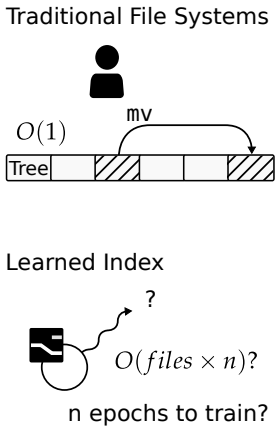


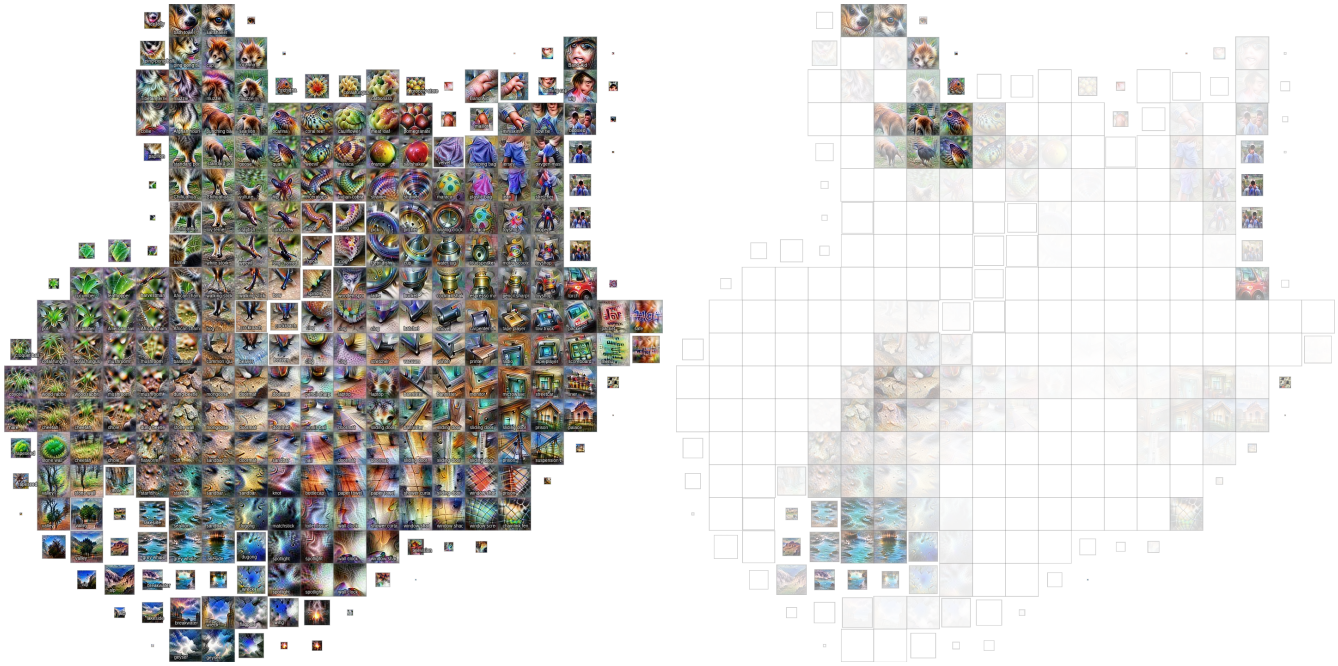
Figure 6.37: Different operations may impose different challenges and opportunities. Consider for example updating a learned index which may potentially be expensive if it has to be retrained, while cost are very low with a traditional tree-based index. Although often simple workarounds may be found, such as adding exception tables for move operations, doing so adds undesired complexity.

Correctness in storage systems in this context should be defined as summarizing fulfillment across multiple factors in particular reliability and performance. Thus while reliability on a high-level evaluates the promise to return bit identical data as it was originally persisted, performance would describe the latency to fulfill a request for data. Correctness thus only can be evaluated under a specification and a set of policies, which might allow for various trade-offs or side-effects depending on the use case.
Yet, assessing correctness is also challenging for existing storage systems due to the dynamics of emergent behavior in distributed systems. In practice, no technical system can offer absolute reliability, but by utilizing various error correction and redundancy techniques it is typically possible to design data safety according to a specification although potentially at a significant cost. As a result data safety as part of reliability is already only a trade-off that prevents data loss with a certain probability
Similarly, performance in many distributed storage systems is realized by taking advantage of deterministic algorithms that allow to avoid consensus finding or relying on centralized lookup services. Data placement for example might rely on hashing functions instead of index structures such as trees, thus avoiding metadata requests as targets can be calculated by clients independently.
When aiming to replace entire functional units in storage systems altogether, the suitability of neural networks remains an open research question. When instead aiming at supporting, for example, existing indexing and lookup strategies to improve performance, neural approaches could alternatively be integrated as auxiliary indexes that are built for a workflow without replacing indexes used to ensure data safety, see Figure 6.36 and Figure 6.37.
That such learned indexes can outperform tree-based lookup structures, as well as classical hash-based approaches, was demonstrated for hierarchical as well as spatial namespaces [Kraska et al., 2017; Wang et al., 2019].
When taking a workflow perspective it seems plausible, that workflows often experience a number of I/O motifs for which optimized indexes can be generated. As such, the trade-off when adaptively deciding where to store information effectively becomes a trade-off between risking to forget where data was stored or accepting additional overhead for bookkeeping.
When simply augmenting existing systems, data reliability might remain unchanged while allowing to improve performance in a limited number of special cases, as may be present on a workflow to workflow basis. In this perspective, neural approaches might be generously applied in situations which are effectively concerned with caching or secondary indexing.
Extending on the idea of hybrid systems, unintended side effects may be limited by employing neural architectures that remain conceptionally close to established algorithms and strategies. Scheduling, for example, makes extensive use of computational graphs to discover data locality and opportunity for parallel computation. Unfortunately, this might require to substantially adapt existing ML architectures to be compatible with existing systems. Instead of aiming to train a neural scheduler, a neural network might add or adjust node and vertice attributes of the graphs used as inputs for some traditional schedulers.
Geometric or graph neural networks, may be particularly useful tools for many other workflow-related storage optimizations. Graph neural networks, however, are a relatively new field of deep learning [Bronstein et al.,

2017; Wu et al., 2020] even though combining machine learning and graphs is not a new idea [Gori et al., 2005].
That seeking heuristic approaches is necessary is also supported by the fact that subgraph isomorphism is NP-complete [Cook, 1971], so that heuristic approaches are needed for which graph neural networks are promising candidates when seeking to discover I/O motifs in workflows. There are, however, also a number of limitations for current approaches used for pattern finding in graphs using neural networks. In particular, some of the cheaper approaches to graph convolutions are isotropic and thus do not account for directionality. While this can be also a strength as it offers rotation invariance, a lack of directionally seems problematic when modeling read and write relationships in workflows which are directional.
Finally, when deploying multiple adaptive agents there is the potential of complex unintended interactions. In particular, autonomous agents with the mandate to optimize for a given workflow will compete and interfere with other agents optimizing for another workflow or the overall site. Such unintended behavior can lead to oscillations or complex cascades that are hard to discover or anticipate. In a storage I/O scenario, a data placement decision component that accounts for storage fill level might repetitively trigger undesired data movements as it monitors two potential storage targets, and every time data was successfully transferred from one to the other, decides that the former target is now more attractive again.Often there are simple solutions to mitigate such effects, for example, using additional metadata to impose timeouts or delays before an action may be repeated.
As experts struggle to understand the dynamics and the complexity of current systems, the addition of adaptive self-learning approaches only increases the urgency of developing better inspection methods.

Explainability and Error-Prevention: An often raised concern with machine learning, and neural network-based methods, in particular, is limited explainability. While this criticism is valid for many early methods and large networks in particular, research in these areas has produced a wide range of methods and architectures which offer leverage for introspection. Leverage, because in many cases the neural network would not find representations that are directly interpretable, but instead by backtracing activations or by sampling generated examples it becomes possible to indirectly reason about the decision boundaries. This can require significant computational investment. Similarly, many simple neural networks are vulnerable to noise, out of distribution prediction or adversarial attacks partly as a result of overfitting to the training data. At the same time, there are mitigation strategies to increase robustness against such failure modes.
Robustness, for example, can be improved by model averaging, ensemble methods or boosting, all of which basically train multiple models and then let each model cast its prediction as a vote into a consensus finding algorithm. While this might seem wasteful, due to reduced connectivity this is not necessarily the case as demonstrated by *boosting* which combines multiple weak and therefore less computationally demanding learners to form a strong one [Schapire, 1990; Freund and Schapire, 1997]. Other approaches which are more robust to noise or even malicious inputs are Generative Adversarial Networks (GAN) as they are typically trained to discriminate authentic samples from synthetic ones, forcing the network to find a compact internal representation capturing the essential characteristics needed to

perform characterization and synthesis.
Methods from representation learning such as (Variational) Auto Encoders (VAE) and GANs are not only effective at producing useful dimensionality reductions but typically also allow to generate output which can be useful for introspection. While this is important for ensuring reliability, introspection is also sought to generate scientific or operational insight. Thus when considering neural heuristics for storage and workflow optimization, favoring methods that are designed for robustness or which provide leverage for introspection may be favored in many cases. This way it becomes possible to discover usage patterns that can help to inform users, software development, and system design.
Best known for offering some intuitive introspection for humans are two-dimensional CNNs, which when visualizing the learned kernels directly reveal which features the network responds to. While this is a result of the architecture, interpreting the hierarchical processing of input signals being amplified or dampened as they propagate through the network requires more sophisticated tools. Particularly user-friendly is, for example, CNN Explainer which allows to interactively trace back and highlight the effect of activations throughout the network [Wang et al., 2020].



(a) Clustering of activations compiled into an activation atlas which can be obtained when using, for example, auto-encoders to generate outputs for various activation permutations.

(b) Only highligthing activations which positively impact the classification of a *sealion* which in this example decomposes into features which appear to be animal faces and bodies of water.

Figure 6.38: Example of an activation atlas giving an overview of learned features a) and highlighting only feature relevant for a particular label b). This and other examples to build a better intuition into activation atlases can be obtained from the interactive tools published at [Carter et al., 2019].

Another such approach are *activation atlases* which use activation maximization to visualize what a network has learned as well as potential failure modes [Carter et al., 2019]. An example of an activation atlas is shown in Figure 6.38 which shows (a) a clustering of various compiled activations and (b) the same atlas with only the activations which affected a particular label. For images showing sea lions for example the network finds two signals to be especially relevant, which when visualized translate into what could be animal faces in combination with water. The example considers actual photographs, as no sufficiently large training body for I/O related data was available to demonstrate the effect with a similarly intuitive example.

While it may not always be possible to generate visualizations on telemetry or workflow data which are equally intuitive to interpret, there exist efforts such as [Lundberg and Lee, 2017] working on more unified approaches to perform similar analysis on arbitrary models.
Another problem when informing decision-making stems from uncertainty. The result of a decision aid such as a performance prediction, for example, may decay in certainty as time advances. On the other hand, dynamic workflows such as the post-mortem analysis of trajectorial physical phenomena, such as turbulence or tropical storms, might offer an opportunity to prefetch regions of interest for follow-up timesteps. For structured data, such trajectories can be obtained from monitoring access patterns, but a resulting prediction is still subject to uncertainty.
Most neural architectures are not designed to account for or propagate such uncertainty. For costly or risky actions, it is often favorable to discard a prediction and remain passive, but this requires methods that assign prediction results with a measure of confidence. Promising architectures that offer such capabilities are VAE, Bayesian Neural Networks (BNN) but also traditional approaches such as Markov Models. Unfortunately, these many of these methods are still an active area of research and often hard to train. When curating regular labeled training data assigning fair uncertainties to such label pairs is often challenging because already the prior probability distributions are not well established. It may still be worthwhile to consider, for example, Bayesian neural networks (BNN), because they offer, if successfully trained, intuitive means to perform causal inference and logic deduction in addition to modeling uncertainty.

Generalization A common problem with machine learning methods is overfitting which often results in neural networks only being applicable to the very specific task they were trained for. In fact, a network's skill often significantly depends on the distribution of training examples limiting deployability in different environments which diverge ever so slightly. Typically, overfitting is limited by employing regularization which increases bias but reduces variance. This paragraph, therefore, discusses potential implications for workflow and storage optimization and as well as strategies that allow training models that generalize across similar tasks as well as future unanticipated applications. For neural approaches to be feasible, the following discussion outlines a roadmap where neural heuristics provide value in short-term, mid-term, and long-term perspectives:

- In the short-term, neural heuristics can provide value when given contextual hints which allow neural heuristics to remain simple. But multiple networks for different workflows and contexts need to be trained.

- In a mid-term perspective, *representation learning* and *transfer learning* allows to combine neural heuristics to robustly generalize for a specific workflow but across multiple sites, or for a specific site across multiple workflows in part enabled by forms of multi-task learning. Different stakeholders independently have incentives to develop such more general representations as it allows to limit resources needed to coordinate and exchange decision aids.

- In a long-term perspective, advanced approaches such as *reinforcement learning*, *progressive learning*, and *meta-learning* may be used to speed up

adaptation to new architectures and to preserve more universal concepts learned by the networks. In combination with evolutionary approaches and *system simulation* some degree of automatic exploration and optimization under policy constraints may be possible.

In support of the short-term perspective, it is important to note, that by taking a workflow perspective to storage optimization many undesired consequences of overfitting can be reduced. Instead of requiring a detector that works across a wide range of different workloads with different signals, ambiguity, network complexity and training times can be reduced. This is the case because in many situations a workflow will only feature a very limited amount of distinct patterns. Thus, by exposing workflow context, for different decision contexts, different learners can be trained which often can be held to lower standards while still providing value.
From an operational perspective, training of too many different models is undesirable as both training and preservation of trained networks require precious resources. Also from a research perspective, more generalizable approaches are sought, as discovering more universal patterns might reveal candidates for system and software improvements. But comparing independently trained networks is usually not possible due to the random initialization or because their internal representations change meaning as training progresses. Thus, intermediate representations are sought which should generalize across workflows and even compute sites.
Such intermediate representations increasingly might emerge in the mid-term as maintaining multiple models becomes impractical. By turning to *representation learning* and *transfer learning* approaches, networks trained for one task can be repurposed by retraining only a small fraction of the network to obtain a mapping to a new task.
Transfer learning takes advantage of the fact, that internal representations found for the original task might be useful for other tasks as well. The challenge here is, to learn a representation that captures all necessary latent information and ensures they are propagated and preserved for application in a later task. Particularly useful are intermediate representations obtained using, for example, Autoencoders, which can be trained without supervision when tasked to reconstruct the input. By forcing information to be reduced to fit through a bottleneck, only essential information is encoded in the latent representation. In a second step, a much simpler neural network can be trained which takes the latent representation and maps it to a target. This way, telemetry information can be aggregated and propagated as a decision aid, which serves as an input for secondary decision aids.
Thus carefully phrasing the learning tasks can later allow the recombination of neural heuristics for unanticipated tasks. Multiple strategies to maximize the generalization of such intermediate representations exist. Especially suitable for the optimization of workflows and storage I/O appears to a setup of the learning task as a *multi-task learning* (MTL) problem. While the ideas go back as far as [Caruana, 1993], multi-task learning only recently regained broader recognition in the machine learning community. MTL operates under the assumption that different task share structure, which is captured by the introduction of shared model parameters.
Setting up multiple tasks simultaneously requires to setup these tasks in end-to-end pipelines and ensure synchronization of shared parameters between tasks, as illustrated in Figure 6.39. For many problems in the work-

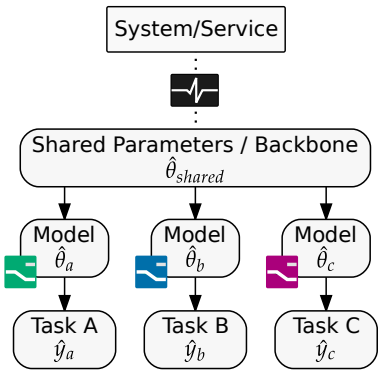


Figure 6.39: Parameter sharing as done in multi-task learning (MTL) finds an intermediate representation that is useful across multiple tasks. Different models and tasks in this context may specialize for different workflows or for two different storage decisions at different levels in the stack which require different inputs.

flow I/O and storage domain, setting up such end-to-end scenarios requires a lot of library and application changes. Here the Decision Aid Propagation Service (DAPS) in combination with the Decision API becomes valuable, as shared parameters can be registered as a decision aid to which different neural heuristics can subscribe.

This architecture also becomes relevant, for a more ambitious long-term perspective which takes advantage of *reinforcement learning*, *progressive learning*, and *meta-learning* for (semi-)autonomous exploration and to reduce adaptation times to new or changing systems.

HPC systems while traditionally relatively homogenous, become increasingly heterogeneous. For example, some systems are procured in multiple phases or get storage or compute extensions. Such changes effectively invalidate a part if not all of the gathered experience if it does not generalize to some degree. Established ground truths such as peak performance are also subject to change, in addition to extensions, storage systems occasionally experience aging effects or performance degradation if fill levels are high. So despite having made the effort to allow recombination of decision aids and neural heuristics, some system changes may be too dramatic to allow to continue using already trained heuristics because their predictions have become useless. One possibility is to reset and retrain decision aids and neural heuristics, but this can take some time before the system becomes effective again.

Relief might be offered from *meta-learning approaches* such as *progressive learning* or *curriculum learning*, which start to train on, for example, a simplified task and then progressively adjust the target, typically, increasing the task complexity [Elman, 1993; Bengio et al., 2009; Fayek et al., 2020]. Progressive learning aims to solve a new task by utilizing previously acquired knowledge. Similarly, curriculum learning aims to challenge an existing network with different tasks. Considering the system extension scenario, instead of exposing the entire new system at once, the neural heuristic might be able to adapt by enabling new system partitions in multiple steps. Similar to a study curriculum the order of tasks can help the network to more quickly reach a state which allows solving the actual target problem. It seems plausible that neural heuristics which should be deployed to act within a complex system such a supercomputer might benefit from such strategies for example to imprint policy constraints or to gradually adapt from an old system to a new one.

An alternative approach to optimization under policy constraints may also make use of *reinforcement learning* (RL) and *evolutionary approaches*. Within a running system, this requires exposing both decision points where actions to optimize I/O can be taken, as well as performance feedback. In the proposed architecture introduced in this chapter, actions are exposed through the Decision API, while the system state can be obtained through the Decision Aid Propagation Service. The basic feedback loop for reinforcement learning is illustrated in Figure 6.40. In RL an agent (the neural heuristic) monitors the system state and is presented with multiple actions that manipulate the environment. As the neural heuristic performs actions, it attempts to maximize reward by exploring different sequences of actions. For workflow storage scenarios, this sequence of actions is found, for example, when considering tunables for different tasks, and reward may be defined relative to expected storage peak performance.

To explore parameter spaces in parallel, to save cost, and to minimize in-

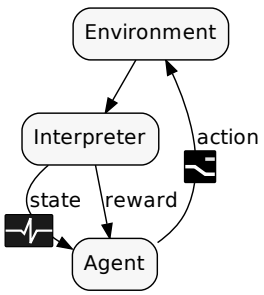


Figure 6.40: Basic feedback loop employed in *reinforcement learning* approaches, which can be expected to become more relevant also to system architecture when thinking of different subsystem in a distributed environment as more or less independent actors governed by a reward function which accounts for a variety of policies put up by site operators.

terference with production systems, it may not be feasible to perform (early) exploration on the actual system. Instead, (semi-) autonomous exploration and optimization using reinforcement learning and evolutionary approaches likely require integration with *system simulation*, see Section 6.7 (Integration with Synthetic Benchmarks & System Simulation). While such surrogate models may not be exactly accurate, they may offer enough similarity to the actual system to allow pre-trained neural heuristics in the spirit of curriculum training.

6.6.4 Summary of Learning & Decision Components

This section discussed the requirements and challenges for decision aids and decision components that want to take advantage of machine learning methods. The discussion started with an integration overview covering the complete feedback loop to influence workflow I/O related decision-making across HPC storage stacks. The discussion highlights the need for interfaces and common concepts to allow interoperability between workflow engines, middleware, telemetry representations, and learning and visualization tools. With machine learning research offering the capability to approximate arbitrary functions and even model non-linear mappings, the section outlines where established machine learning task statements and solutions map to applications for workflow I/O optimization. The discussion then proceeds to identify domain adaptations the storage and workflow communities might want to consider to make machine learning-based approach more suitable also in resource-constrained and real-time environments. In particular, this requires tuning of the data-generation process to ensure fair sampling, the exploration of novel model architectures better suited to capture locality and contention in distributed systems, as well the identification of regularization methods suitable to penalize based on policy considerations from an operating perspective.

The section concludes with an outlook on various challenges machine learning approaches when used for workflows I/O optimization are facing, such as overhead as well as limited reliability, explainability, and generalizability. In particular, storage systems often achieve reliability and performance by exploiting determinism while neural network-based approaches are typically non-deterministic, often require acceleration hardware, and potentially behave unpredictably when confronted with inputs diverging from the training distribution. The discussion highlights areas, where neural network approaches nonetheless provide utility, as computational intensity often can be limited to training or reliability concerns, can be side-stepped when deployed as auxiliary indexes. When choosing architectures wisely, neural approaches even allow for introspection and as such might inform research and operational aspects. Finally, limited generalizability may be lessened by turning to end-to-end and multi-task learning while possibly integrating with synthetic workload generation and system simulation to provide sufficient training examples.

Consequently, and because other domains such a machine vision, natural language processing, and robotics which today attract a majority of research activities in machine learning, benefited from the definition of toy problems, the storage and workflow communities too should make an effort to curate worthwhile problem statements, datasets and workload generators.

6.7 Integration with Synthetic Benchmarks & System Simulation

When aiming to deploy autonomous agents in support of workflow and storage optimization a key concern is limiting unintended behavior and reducing the time it takes to acquire a neural heuristic which reliably solves a task. This section discusses how synthetic workloads and integration with system simulation can help to verify and pre-train neural heuristics. The section further discusses how the previously proposed architecture, in particular the Decision API and the Decision Aid Propagation Service (DAPS), help to remove development overhead needed to transfer between simulated and production environments.
Besides the two already mentioned reasons to seek such integration additional motivation comes from recent machine learning research, which integrates reinforcement learning and genetic algorithms as means to train intelligent adaptive agents and perform automatic exploration as discussed in Section 6.6 (Learning & Decision Components). An integral part of the successful training of intelligent adaptive-agents in complex environments was the provision of sufficient experience through simulated playgrounds and environments. Exploring the viability of all of these for storage and workflow optimization is out of the scope of the thesis, but multiple design decisions cater toward allowing integration with such in the future.
To demonstrate how the first two objectives as well as a possible integration with reinforcement learning can be achieved the section first provides an integration overview and then proceeds to discuss the role of the two core building blocks, *synthetic workload generators*, and *system simulators*, before concluding with limitations and open challenges. The section is broken up into the following subsections:

- Section 6.7.1 (Integration Scenarios to Pre-Train Neural Heuristics) starts with an overview focused around two slightly idealized scenarios: The first is concerned with the speed-up and pre-training of neural heuristics and the second describes more explorative end-to-end scenarios aimed at training particularly capable agents. Essential to both are synthetic workload generation and fast enough surrogate simulation models for HPC systems. Synthetic workflow workloads are relevant both in real systems as well as in simulated ones because they allow generating large amounts of labeled data, and thus provide sufficient training samples for neural network-based approaches to learn patterns.
- Section 6.7.2 (Discrete Event Simulation of HPC Systems) discusses the component and granularity requirements for a computationally affordable simulation model. For a simplified storage systems the major storage services commonly found in a HPC data center are covered.
- Section 6.7.3 (Summary of Synthetic Workload and System Simulation Integration) concludes the discussion with limitations and open challenges that prevent such approaches from providing value today. One challenge is the lack of interoperability which becomes more manageable if decision aids in both simulated and production systems are using the same interface, for example, by using the Decision Aid Propagation Service (DAPS). Unfortunately, even though capable simulation frameworks for HPC systems simulation exist, many existing simulation models focus on individual subsystems or require significant development effort and calibration.

6.7.1 *Integration Scenarios to Pre-Train Neural Heuristics*

The discussion on learned heuristics as introduced in Section 6.6 (Learning & Decision Components) already raised the question of how to best train such adaptive (neural) heuristics. This section introduces two idealized scenarios that take advantage of synthetic workload generation and system simulation to pre-train neural heuristics. The first scenario is focussed on reducing the training time until neural heuristics become useful for workflow optimization, while the second scenarios is focused on training especially capable neural heuristics with coordinating responsibilities in support of workflow and storage optimization:

- *Scenario 1:* Accelerating production readiness by breaking up training into two phases. The first phase establishes general tendencies within the neural heuristic by training on artificial data, obtained from synthetic workloads on real-world and simulated systems. The second phase takes labeled instrumentation data from real-world workflows to continuously fine-tune and improve the decision aid. This way reducing the time and effort necessary to collect sufficient training data to train initial operationally useful decision aids.

- *Scenario 2:* Enabling rapid-evolution and brute-force exploration in simulated environments. Training more capable autonomous agents, in other contexts such as game design, has shown to be successful on both perfect information games but also in complex multiplayer environments with incomplete knowledge. In both cases, however, significant amounts of training experiences had to be provided, achieved through simulation. In many ways, an optimization decision to improve storage in workflows can be framed similarly to a game where a decision aid's earlier actions have consequences downstream reducing or increasing its score.

The following more detailed use case discussions will motivate why in particular synthetic workload generators and system simulators including system models are the core building blocks requiring research to realize these two scenarios.

Scenario 1: The first scenario focuses on the training of a neural heuristic, for example, to detect and classify access patterns from instrumentation data, which with current methods requires large amounts of labeled training data. Labels might indicate I/O motifs or associate performance with a sequence of actions or instrumentation data. Unfortunately, most collected instrumentation data by itself lacks these labels, and manually labeling even across only a single system in many cases may be prohibitively expensive. Similarly, for recently deployed systems it will take some time to accumulate sufficient training data. Relief for those cases may be offered from both synthetic workload generation on physical systems as well as from simulated environments.
Needed for this are parametrized synthetic workload generators with understood outcomes to produce large amounts of labeled training data. At least for common patterns, if more generalizable detectors fail to materialize, context-limited specific detectors can still be trained as is illustrated in Figure 6.41. The process breaks up into the following steps:

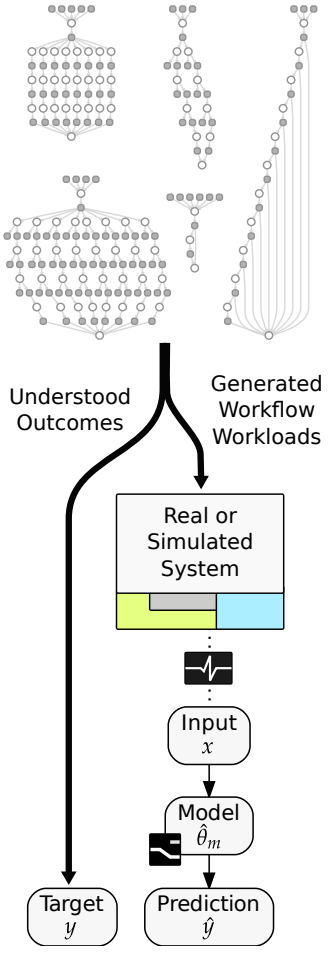


Figure 6.41: Using syntehtic workdloads to pre-train a network, as arbitrary amounts of labeled training data with understood outcomes can be generated.

1. Execute generated workflow workloads and record telemetry and also

record timings, performance, topology masks, or other relevant annotations for key events.

2. Use raw or transformed telemetry to generate *inputs*, and use annotations such as timings or performance measurements to generate *targets*.

3. Use obtained data to train neural heuristics.

A variation of this scenario would use simulated or testbed systems to perform the same sequence of steps to produce training data. Simulated surrogate systems, however, may be more cost-effective, as the usage of discrete event simulations allows to simulate coarse system dynamics much faster than in real-time. As an additional benefit, alternative systems designs and designs which aren't commercially available can be explored, although for both there remains uncertainty about how well the emergent behavior is approximated.
Assuming sufficiently capable and affordable system models are becoming available, synthetic workload generators are also important tools for verification and calibration of system simulators, by comparing emergent behavior on the real system to emerging behavior on the simulated system. This is often done by comparing probability distributions, for example, for request latency as may be appropriate for a storage system. Coincidentally, neural networks are sometimes considered to estimate these probability distributions instead of simulating a semantic model.

Scenario 2: The second scenario is focused on training especially capable neural heuristics, potentially with coordinating responsibilities, in support of workflow and storage optimization. In part, this scenario is inspired by recent accomplishments in more AI-related research, which often couple multiple different types of learning algorithms to complement each other and combine them with synthetic or simulated learning experiences for rapid evolution.
Two types of such learning systems raise the question if analog strategies can also be leveraged for the optimization of HPC storage and workflow scenarios. The first, [Silver et al., 2018] introduced an approach that generalized to multiple two-player perfect information games such as Chess, Go or Shogi, routinely outperforming human players. If it is possible to frame a decision to improve storage performance in workflows in a similar fashion such approaches might also yield utility to HPC workflow I/O optimization. By taking a workflow perspective where all followup tasks are known in advance, a similar such game indeed emerges where an action by a decision aid on some task has consequences downstream reducing or increasing its score. While it remains speculative, if this is indeed possible, the results by [Silver et al., 2018] are particularly interesting because they provide a rough blueprint how to explore an unfeasibly large search space by training different neural networks which would make policy and value predictions while competing for survival within the setup of a genetic algorithm. Although training a network which could beat human players required experience in the order of 40 million games played, wallclock time remains low in the order of hours when taking advantage of parallel computation. While this does not improve overall computational cost, it demonstrates how this might allow to avoid waiting for sufficient real-world data to accumulate. The researchers report they ran about 5000 games in parallel and

a simulated game takes about 3 seconds. A key question is thus if a useful workflow execution on a HPC system model can fit into a similar envelope. Unfortunately, while approaches tackling popular board games like Chess or Go make headlines, the very nature of these games limits also their direct applicability to augment decision making in HPC storage systems. Chess and Go for example are full-information games with a comparable simple set of dynamics: All the rules can be summarized in a relatively short program, and it only takes a small number of bytes to represent the state of a game. HPC storage systems, however, differ considerably in so far that desirable decision components, typically, will only have access to a limited view of the whole system. Secondly, storage systems are shared resources, so there are countless interfering factors adding uncertainty over time. As a result, it may be more appropriate to explore the integration with other methods such as reinforcement learning and end-to-end approaches. [Jaderberg et al., 2018a], for example, takes an agent-based approach which learns actions to successfully play capture the flag directly from the raw pixels of a 3D first-person multiplayer game. An important contributor to the success of this method is "self-play" in a simulated environment. The approach further allowed researchers to isolate a number of different behavioral strategies that were learned by the system. One problem with this approach might be that it requires in the order of 100k to 400k games or 450 million to 1.8 billion played frames (15 frames per second), but then could learn meaningful strategies to cope with the complexity of the game. Again, it seems reasonable to find opportunity in storage systems to formulate a problem similarly, given that large scale distributed systems can also be conceptualized as multi-agent systems.

6.7.2 Discrete Event Simulation of HPC Systems

With the objectives of either accelerating training of neural heuristics or to train particularly capable neural heuristics, this section discusses requirements and challenges for system simulation. To be suitable the following characteristics should be offered by a simulation model for HPC systems:

- *Affordable:* Running a simulation has to be significantly cheaper than running on a production system or extrapolating from a testbed. Cost is not limited to hardware and compute requirements, but also include implementation and modeling efforts. To an extent this can be controlled by adapting model granularity.
- *Fast to Compute:* Simulations should quickly return with useful statistics, for example, summarizing emergent behavior. This is necessary, to allow the generation of sufficient amounts of experiences which can be fed into the training process of neural heuristics. This too can be controlled by varying model granularity.
- *Transferable:* Continuity between simulated and physical systems should be automated as much as possible, for example, by sharing core interfaces and data formats.
- *Accurate:* The simulations should capture tendencies in the emergent behavior. It is unclear, which levels of accuracy are actually necessary. The approaches discussed in the motivating scenarios could rely on an absolutely accurate simulation, as model system and self-play systems

coincided. The second approach by [Jaderberg et al., 2018a] however, did start with a simpler objectives, which suggests that pre-training a model on a simplified system still might be valuable.

A core challenge to simulating computer systems unfolds because of the complexity and the large number of possible combinations for hardware and software. Accurately modeling the emergent properties of an entire HPC systems in most cases should be considered unfeasible at high granularities because it violates acceptable time and resource envelopes. Fast scalable simulation can be provided by frameworks which implement parallel discrete event simulation such as SST [Rodrigues et al., 2011] and ROSS [Jr. et al., 2013]. Still creating a simulated model for an existing or an anticipated data center requires significant effort. Unfortunately, adequate parametrized models are usually not readily available, although community efforts to curate system models useful to perform various HPC storage and interconnect case studies exist as open-source [Mubarak et al., 2014]. But also standardized model components require calibration to match the characteristics of the target system.
Actionable approximations to sway a decision, however, might not even require high-fidelity simulation accuracy as long as the tendencies are correct. As demonstrated in [Lüttgau and Kunkel, 2017] a simplified discrete event simulation can be used to estimate quality of service or total cost of ownership. This and similar cost function predictions would typically be the input for reinforcement learning algorithms. The following paragraphs will provide an overview to this simplified HPC data center model and discuss how the integration with the Decision API and the Decision Aid Propagation Service (DAPS) help to ease continuity between training a model on a simulated environment and then migrating the neural heuristic to a production environment for fine-tuning. The discussion splits into three parts starting with an overview which describes a basic queuing-network-based approach, followed by the discussion of the life-cycle of a request and a discussion of the artificial software stack which is needed to drive the behavior of the simulated components. While the simulation was originally designed to serve as a tool to enable more educated deployment and extension of tape libraries, its models could also be useful to accelerate learning of neural heuristics.

Lüttgau, J. and Kunkel, J. (2017). Simulation of Hierarchical Storage Systems for TCO and QoS. In *High Performance Computing - ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^3MA, VHPC, Visualization at Scale, WOPSSS, Frankfurt, Germany, June 18-22, 2017, Revised Selected Papers*, pages 132–144. https://doi.org/10.1007/978-3-319-67630-2_12

Modeling Hierarchical Storage Systems using Queuing Networks: Instead of modeling individual components aiming to accurately reproduce emergent behavior, it may be sufficient for the training of many decision components to act on a more minimal conceptional system model. While it may be less realistic to hope machine learning approaches can find these concepts completely by themselves, for example, by means of representation learning, using simplified models allows reducing parameter counts into regions for which machine learning approaches have been successfully applied.
Figure 6.42 illustrates how such a simplified model for a HPC data center with a hierarchical storage system might look like. The discussion uses the same architecture as introduced in [Lüttgau and Kunkel, 2017], but adds the perspective of integrating with services such as the Decision API and the Decision Aid Propagation Service (DAPS).
The system consists of multiple queues which handle requests, either relaying or persisting information on different storage systems. Here a large disk-based system combined with a tape system for use as a long-term

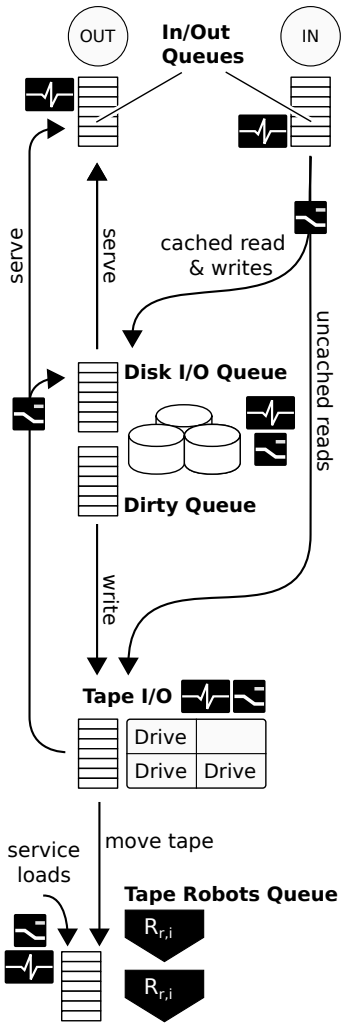


Figure 6.42: A simplified model for a hierarchical storage system which might be suitable as a learning ground for discrete event simulation. The simulation couples a queuing system for I/O operations, with a naive physical models spanning the spectrum of storage services. Exemplary decision points and information sources have been added, which when using DAPS often have an equivalent also in real deployments and vice versa.

archive. As the slowest component in this simulation are the tape libraries, a physical model is used for the tape archive to improve tape seek time accuracy. Here using two-dimensional or graph-based topology models allowed fair approximations of the library dynamics [Lüttgau, 2016]. Calibrating the system still presents major challenges, as various timings have to be taken. Often this is not easily possible and information on proprietary designs can be hard to find.
Monitoring might record activity across the system but focus in particular on entry points for the HSM at large as well into individual subsystems such as the online disk pool as well as the tape pool. Decisions occur for example when an opportunity for caching exists as data is requested or ingested. Similarly, some systems might face the decision to reorganize internally, such as moving data to a faster to reach location, or to replicate for parallel access.
An architecture of the system is depicted in Figure 6.42 which highlight various *information sources* and *decision points* that can be relevant from a workflow perspective. The workflow and the system, in this case, are assumed to operate on file or object granularity, which also allows to reduce model complexity in comparison to, for example, block- or package-level simulations. Requests to this system can be divided into read and write operations for which workflow-related optimization opportunities can arise as follows:

- *Write:* A write requests enters the system at the IN queue (top right). Depending on the request, the system can decide whether the file should be directly redirected to tape, or more typically be temporarily be stored to disk as the disk system can absorb data faster. The file can then be drained to tape in subsequent data movements. In a workflow scenario a decision aid might decide if a file is kept on the disk-based cache or not.

- *Read:* A read request seeks and sends data to a consuming node. As the request arrives in the OUT queue (top left), the system checks if a copy exists on the disk system and if not requests the tape that holds the file. The system then may be confronted with the choice to keep a copy of the requested file in cache. In addition to deciding if a file remains cached, a workflow scenario would also consider staging additional files which would be predicted by a neural heuristic based on the workflow context.

In practice, additional operations on data have to be considered. In particular, the removal of data typically leaves the system in partly unfavorable states, for example, as a result of fragmentation as continuous blocks of data become less common. Most storage systems thus have service routines that rearrange data. In simplified models, these aging effects and service workloads may not be modeled. It is unfortunate, that many similar routines responsible for driving a system have to be implemented also for simulated systems as it increases implementation burden and simulation time.

Integration with Artificial Software Stacks: Such an artificial software stack is depicted in Figure 6.43 which shows the core components (gray boxes) as might be considered for a simplified HSM software stack used in a simulation. Highlighted in black are possible integration points for information sources and decision points as might be considered for neural heuristics in support of workflow and storage I/O optimization.

Components include logic for coordinating components such as the *I/O schedulers*, or the *index* and *location services* such as the *file/object/tape manager*, as well as for subsystem components model the behavior of the physical components. In the simplest case, an action on the physical components just imposes a time penalty for the simulation before a follow-up action can be performed. From the top left corner, covering high-level functionality to the bottom right corner covering individual components model granularity increases. With model granularity or level of detail (LoD), simulation runtime, resource consumption and implementation effort increase as well, so that a more sophisticated model does not necessarily lead to better prediction outcomes as more components require calibration or errors accumulate.

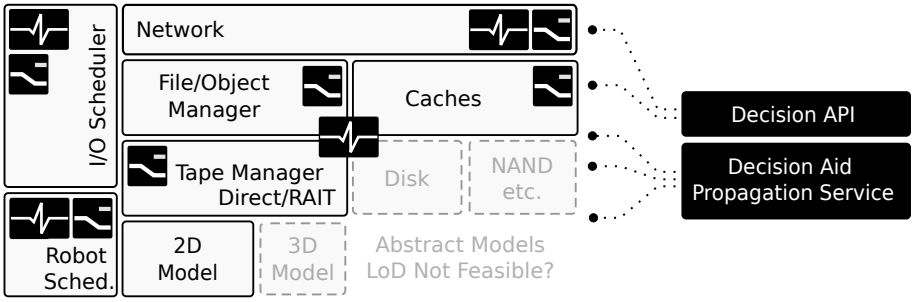


Figure 6.43: Components and layers required to simulate a hierarchical storage system using discrete event simulation. Implementations in such a stack might span semantic, probabilistic, and physical model components. Most components in this abstract component act as both information sources and decision points, which might take advantage of the Decision Aid Propagation Service (DAPS) to establish continuity between simulated and physical systems.

Finally, integration with services like the Decision Aid Propagation Service (DAPS) and the Decision API help to streamline and thus potentially speed up the training of decision aids and neural heuristics by providing interfaces that are shared between the simulated and physical system. Attractive targets for workflow I/O optimization in this context are the components with scheduling and data placement responsibilities such as the I/O scheduler as well as the file/object manager.

6.7.3 *Summary of Synthetic Workload and System Simulation Integration*

The discussion in this section is kept optimistic, but it is important to point out that a large variety of challenges have to be overcome to actually generate value from approaches like the two scenarios to speedup decision aid training or to train especially capable ones. As with the discussion of neural heuristics in the previous section, this discussion is meant to concretize a possible roadmap and better understanding of where research efforts are necessary to lend more "auto-tuning" capabilities to systems themselves and offer some relief to experts and researchers by lowering the time that has to be spent on short-living system-specific optimization.
Both for system simulation but also for synthetic workload generation in HPC, the ecosystem and communities are small and in an early phase. Many system models while conceptionally building blocks that can be rewired to model different HPC subsystems or sites, in practice do not allow this reconfiguration without also investing a lot of effort into recalibration and implementation changes. In particular, parallel discrete event simulation engines materialize speed-ups by requiring the implementation of forward and backward time evolution, to allow the occasional rollback of pre-simulated states as late-arriving events trigger recalculation to reflect the actual timeline of events.
Similarly, middleware and systems are usually not exposing decision points and information sources, again requiring a larger initial effort even if targeting only a number of pivotal software for a particular use case, such as

a specific workflow engine, a specific data format, and a specific storage service.
Finally, even with good system simulators and services like the Decision Aid Propagation Service (DAPS) in place to simplify the collection and propagation of workflow artifacts and decision aids, adaptive and learning approaches for system and workflow I/O optimization in particular are still in their infancy. It thus remains speculative if the promising results achieved in fields like game AI to cope with large action spaces and complex environments really can be applied successfully in system optimization.

Chapter Summary

This chapter introduced an architecture proposal to enable storage systems to better accommodate scientific workflows. With a model procedure to discover, apply and automate workflow related optimizations, a stakeholder analysis is conducted which identified various roles (scientists, developers, operators, and vendors) across participating organizations, which at least temporally find themselves in the need to research I/O behavior. Often these different representatives will have conflicting interests which in current systems are typically not resolved, but should not be ignored especially when interacting with shared storage systems. A number of guiding use cases, and a variety of storage-related optimizations are discussed, namely opportunities to realize I/O-aware scheduling, workflow-aware data layouts, decision support for middleware as well as reconfigure-able networks and active storage.
Driven by these requirements, an architecture consisting of 4 core "components" was proposed: ❶ Discovery tools are required to combine, reduce, and visualize various artifacts collected from workflow engines and different physical systems to reconstruct the performance implications of executing a workflow. ❷ A new interface, the Decision API, for applications, middleware, and systems to expose decision points and as a mechanism to delegate making a decision to an exchangeable component, thus allowing to provide and adapt a decision component exclusively for a particular workflow, user, or application. ❸ A new service, the *Decision Aid Propagation Service* (DAPS) which offers a publish/subscribe interface to allow efficient distribution of decision aids. By decoupling the generation of advice, from the distribution and from consumers, changes to existing software remain minimal while the architecture is flexible to integrate with new systems. ❹ Decision aids, which often are aggregation units that add value to, for example, raw telemetry by reducing it into information which sways a decision component's verdict to favor one action over another.
Finally, there need to be tools to help with potential training and tuning of decision components, especially when machine learning-based approaches are being used. Consequently, the chapter concludes with a discussion on how the architecture proposal was carefully designed to accommodate machine learning and in particular neural approaches that are relatively well understood, while also considering promising new approaches in these areas of research. Parts of this discussion address how system simulation is likely an important tool to speed up training, allow self-optimization or genetic approaches as well as safeguarding a stable behavior before deploying decision components. Here the Decision Aid Propagation Service (DAPS)

and the Decision API can help to bridge between simulated and physical systems by providing a shared interface abstraction. Integration of system simulation, however, remains a more distant vision, middleware and systems currently often do not expose decision points and information sources, machine learning algorithms for system optimization just begin to see more widespread consideration, and system simulation models are usually still too cumbersome to implement and configure for most data centers in comparison to easier to justify and less risky optimization attempts.

7 Evaluation

This chapter evaluates a use case in support of the architecture proposal introduced in Chapter 6 (Architecture for Workflow-Aware Decision Components) to bring workflow-awareness to storage stacks. The evaluation then proceeds with a component-based analysis to inform the implementation of different core components and prototypical decision aids. The chapter is structured as follows:

- *Section 7.1 introduces information about the software requirements, test environments, and clusters that have been used for the evaluation.*
- *Section 7.2 starts with an analysis of real-world applications, such as the ICON climate model, which often required the development of non-existent tools to compile the I/O perspective.*
- *Section 7.3 performs an assessment of different complex decision aids and discusses how they can be integrated with existing systems.*
- *Section 7.4 introduces a benchmark used to inform the latency and throughput which can be expected from a prototype implementation for the Decision Aid Propagation Service (DAPS).*
- *Section 7.5 demonstrates the benefit of considering a workflow-aware storage strategy in a proof of concept looking at amortized costs in a single producer multiple consumer scenario.*

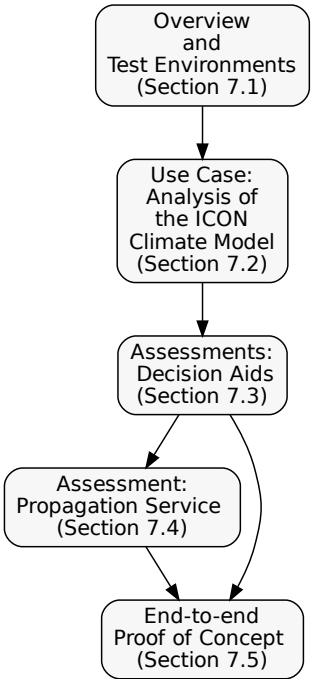


Figure 7.1: Relationship of the different sections in this chapter. Starting from the analysis of a real-world application a number of prototypical decision aids are developed, two benchmarks inform the implementation of the Decision Aid Propagation Services and demonstrates benefits in a workflow scenario.

7.1 Overview & Test Environments

A major challenge to test the framework for decision support introduced in Chapter 6 (Architecture for Workflow-Aware Decision Components) originates in the complex mix of hardware and software systems into which this work integrates. For this reason, the evaluation first performs a component-based analysis of different core components covering discovery tools and the generation of decision aids, before moving on to the discussion of an overarching proof-of-concept. The evaluation is divided into four aspects and is organized as illustrated in Figure 7.1.
As a motivating use case, a real-world application, the ICON climate model is analyzed from a workflow I/O perspective. This analysis exposes multiple opportunities for optimization and serves as a model for the development of a proxy application that offers more direct control for subsequent benchmarks. Derived from this and the proposals from Chapter 6 (Architecture for Workflow-Aware Decision Components) a number of assessments to inform the development of decision aids are performed.
To distribute the decision aids, the decision support framework for storage systems provides a special service. As it was unclear, what kind of latency and throughput can be expected from a Decision Aid Propagation Service

(DAPS), a benchmark was developed using ZeroMQ as a transport in Section 7.4 (Component Assessment: Decision Aid Propagation Service). Finally, Section 7.5 (End-to-end Proof of Concept) demonstrates how the different prototypical components would come together to realize a workflow-aware optimization for the proxy application by relating them to a parameter study that benchmarks performance for the proxy application under various domain decomposition and access sizes.

7.1.1 *Documentation of Software Environments and Required Changes*

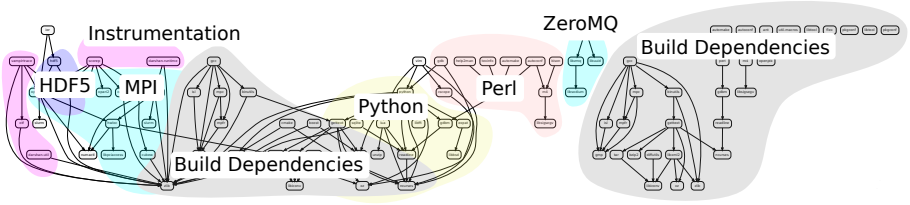


Figure 7.2: Software dependency graph of software managed by Spack for the workflow testbeds on Mistral to illustrate complexity of software trees required in scientific workflow settings.

Software environments and dependencies are managed and compiled using Spack [spack.io, 2020] where possible. In general, this eases setting up a similar environment across different sites considerably. For none of the studies are compiler flags and optimization levels of larger importance, as latency is typically imposed by either the used distributed services or by the use of high-level programming languages which are extensively employed by workflow engines. Figures 7.2 and 7.3 illustrate the complexity the software environments at two different sites. With the exception of Swift and Decaf, none of the workflow engines used in this evaluation could be provided through Spack packages. Cylc and COMPSs both provide installers which also can install most of the dependencies automatically. Cylc requires a Python interpreter, and COMPSs requires a Java runtime to be installed.

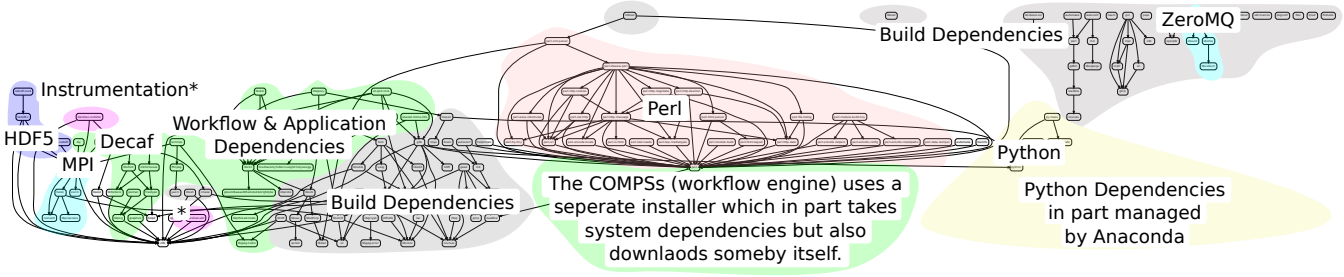


Figure 7.3: Software dependency graph of software managed by Spack for the workflow testbeds on Bebob to illustrate complexity of software trees required in scientific workflow settings. The additional dependencies are in support of the applications for the analysed in situ workflows.

Specific versions become relevant for a limited set of software dependencies, mainly for API compatibility. Table 7.1 lists software and required modifications in detail. Various additions to Darshan and Decaf have been made to conduct the research presented in the subsequent sections. Conflicting versions for sub-dependencies can be largely avoided by populating a packages.yaml provided by Spack.
To conduct the evaluation, changes to various layers and third-party software dependencies had to be introduced. These modifications are also contributions of this research and could be in part be integrated back or as new features into the respective software. Most contributions are to the Darshan ecosystem in the form of analysis interfaces and tools as well as additional instrumentation modules. Changes to Swift, Cylc, COMPSs, and Decaf in most cases are concerned with exposing contextual information about the task or execution context to monitoring and profiling tools such as Darshan.

Software	Versions	Description	Modifications/Integration
Darshan	3.2.0	Instrumentation and profiling for common HPC I/O interfaces such as POSIX, MPIIO, HDF5. Used as the main telemetry source for experiments due to low overhead.	Additional instrumentation modules (C++ prototype, Decaf prototype) and additions to the utility library for data access in newly developed analysis and visualization tools. Python bindings for Darshan to access raw-data for subsequent analysis using Python's data science ecosystem.
Decaf	dev	Middleware to transparently optimize dataflows and transformations for in situ workflows.	Small modifications to allow more meaningful introspection and instrumentation. Used, for example, by the prototype instrumentation module for Darshan.
(Py)COMPSs	2.6	Workflow management system for HPC environments developed at Barcelona Supercomputing Center (BSC). The runtime is implemented in Java but Python bindings are needed for the performed experiments.	Various changes to launch scripts to expose task information and add instrumentation using Darshan.
Swift-T	1.4.1	Workflow execution engine for the Swift DSL to describe workflows a very fine granularity. Multiple runtimes exist, the T (Turbine) runtime is targeting in situ workflows and is using MPI communicators to propagate execution context. It is based on futures to help distributed scheduling.	Workflow graph generation requires enabling logging and uses a convention to associate activity with tasks.
Cylc	7.7.0	Workflow execution engine with coarse granularity for cycling workflows. Cylc originated the climate and weather context and coordinates	For instrumentation, `LD_PRELOAD` has to be added to the environments of executed commands.
HDF5	dev+VOL	Data description format commonly used in scientific applications. HDF5 is a good target for I/O optimizations because besides offering a rich API to express structured data it also offers many tunables.	Small modifications to transparently inject domain decomposition decisions.
OpenMPI	3.1.4	Message Passing Library to coordinate multiprocess jobs and perform parallel I/O.	No modifications necessary as decision aids within MPI not included in the evaluation.
ZeroMQ	4.3.2	Message-oriented middleware used as a potential transport for the Decision Aid Propagation Service (DAPS).	A custom benchmark was developed to assess the performance when using ZeroMQ as a transport for DAPS.
Python	3.6	Interpreter required by some of the workflows, and used for many of the analysis scripts and benchmarks.	Custom data collection, analysis, and visualization packages to combine darshan instrumentation data, workflow artifacts, telemetry, and other system information.

Table 7.1: Description of software dependencies and notes about modifications necessary to perform experiments.

7.1.2 *German Climate Computing Center*

The Mistral supercomputer is operated by the German Climate Computing Center (DKRZ) to provide compute resources and support to users in the climate modeling and simulation research community. The following experiments of the evaluation were run on the DKRZ Mistral system:

- The ICON climate model I/O analysis, see Section 7.2 (Use Case: Analysis of the ICON Climate Model).
- Performance estimates with ZeroMQ as a transport for the Decision Aid Propagation Service (DAPS). See Section 7.4 (Component Assessment: Decision Aid Propagation Service).
- The proof-of-concept and performance landscape measurements in Section 7.5 (End-to-end Proof of Concept).

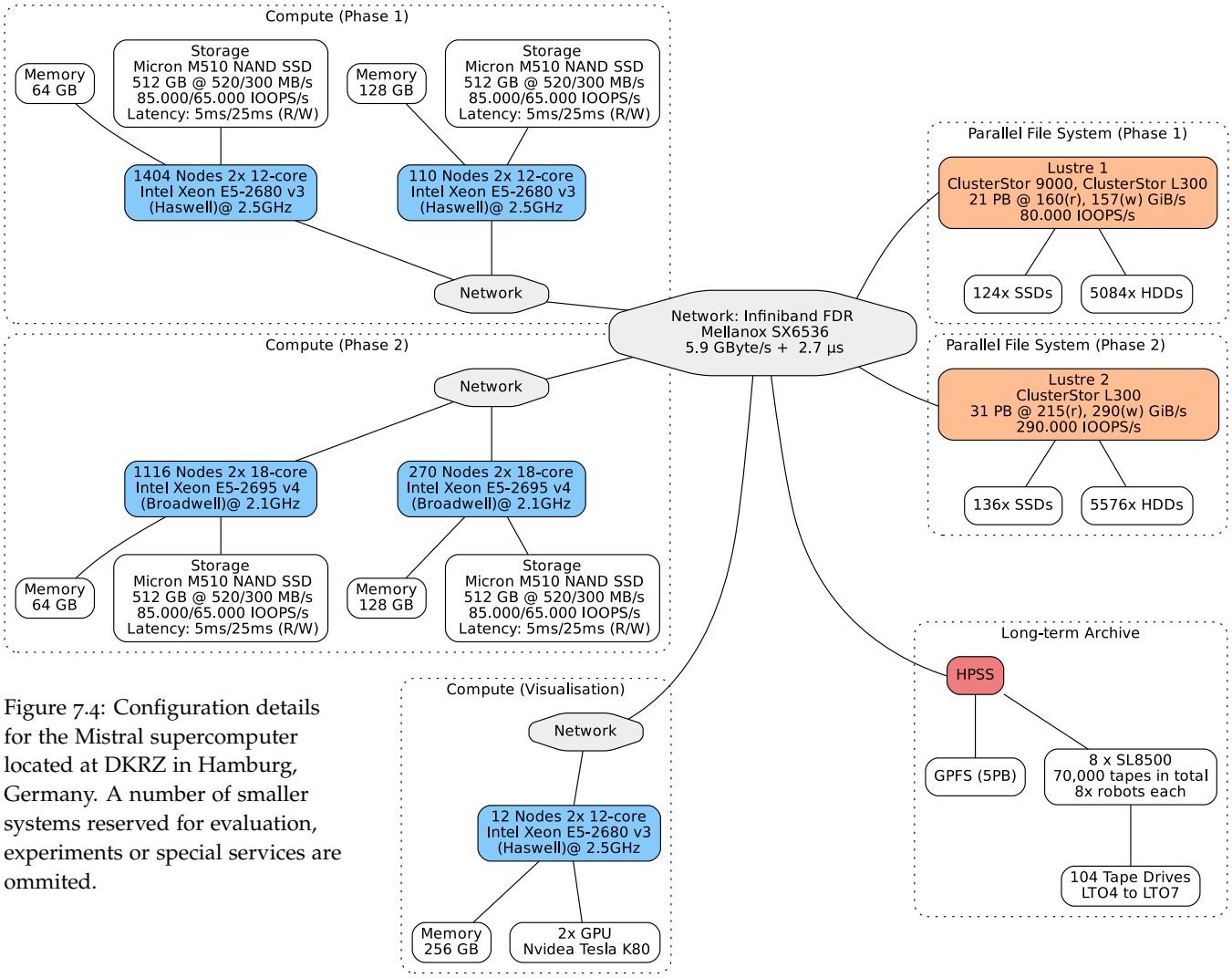


Figure 7.4: Configuration details for the Mistral supercomputer located at DKRZ in Hamburg, Germany. A number of smaller systems reserved for evaluation, experiments or special services are ommited.

Mistral: Featuring 3.6 petaflops with 3380 compute nodes, Mistral was deployed in two phases. The nodes are connected using Infiniband FDR as its interconnect with a sustained Bandwidth of 5.9 GBytes at latencies of about 2.7 microseconds. An overview of the system is illustrating the various subsystems is provided by Figure 7.4. Persistence storage is provided through two Lustre systems totaling 52 petabytes of online storage as well as a tape-based long-term archive. The two Lustre systems come with 5 MDTS for 124 OSTs for Lustre1, and 8 MDTS for 148 OSTs for Lustre2. The

batch scheduling system is using SLURM, and the systems is running CentOS6. Occasionally, this leads to the additional effort necessary to compile modern software suites.

7.1.3 *Argonne National Laboratory*

A number of experiments on real-world workloads, in particular studies in relation to using Darshan for telemetry capture and to explore in situ workflows, have been conducted using resources operated by Argonne National Laboratory. The following experiments included in the evaluation were run on the ANL LCRC Bebob system:

- Workflow I/O behavior reconstruction for nested *in situ* workflows based on Decaf and COMPSs. See Section 7.3.1 (Decision Aid: Augmenting Workflow Graphs with I/O Behavior).
- Performance estimates to inform the design of the Decision Aid Propagation Service (DAPS). See Section 7.4 (Component Assessment: Decision Aid Propagation Service).

Laboratory Computing Resource Center (LCRC) Bebob: Measurements for the Decision Aid Propagation Service (DAPS) were also run on the Bebob supercomputer operated by LCRC. Bebob entered the Top500, 2017 ranking #117 featuring 46,720 cores while achieving 1.07 petaflops Linpack performance. Bebob consists of 1024 compute nodes and provides the OmniPath Fabric Interconnect using Intel Performance Scaled Messaging 2 (PSM2). Omnipath has since been discontinued by Intel.
Bebob jobs are submitted through Slurm, a topology configuration is provided. An overview of the node configurations available on Bebob is listed in Table 7.2. The system consists of two different architectures with the majority of nodes running on Intel Broadwell, and about a third of nodes using Intel Knights landing.

Partition	#Nodes	CPU Type	Cores/Node	Memory/Node	Local Storage
bdw	600	Intel Xeon E5-2695v4	36	128GB DDR4	15 GB
bdwd	64	Intel Xeon E5-2695v4	36	128GB DDR4	4 TB
bdws	8	Intel Xeon E5-2695v4	36	128GB DDR4	15 GB
knl	288	Intel Xeon Phi 7230	64	96GB DDR4/16GB MCDRAM	15 GB
knld	64	Intel Xeon Phi 7230	64	96GB DDR4/16GB MCDRAM	4 TB

Table 7.2: Cluster partitions and configuration of the Bebob supercomputer. [ANL/LCRC, 2020]

7.1.4 *Research and Education Cluster of the Research Group Scientific Computing, University of Hamburg*

A considerable number of investigations used to inform software development and testing would not have been possible without the computing resources maintained by the research group Scientific Computing of the Department of Informatics at the University of Hamburg. The group is operating a variety of different architectures as well as a Lustre installation at a small scale for educational and research purposes.
The research cluster environment has proven invaluable to conduct research without being constrained by the same stability and security considerations as they are present for production systems. Software is maintained using Spack, and Slurm is used as the batch scheduling system.

7.2 *Use Case: Analysis of the ICON Climate Model*

This section analyses the ICON climate model from a workflow I/O perspective. The motivation to do so breaks down into three objectives:

- Identify optimization opportunities in a real-world application that can benefit from a workflow I/O perspective to motivate the discussion and development of prototype decision aids.
- Inform the development of a synthetic benchmark more suitable to perform quantitative analysis of workflow I/O tuning.
- Test existing and newly developed tools able to accommodate the complex I/O behavior of scientific applications used in production.

The analysis demonstrates how complicated it can be to apply even well-known optimization opportunities to existing software. Challenges to transition to more friction-less abstractions are also faced by applications that invest considerable effort into optimizing I/O, for example, by implementing tailor-made two-phase I/O. As a result, a common scheme observed for different real-world applications is that fairly well-known optimization opportunities remain unused.

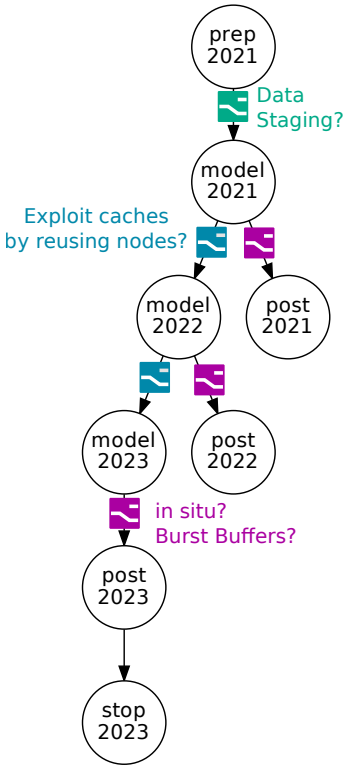


Figure 7.5: Example of a simple climate workflow with different related I/O and storage optimizations highlight. Decision points that share the same color might benefit from being application or workflow specific. The purple decision, for example, might sample performance for different strategies for the 2021 and 2022 step and then settle on the better performing strategy for 2023 and subsequent steps.

ICON Climate Model: The ICON climate model is a global coupled earth system model primarily used to simulate atmospheric and ocean dynamics in earth systems. ICON is using MPI for parallel coordination across compute nodes but also support thread-level parallelism for computation. ICON is using I/O coordinator nodes/processes which collect data using remote memory access so simulation will not have to wait for I/O, thus implementing two-phase I/O. Effectively, this setup emulates a burst buffer, but without providing the same data retention guarantees. The application was run on the Mistral supercomputer.

Workflow-Context for the ICON climate model considered for the subsequent evaluation assumes a workflow similar to Figure 7.5. An initial state for the model tasks is generated by `prep2021`, which is then evolved in time. For each model run, a post-processing task performs some analysis such as the calculation of a temperature anomaly. Highlighted in color are three examples of decision points which might allow improving I/O behavior of the overall workflow. The decision points of the same color are assumed to share the same optimal decision for the given context, and that because multiple post-processing tasks can be observed, there might be an opportunity to utilize adaptive methods to find an optimization.
In a more complex scenario, this sequence also is an adequate blueprint for a single pipeline as it might be performed in an ensemble simulation where the input parameters to each pipeline are slightly perturbed. In this case, multiple model tasks for each year could be run in parallel, which can lead to a predictable situation where tasks are competing for the same shared resource. This gives rise to the scheduling scenario, where knowledge about the I/O phases of a task, job, or pipeline offers interesting opportunities to adapt scheduling decisions to schedule I/O out of phase of each other, compare Section 6.1.5 (Use Case: I/O Aware Scheduling and Staging).

7.2.1 *Profiling Two-Phase I/O and Analyzing Traces*

Profiling and tracing can reveal some valuable information about the I/O behavior of the ICON climate model without requiring to audit any code of the original application, a codebase exceeding 300.000 lines of mostly Fortran code.
Figure 7.9 (page 182) shows an overview of trace visualization of an ICON simulation run with different relevant I/O phases of the application execution annotated. Note that the model's checkpoint interval was deliberately set to a very short value, so that compute phases vanish to fit multiple checkpoint phases into one illustration. Depending on the model parameters such as model granularity, checkpoint phases will be spaced further apart. This also reinforces the need to seek interactive tools as were developed for this kind of analysis. From a workflow I/O perspective a simulation run breaks down into the following phases:

- A brief initial phase of file I/O as a result of setting up MPI, as well as loading the model configuration, much of which is specified in so-called namelist files.
- It follows a phase where initial data is loaded. This is performed collectively by all ranks. Since the simulation uses a multigrid, a number of different NetCDF files to populate coarse-grained global fields, as well as finer-grained regional fields are read on start-up.
- Checkpoints are written regularly using Two-Phase I/O. In this example, all variables collected to dedicated I/O nodes which than asynchronously drain the most recent checkpoint to storage.

The trace was obtained using Darshan with the extended tracing (DXT) capability turned on. As no tool to visualize I/O access patterns representing both temporal access and spatial access (offset in a file) in a compact way existed, a special tool was developed which can render arbitrary DXT traces into a timeline for interactive exploration (as illustrated in Figure 7.9).
Visualizing DXT file access traces allows an intricate understanding of file I/O patterns as will be discussed in the following three examples.
In Figure 7.6 the I/O access to a file `ifs2icon_R2B09_DOM01.nc` is visualized. The information in this file are required to resolve serialized data stored in input and output files to their respective coordinates in the icosahedral grid used by the simulation. All processes require some information from these files, but only 14 out of 196 processes are reading from it.
Displayed in Figure 7.7 is the read activity accessing an 55.7 MiB input file `extpar_icoles_IconChecksuite_DOM01_R4996m.nc` which holds the initial state to drive the simulation. 192 of 196 ranks are accessing this file. The trace visualization also reveals that certain regions in the file are accessed multiple times, but the thin vertical read accesses indicate that data was cached. As the model is nesting two domains, a second file for `DOM02` is also being read which features very similar access patterns.
Finally, Figure 7.8 visualizes the I/O activity recorded on a restart file `atm_icoles_nested_restart_atm_DOM02_20130424T000300Z.nc` of about 196.8 MiB in size. This file is used to continue the simulation when the job is resubmitted if the simulation time exceeds the site's maximum job length.

Figure 7.6: Example of reading coordinates required to map between model and input/output data as an unstructured grid is used. This is performed by a subset consisting of 14 ranks.

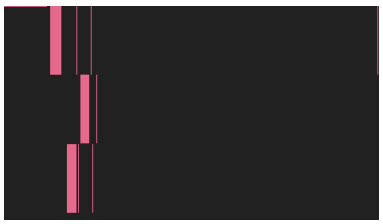


Figure 7.7: Example for the I/O activity for an input file, which shows repetitive access to the same regions which return fast due to the use of caching likely performed by HDF5. This access patterns is shared by 192 ranks.

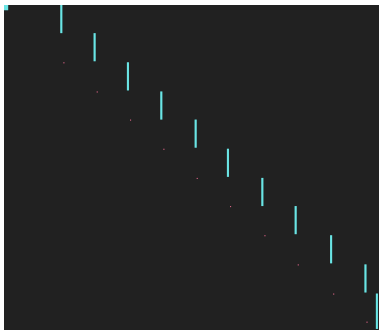


Figure 7.8: Example for a trace of writing a checkpoint file. All I/O is performed asynchronously to compute by a single rank designated as an I/O PE.

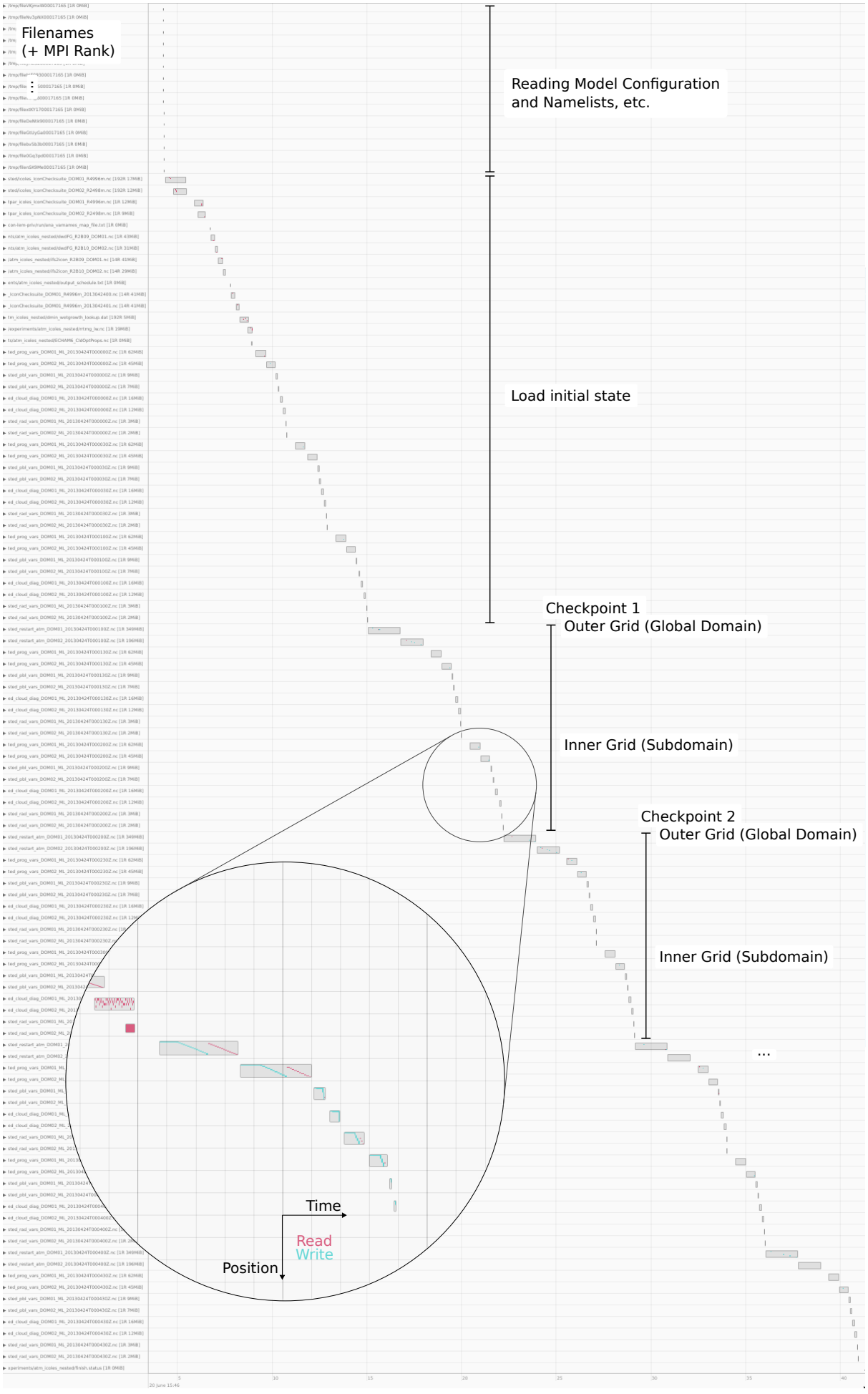


Figure 7.9: Detailed file access patterns touched by ICON, a climate model code used by researchers MPI and DWD.

7.2.2 *I/O Node Placement*

While Section 7.2.1 (Profiling Two-Phase I/O and Analyzing Traces) focused mostly on the temporal aspects of I/O behavior, this section focuses on the physical mapping and spatial relationships. In particular, this section discusses how combining topology- and workflow-awareness can provide a better understanding of performance dynamics.

The ICON climate model distributes work using the notion of processing elements (PE), and distinguishes between compute PEs and I/O PEs. During simulation ICON routinely evolves in the order of up to 300 different variables necessary to compute metrics such as temperature, velocities or radiation. Each of these high-level metrics might be composed of multiple variables as many models rely on parameterization. The dynamics of various sub-models as well as which variables are ultimately loaded and later written to snapshots is configured through namelists. In particular multiple ranks, or I/O PEs can be assigned responsibility for the collection and storage of a subset of variables. Typically, these namelists are generated by the job script when passed to the job scheduler. The job script features multiple such *decision points* which influences the subsequent application behavior. Figure 7.10 illustrates how ICON might map compute PEs and I/O PEs (45-47, colored green) onto a list of nodes after receiving a node allocation. Typically, this topology overlay view would not be provided to users, which when using the default mapping results in all I/O PEs ending up on the same node which leads to unwanted contention.

This serves as a good example of why a service like DAPS requires interoperability between different toolchains such as command-line utilities for bash scripts. Other common decision points on the application level are discussed in Section 5.3 (Application Level).

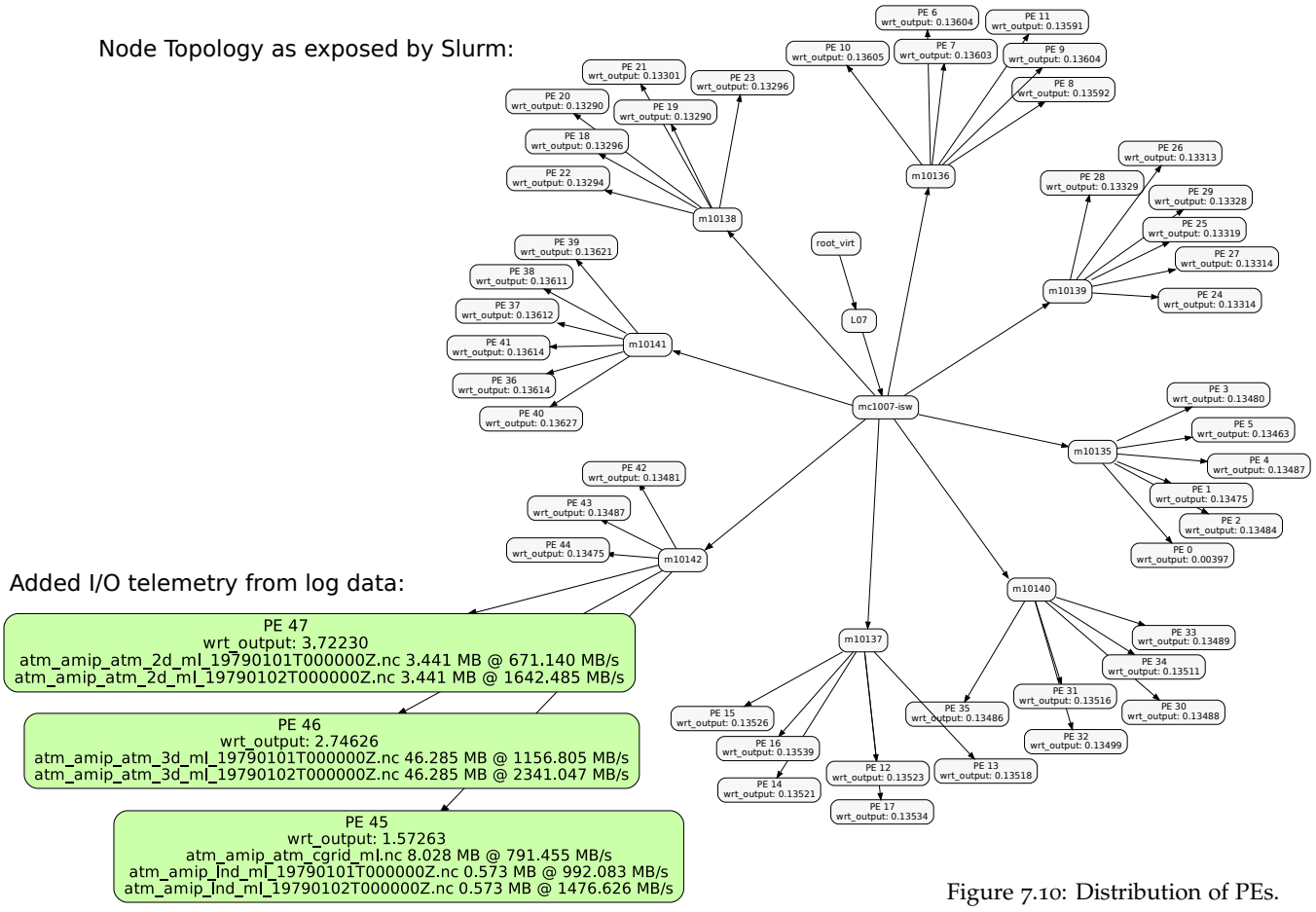


Figure 7.10: Distribution of PEs. Note the I/O PEs responsible for writing to the parallel file system.

Opportunity for Caching: Another opportunity that brings scheduling and resource mappings together stems from opportunities to cache snapshots in between restarts. Most sites limit job durations to a couple of hours only. The I/O node ranks are determined at startup time, but currently, neither ICON nor the batch scheduling system (here Slurm) take the system topology nor the workflow into account. This is problematic because:

- As ICON defaults to using the last n ranks for I/O, all I/O PEs can end up being on the same compute node with performance implications for the following situations:
  - As timestep data for a variable is gathered on the I/O node, contention can occur, although transfer scheduling can attempt to avoid this when remote memory access is used.
  - I/O performance can become restricted by the uplink bandwidth of this node when draining to the storage system.
- For improved fault tolerance the simulation is using a checkpoint/restart mechanism, which submits a new job to the batch scheduler. Typically, the new job will be started with a different node list, therefor waiving opportunities offered for example by client-side PFS/Lustre caches.

There are a number of possible interventions to improve the I/O behavior of this application, some of which can be performed transparently to the application. This is often important because changing the application itself is often not easily possible.

Optimizing Node Placement using Topology Knowledge: Topology knowledge can be obtained from a number of different sources such as Slurm's `/topology.conf` or the `ibdiscover` command, as discussed in Chapter 5 (Information Sources and Decision Points). More useful would be a normalized approach that would subscribe to a topic in the message-oriented middleware introduced in Section 6.5 (Decision Aid Propagation Service). *Decision points* to intervene with the default behavior can be found at multiple levels. Taking advantage of options provided by the batch scheduler, such as Slurm, allows to influence task distribution down to a granularity of pinning processes to individual CPUs. At the MPI runtime level host- and rank files can be provided directly, compare Listing 7.1.

Listing 7.1: OpenMPI for example offers the concept of rankfiles which allow to influence the mapping. `https://www.open-mpi.org/doc/v2.0/man1/mpirun.1.php`

```
$ cat rankfile
rank 0=hosta slot=1:0-2
rank 1=hostb slot=0:0,1
rank 2=hostc slot=1-2

$ mpirun -H hosta,hostb,hostc,hostd -rf rankfile ./hellompi
Rank 0 runs on node hosta, bound to logical socket 1, cores 0-2.
Rank 1 runs on node hostb, bound to logical socket 0, cores 0 and 1.
Rank 2 runs on node hostc, bound to logical cores 1 and 2.
```

Finally, a configuration file or a routine in the application itself can be in charge of PE placement across the allocation. Limiting the discussion to the user level, each of the approaches can delegate the actual decision to an external decision component. Slurm, MPI, and application configurations are easily realized by querying a decision service in the job script and adapt configuration files or the environment accordingly. Alternatively, each runtime might use the library interface to issue queries.

7.2.3 *Data Locality within and Across Allocations*

The discussion of ICON demonstrated that even though two-phase I/O is used to optimize performance, there is a number of pitfalls when not taking the network topology into account. Two related optimization opportunities are the exploitation of low-hop communication and the usage of node-local storage and caches. The first part of the discussion here focuses on verifying that these opportunities in fact are present also on the Mistral supercomputer and then goes on to construct a consumable decision aid from topology information exposed by Slurm.

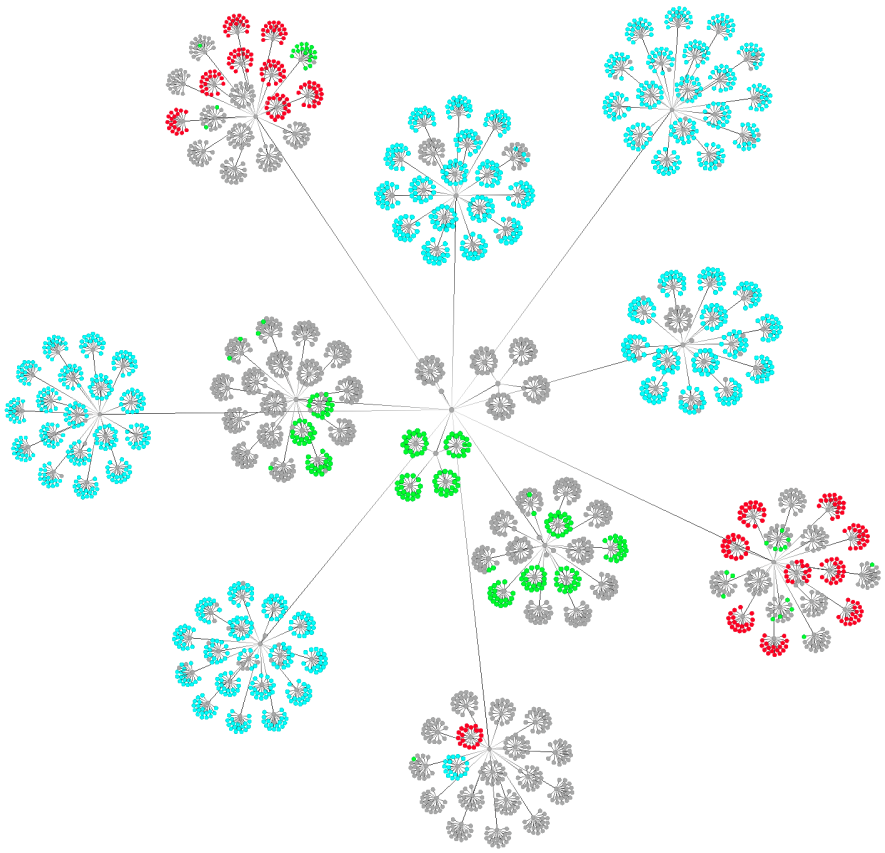


Figure 7.11: Sub-optimal job allocation of several large jobs resulting in unfavorable communication properties between certain groups of nodes belonging to the same allocation (colored nodes).

Placement for Communication or Two-Phase I/O: Figure 7.11 shows a topology visualization of the Mistral supercomputer and colors multiple large job allocations, which exposes how the jobs compete for network resources and that often groups of nodes can be spaced far apart. The visualization demonstrates that as an artifact of the underlying job scheduling strategy, applications routinely have to accept suboptimal node allocations. From an I/O perspective, this can lead to different latency and throughput domains within an allocation as further discussed in Section 7.4 (Component Assessment: Decision Aid Propagation Service).

Placement for Caching: With checkpoint-restart workflows it often becomes necessary to submit multiple jobs as many sites impose maximum job lengths. On Mistral, for example, jobs typically are not allowed to exceed 8 hours of runtime. Long-running climate simulations are thus split into multiple shorter jobs. From a user perspective, this is fully automated so that a single submitted job is allowed to submit follow-up jobs. Unfortunately, in many cases, the batch scheduling system will be ignorant of this

task relationship. A proof-of-concept evaluation of this optimization has been conducted by other researchers [UCSC, 2017; Duro et al., 2015] for a different application, which demonstrates that taking advantage of node-local caches reduces overall workflow runtime.

7.2.4 Conclusion: ICON I/O Analysis

The analysis of the ICON climate model exposes a number of unused I/O optimization opportunities but also shows that sophisticated special purpose tools are often required to understand an application's behavior.
From a workflow perspective, the interaction with pre- and post-processing tools give rise to multiple storage-related decisions such as serialization, data placement, staging and caching decisions. Some of these optimizations are best implemented in coordination with the site or workflow scheduling.
A fine-grained analysis of the files that are being accessed reveals a number of interesting access patterns. In particular, by visualizing DXT trace data I/O behavior can be quickly assessed without having to audit a large codebase. Using the tools for trace visualization developed for this research, the I/O behavior for input, output, and diagnostic files can be quickly evaluated. For structured input data collective I/O successfully leverages also page caches of the operating system. Output files feature a distinct access pattern that shares similarities invariant to file size. By generating a temporal timeline of file accesses and I/O phases of the climate application can be revealed.
By overlaying I/O issuing nodes onto the system topology a performance bottleneck in the used configuration is revealed which can be resolved by ensuring I/O PEs are not contending for bandwidth of the same network link. ICON runs perform checkpoint-restart with jobs typically running no more than 8 hours (site time limit). Typically, the follow-up job continues the simulation on a different set of nodes leaving data already residing in node memory unused.
Many of the optimizations discussed here are best addressed within privileged services such as the batch scheduler or the storage system. A promising candidate for optimizations which can be realized at the workflow or application-level exists at the task boundaries of the workflow: Deciding on the intermediate storage layout for data that is passed between producers and consumers. This use case is addressed in the concluding proof-of-concept discussed in Section 7.5 (End-to-end Proof of Concept).

7.3 Component Assessments: Decision Aids

This section demonstrates and assesses the value-added and the process to generate decision aids. To inform, for example, the decision of picking a storage layout from a workflow perspective a wide variety of information has to be taken into account, which is why the concept of decision aids which are derived from workflow artifacts was introduced. This section performs multiple assessments of generating different decision aids identified in the previous sections and which are used in simplified forms to enable the decision necessary for Section 7.5 (End-to-end Proof of Concept):

- Without some knowledge about the I/O between tasks no workflow-aware storage decision can be performed. The discussion therefore starts

by augmenting the workflow task-data dependency graph with information about the I/O behavior in Section 7.3.1 (Decision Aid: Augmenting Workflow Graphs with I/O Behavior). From here, other more specific decision aids can be derived for more advanced recommendations.

- As an example, the second decision aid highlights how to train feature detectors and classifiers to identify patterns in applications or workflow I/O behavior in Section 7.3.2 (Decision Aid: Complex I/O Phase Characterization and Segmentation).
- Finally, a third decision aid is generated using clustering methods to group fingerprints obtained by normalizing trace data to find similarities without requiring additional supervision in Section 7.3.3 (Decision Aid: File Type Clustering and Classification).

7.3.1 Decision Aid: Augmenting Workflow Graphs with I/O Behavior

When attempting to optimize with a workflow perspective in mind, the first problem to encounter is that an overview perspective that provides the task and data relationships as well as potential I/O implications is often missing. In this section, the effectiveness of a pipeline to generate such an I/O perspective for workflows is demonstrated. A more in-depth description of the approach is also published separately [Lüttgau et al., 2018b]. As discussed in Chapter 3 (Scientific Workflows in High Performance Computing), current WMS typically do not try to include the storage perspective, even less so one which is directly consumable by a researcher or a decision component to base an optimization on.

Lüttgau, J., Snyder, S., Carns, P., Wozniak, J. M., Kunkel, J., and Ludwig, T. (2018b). Toward Understanding I/O Behavior in HPC Workflows. In *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, Dallas, TX, USA. IEEE. `https://ieeexplore.ieee.org/document/8638425/` (accessed 2019-04-23)

The desired information for this initial decision aid follows the stripped-down structure in Listing 7.2. Most notably the decision aid has to provide the workflow graph to inquire about relationships. Reports, files, and other artifacts are indexed independently because it is often not appropriate to attach them to a single node in the graph. Finally, subsequent decision aids as discussed in the following sections might provide summaries or recommendations which can be attached or referenced in the annotation section. Instead of inlining this information, the actual reports typically reference data to allow loading actually required information on demand.

```
{
   graph: {
      nodes: [{type: "task", 'id: "<uuid>", ...}, {type: "file", ...}],
      edges: [{src: "<uuid>", tgt: "<uuid>", ...}, ...], # tasks/data dependencies
   },
   reports: {
      tasks: [...],    # task reports: e.g., summarized or as references
      files: [...],    # file reports
      ...
   },
   annotations: [...]   # generated advice by different decision aids
}
```

Listing 7.2: Annotated structure of the output produced workflow graph decision aid for consumption by visualization tools and other decision aids.

To generate and populate this structure it is necessary to extract a dependency graph of the workflow. This could be derived from source code, via an export feature to dot, or generated from an execution log. As different workflow engines require different strategies, helper scripts for each workflow engine map the native task/data model to the task/data abstraction used by *darshan-workflow* post-processing tools. The process to derive this decision aid is outlined in Figure 7.12 and consists of the following sequence of steps:

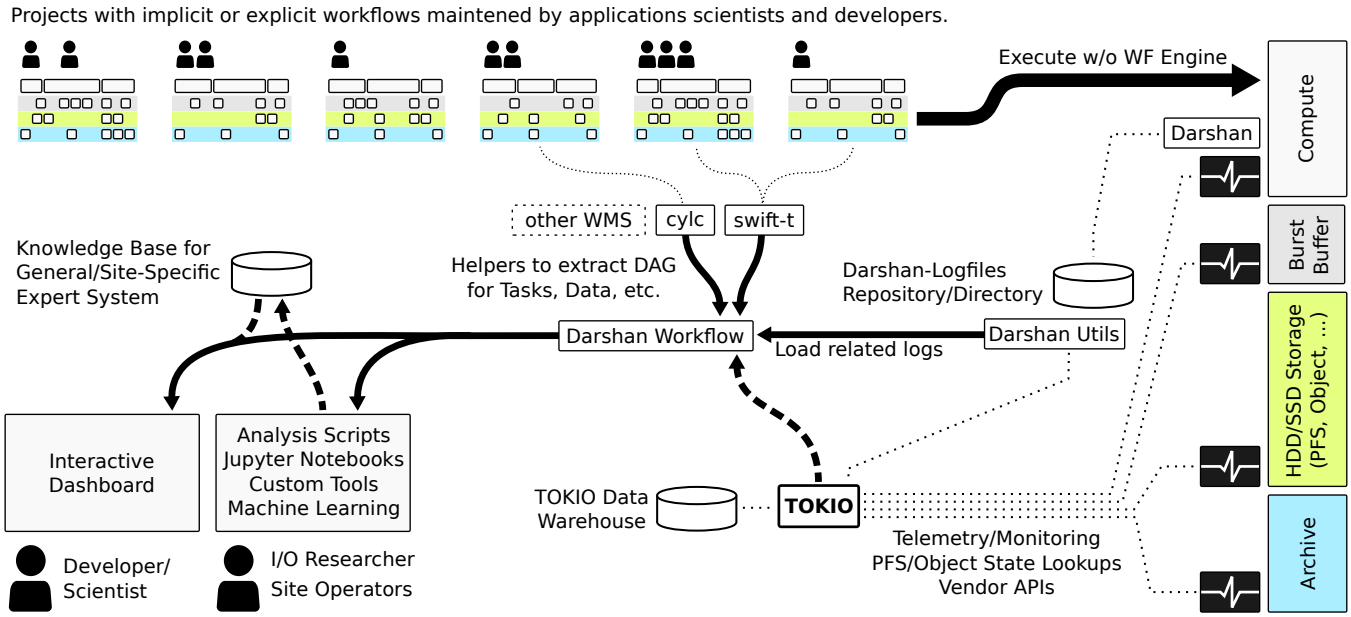
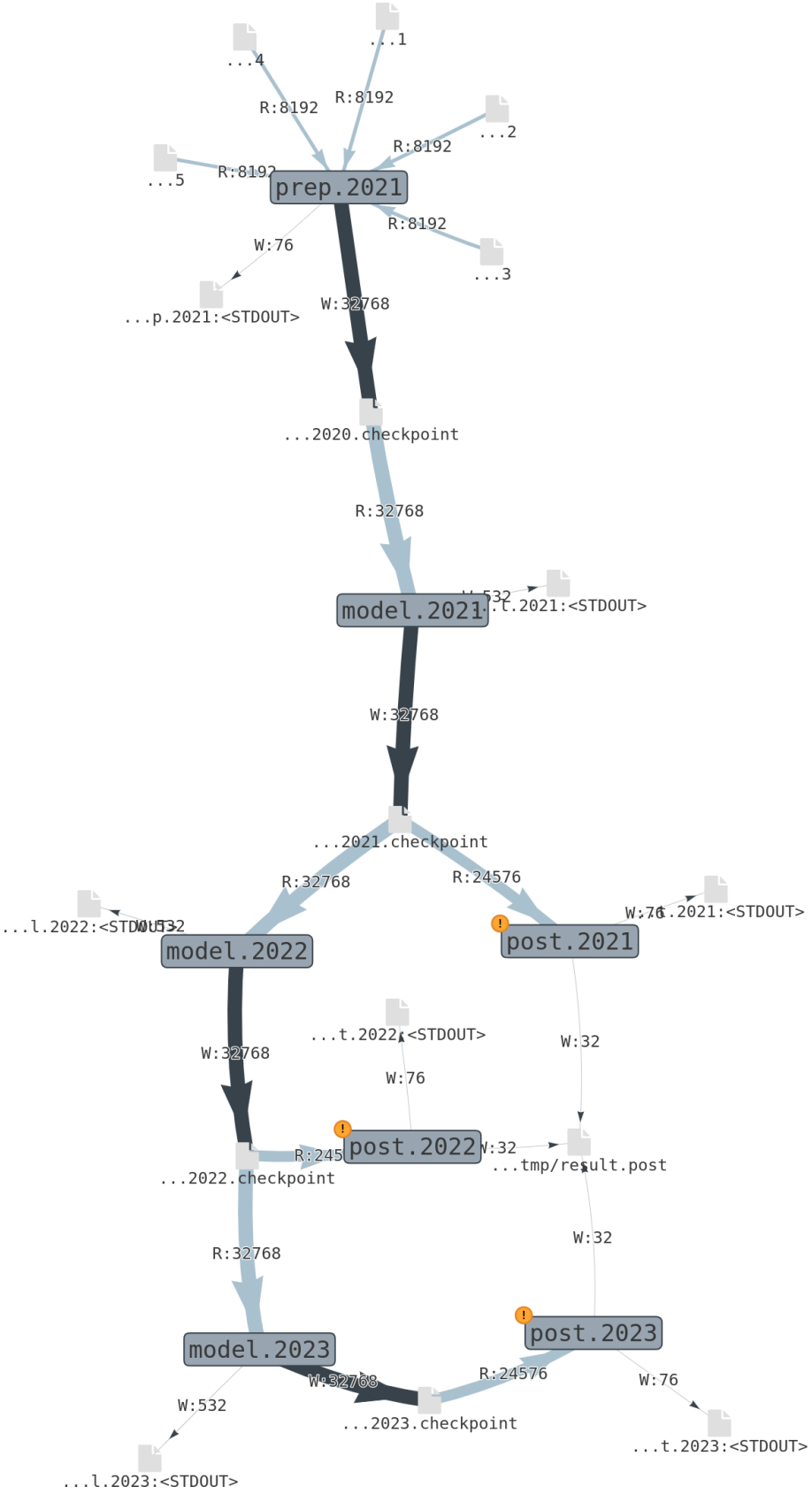


Figure 7.12: Architecture overview to augment I/O behavior in HPC workflows by pulling in information from workflow engines and I/O activity capturing tools. Integration with TOKIO and expert systems is not yet implemented in the proof of concept as indicated by the dashed arrows. Thin dotted lines represent (optional) information sources.

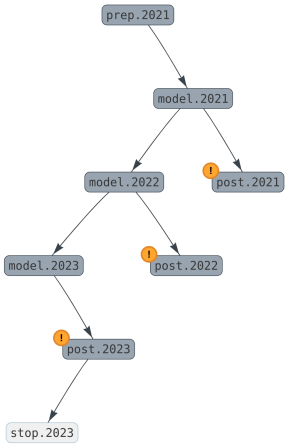
1. Provided a workflow was defined using a WMS (here either Cylc, Swift, COMPSS or Decaf) perform the following:
   (a) Enable instrumentation with Darshan (this can usually be achieved by setting and exporting `LD_PRELOAD`).
   (b) Configure the workflow engine to expose a TaskID within the execution environment. A modified variant of Darshan uses this information to add metadata to Darshan logs or to use in the log file name.
2. For every task, discover Darshan logs (currently by scanning the file system). Ideally, the lookup could also use a database or index files that were populated directly by Darshan or the WMS. Currently, Darshan logs are mainly generated by MPI applications (specifically, Darshan until recently relied on the application to call `MPI_Init` and `MPI_Finalize`).
3. Preprocess Darshan logs, and convert them into JSON representation or Python data objects for further analysis. Preprocessing will typically generate a number of derived performance counters (per file, per operation).
4. Generate a workflow report file, including the data or references for preprocessed reports.
5. (Optional) Add additional advisory products manually by review (human in the loop) or by using machine learning for automatic analysis (can be already added and rendered with interactive tools).
6. Interact with data using Python or Jupyter Notebooks, the workflow engines or custom report dashboards.

Figure 7.14a illustrates a dependency view, resembling the graph representation most WMS would offer to users. In Figure 7.13a a dataflow or I/O activity perspective is rendered that uses tasks from the workflow description and adds files and read/write access information obtained from Darshan log records. The edge label and the width of an edge indicate the number of total bytes accessed in absolute and relative terms.



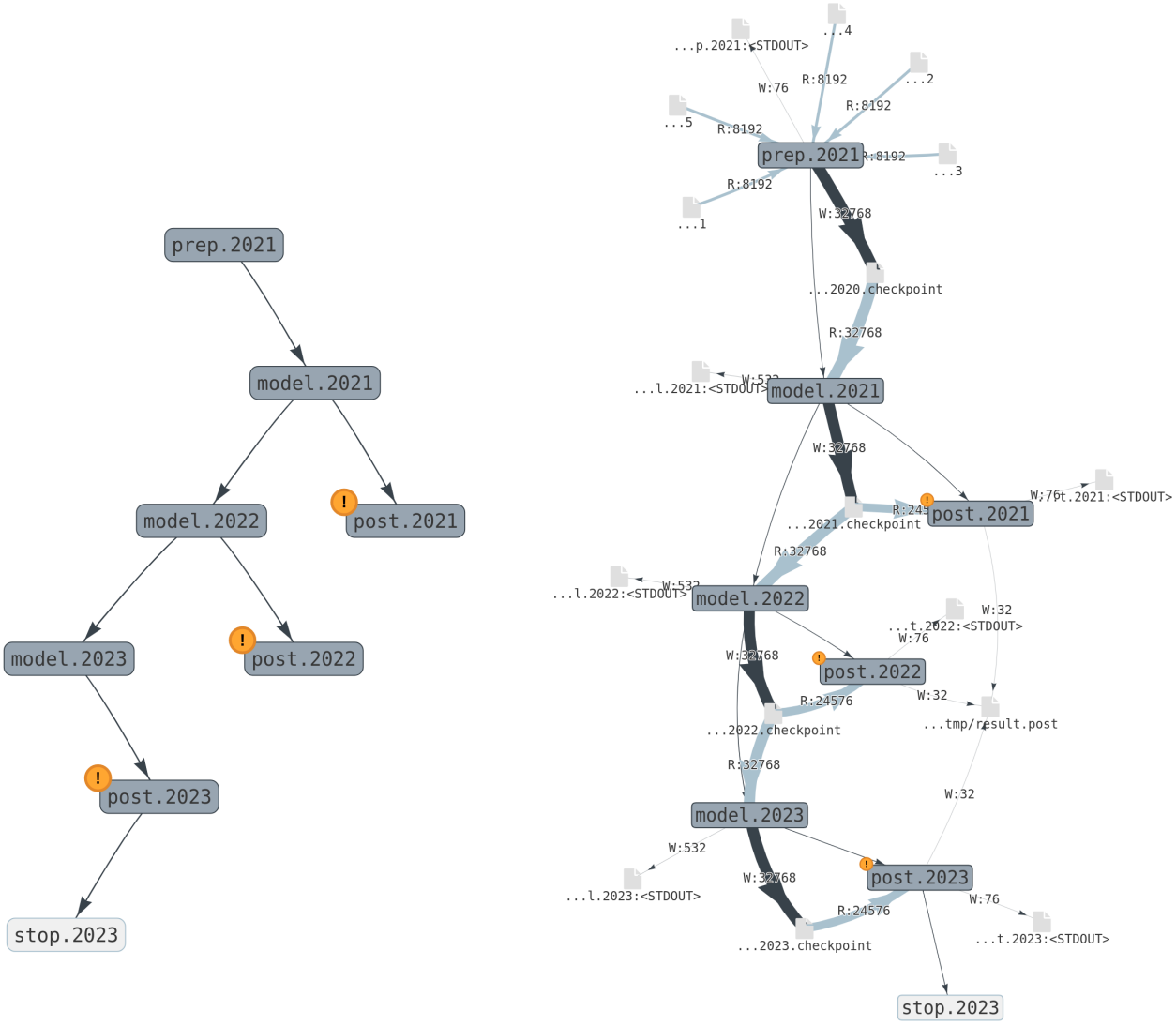
(a) Dataflow perspective reconstructed from Darshan logs. No task dependency information from the workflow engine were used to derive this task-data dependency graph.

Figure 7.13: Automatically generated graph visualizations of the workflow graph, here show a dataflow view which which includes many of the I/O side-effects of executing a workflow which remain unknown to a workflow engine. Figure: [Lüttgau et al., 2018b].



A larger variant of the small task dependency graph can be viewed on the next page in Figure 7.14a. This small graph shows the same workflow, but from the perspective using only the information available to the workflow engine.

Mixing both perspectives as shown in Figure 7.14c can be useful, although challenges exist in automatically choosing which information to display. The graph visualization is implemented with vis.js [Almende B.V., 2018] as a foundation, to allow interaction as well as different perspectives and different levels of detail. JavasScript packaging allows for easy reuse in various contexts and custom tools such as Jupyter Notebooks and dashboards.



(a) Tas dependency perspective as typically defined for execuation by workflow management system or seen by the user.

(c) Adding all or additional information as in this hybrid perspective suggests that mechanisms to filter information are needed.

Figure 7.14: Automatically generated graph visualizations of the workflow graph taking different perspectives based on different information such as only these declared to the workflow engine, or only task identifiers combined with telemetry information. [Lüttgau et al., 2018b].

Also, take note of the orange exclamation marks rendered near the top-left corner of the post-processing tasks. Indicators like this, for example, to indicate usage of MPI-IO, POSIX, or STDIO) can be used to communicate problems and other information in a compact and intuitive representation. Currently, an indicator is added when an annotation for a node/task is defined in the JSON report, which in this case was performed by a script that applied a machine learning classifier to every task.

Nested and In Situ Workflows: In a second assessment, the approach is extended beyond a single workflow engine to in situ and nested workflows. A discussion of these kinds of workflows is important because many workflows are heterogeneous and in situ workflows are becoming increasingly relevant to bypass storage systems for temporary data [Deelman et al., 2018; Peterka et al., 2019].
A particular challenge for the analysis of I/O behavior in nested and in situ workflows is the diversity of workflow artifacts that have to be combined. Often these artifacts are exposed directly at the moment but require a variety of specialized parsers and an intermediate abstraction to combine artifacts from otherwise unrelated services and workflow engines.

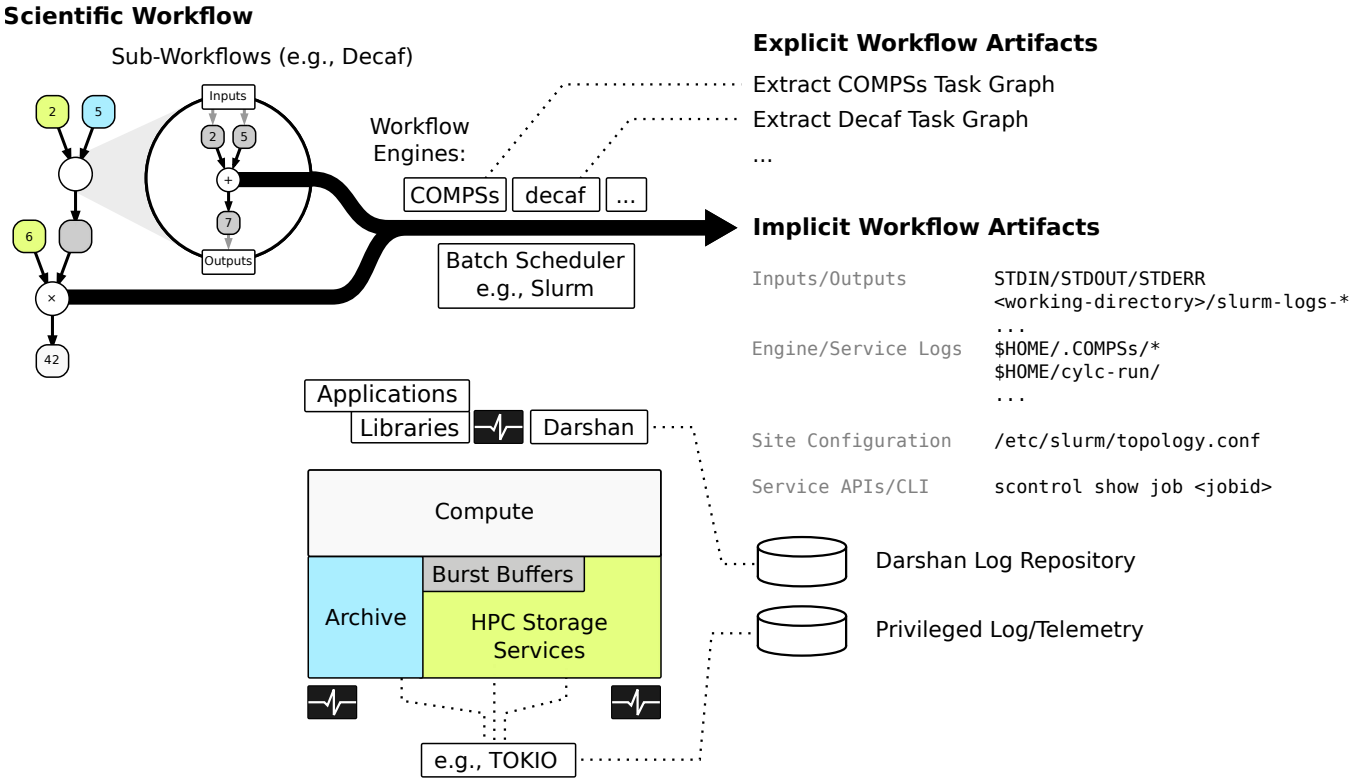


Figure 7.15: Overview of different workflow artifacts which need to be taken into consideration when dealing with in situ and nested workflows.

Figure 7.15 illustrates this complexity by relating the workflow to used services and to implicitly generated artifacts. Starting from the left, a simple workflow is depicted which nests a sub-workflow in one of its tasks. In addition, on an HPC system, the main workflow is typically submitted to a batch scheduling system such as Slurm which might offer basic functionality for dependency management too. Already in this simple scenario, at least three types of workflow management layers are involved.
Taking the batch scheduler or resource manager into account is also necessary because the topological relationship of physical components has significant performance implications. Most importantly, information about the allocated resources inform decisions that optimize data locality, but they also become important to analyze data flow behavior after an execution. It is often necessary to combine instrumentation collected at multiple levels, such as close to the application as well as on service nodes, such as storage nodes, to reconstruct the I/O activity of a workflow.

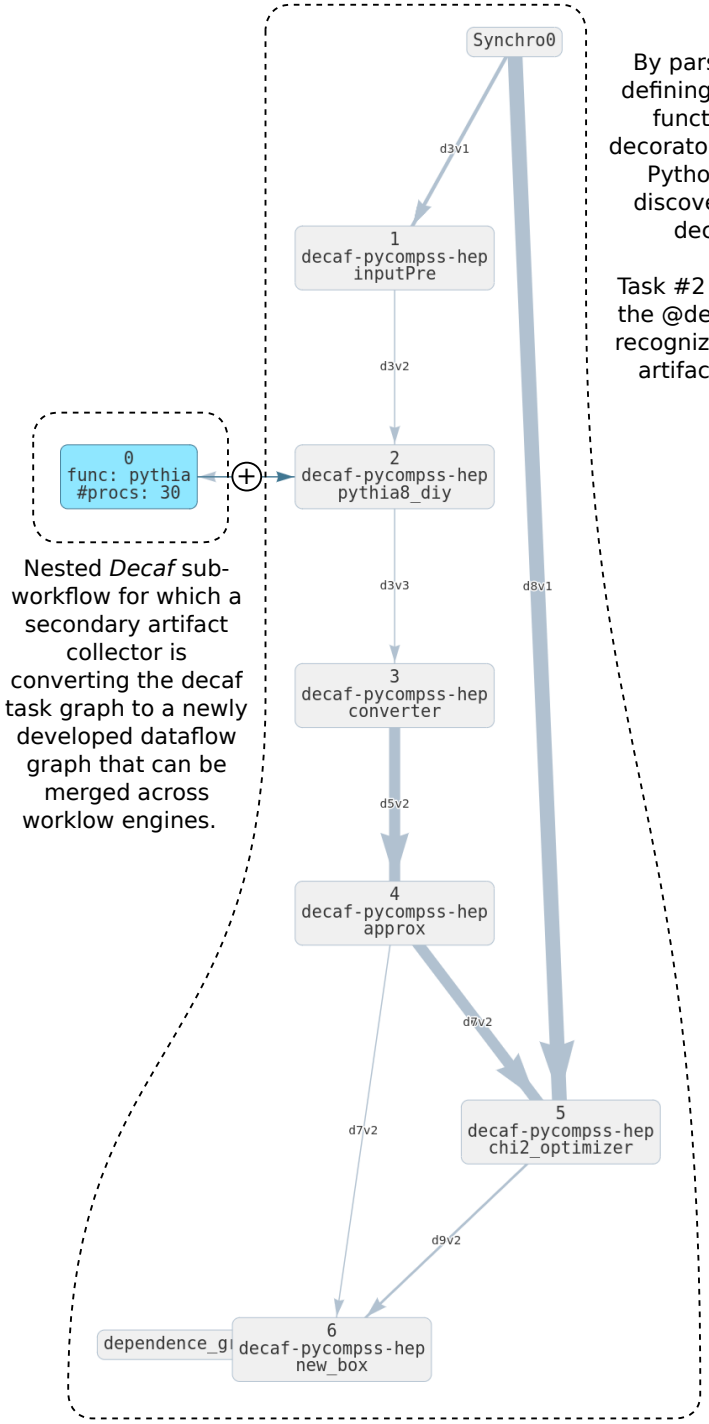


Figure 7.16: Illustration of in situ workflows managed by Decaf (blue/green) nested into a high-level workflow managed by PyCOMPSs (gray). `darshan-workflow-collector` extracts workflow graphs for both and then combines them.

A cycling High Energy Physics (HEP) Workflow: This case study analyzes an in situ scientific workflow to automatically simulate millions of concurrent Monte Carlo proton-proton collisions, search the response surface for minimum statistical difference with experimentally observed collisions, and advance toward new regions of the parameter space to investigate [Yildiz et al., 2019]. The workflow is nesting PyCOMPSs and Decaf, and the workflow tasks are a combination of C++, ROOT, and Python. Figure 7.16 shows the combined dependency graph with components of the Decaf workflow in green/blue and the high-level PyCOMPSs workflow in gray. The workflow initializes a simulation, performs multiple post-processing steps, determines update parameters, and repeats the cycle, which in COMPSs is expressed through so-called synchronization tasks.

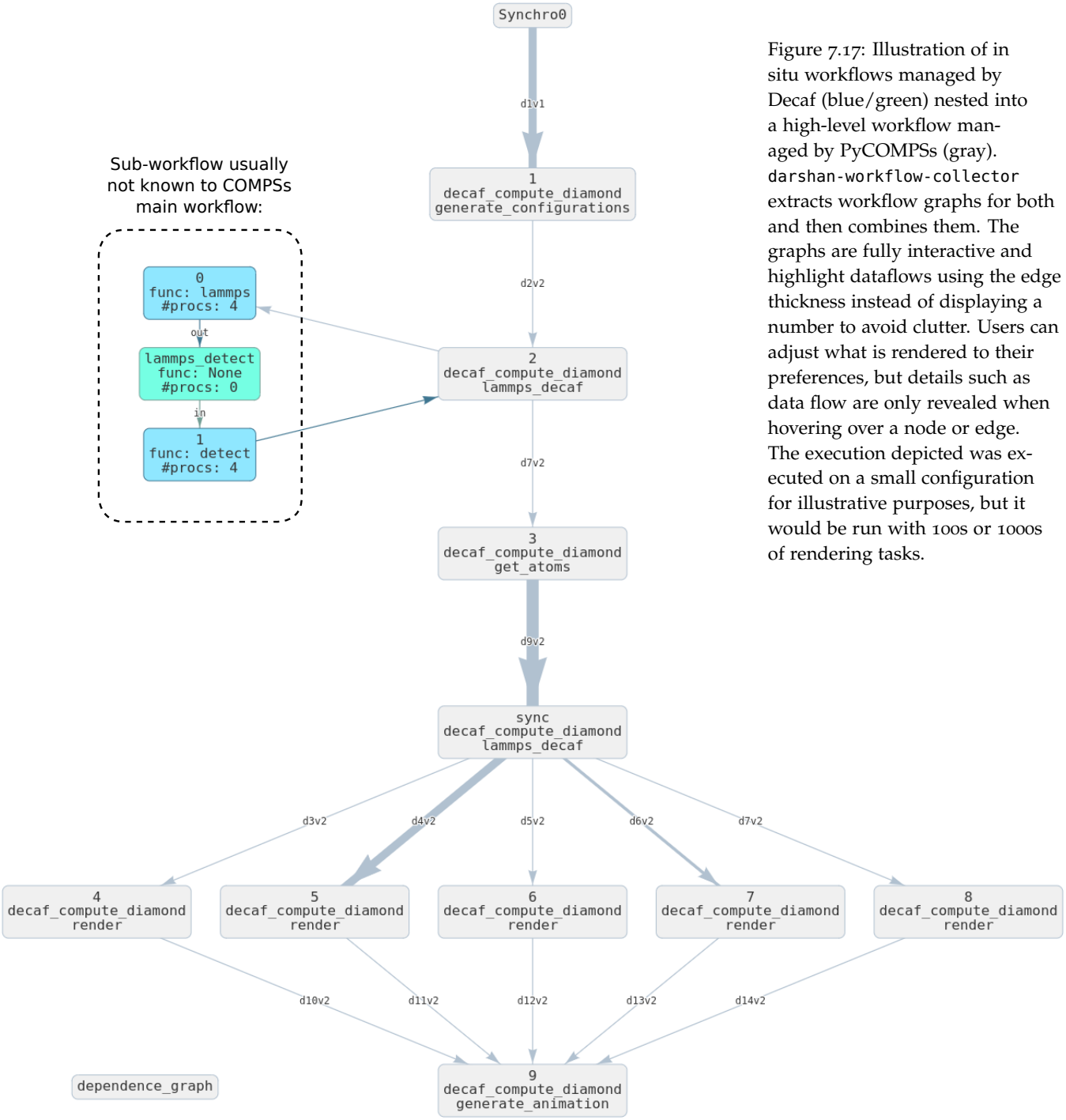


Figure 7.17: Illustration of in situ workflows managed by Decaf (blue/green) nested into a high-level workflow managed by PyCOMPSs (gray). `darshan-workflow-collector` extracts workflow graphs for both and then combines them. The graphs are fully interactive and highlight dataflows using the edge thickness instead of displaying a number to avoid clutter. Users can adjust what is rendered to their preferences, but details such as data flow are only revealed when hovering over a node or edge. The execution depicted was executed on a small configuration for illustrative purposes, but it would be run with 100s or 1000s of rendering tasks.

A nested Molucular Dynamics (MD) Workflow: This use case considers a more complex nested workflow from molecular dynamics. In this workflow an in situ LAMMPS [Plimpton, 1995] simulation is coupled with visualization in Ovito [Stukowski, 2009]. The workflow implemented by nesting Decaf and COMPSs with the goal to allow computational steering when studying nucleation. After an initial configuration, the LAMMPS task simulates the freezing of substrate until nucleation occurs. Since this is a probabilistic process, different pipelines can behave differently: If more atoms nucleated in a simulation more data is passed for rendering. This behavior is also observable in the data-flow graph in Figure 7.17 , which shows the combined dependency graphs of the Decaf workflow in blue/green and the high-level COMPSs workflow in gray.

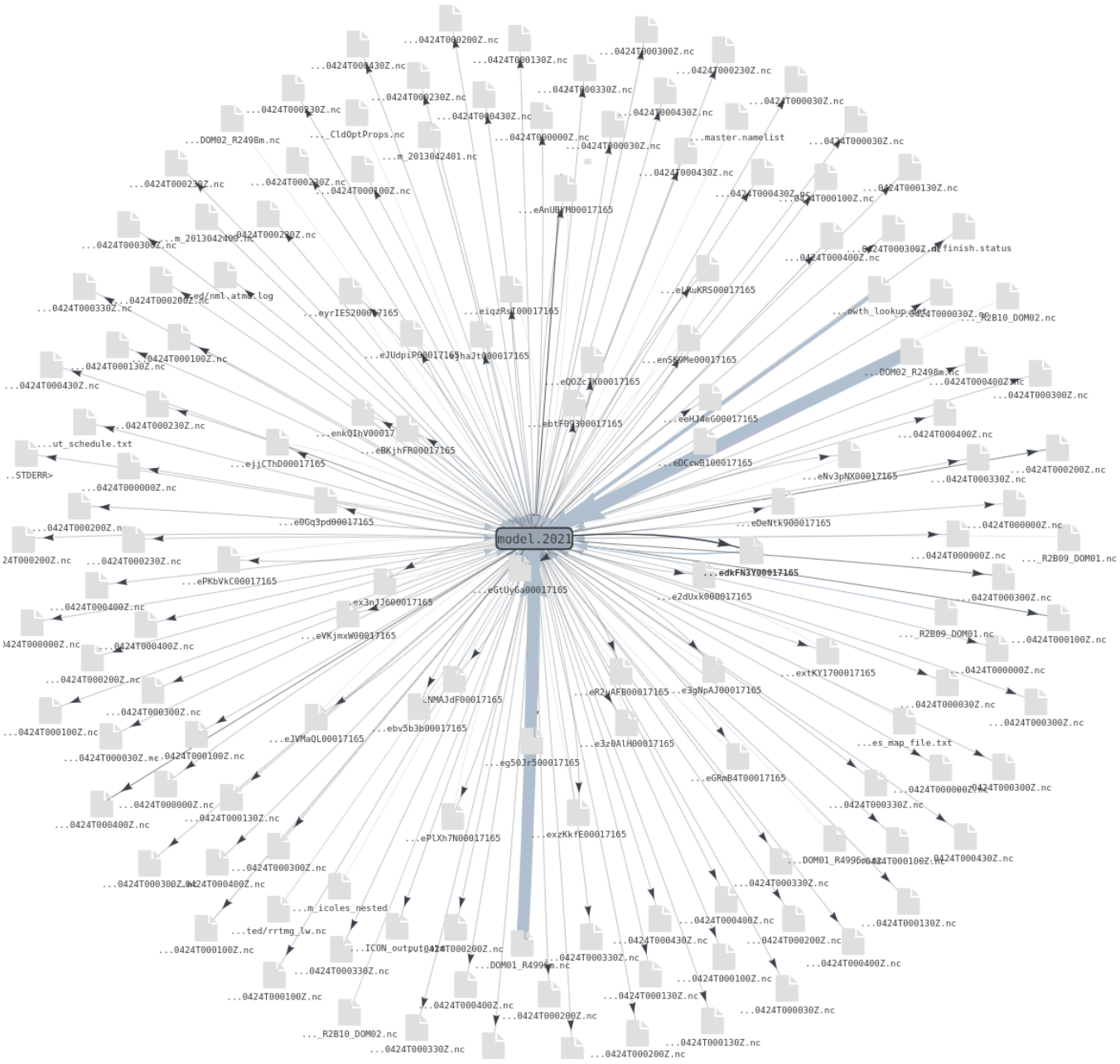
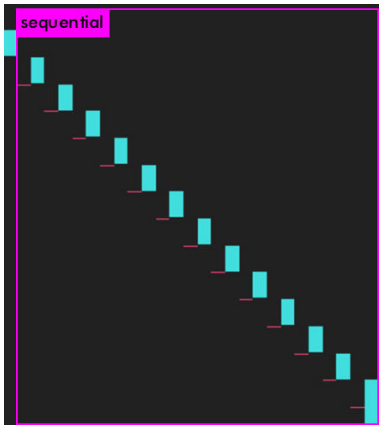


Figure 7.18: Figure illustrating how also visualization tools require filtering and pre-processing: While more accessible than the raw telemetry in text form navigating this also becomes overwhelming. Visualized here are all files touched by ICON, a climate model code used by researchers by the Max-Planck-Institut for Meteorology and the German Weather Service (DWD).

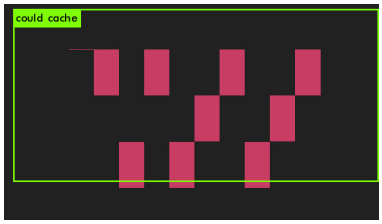
Conclusion for Augmenting Workflow Graphs with I/O Behavior: The approach demonstrates that it is possible to reconstruct the original workflow graph for some workflows by just processing I/O profiling information obtained using Darshan. Because this decision aid uses a JSON representation it is possible to augment this decision aid or reference additional insight from subsequent decision aids. The case study also revealed a number of challenges, namely, that of information overload which may occur if not using strategies to reduce and aggregate data beforehand. That tools to support such filtering operations are necessary is demonstrated when visualizing all accessed files even for a very short ICON run, as shown in the dataflow graph in Figure 7.18. The task in the center is the climate model, the exact filenames names are not important to portray the file I/O activity: The model mainly reads from two files (large gray arrows inwards), but writes to hundreds of NetCDF files. In the following two machine learning approaches discussed which can be used to further augment or filter less interesting interactions from such overwhelming graphs.

7.3.2 *Decision Aid: Complex I/O Phase Characterization and Segmentation*

The relationships of telemetry information combined with the workflow data-dependency graph can often be complex, with recommendations depending on the task or even the context of a task within a workflow. It is this context-sensitivity which makes it hard to program systems to take advantage of the many different possible opportunities for workflow I/O optimization discussed earlier. A common scheme in learning and decision-making is thus to break down a very challenging decision or recognition task into smaller easier problems. Often this involves feature extraction out of a stream of raw input data.
This section looks at an approach developed together with a student to extract such features from trace data, obtained using Darshan's extended tracing (DXT) capabilities [Frank Röder, 2019]. A slightly indirect but visual approach is used here, as it is easier to discuss but also offers a number of other benefits for the integration with tools to explore and discover optimization opportunities or to communicate with non-technical users. It is also straightforward to label data on top of this representation.
Figure 7.19 shows two examples for traces visualizing time on the x-axis and position in the file on the y-axis, and the colors red and cyan coding for read and write access respectively. A neural network-based approach is then used to generate bounding boxes for a number of predefined access patterns such as "sequential" or "could cache".
In an automated software stack, it often might be appropriate to deploy a more light-weight or specialized derived detector. With transfer learning in mind, it is often possible to train for one objective, but because a latent representation is learned, it is possible to reuse parts of the networks in different contexts.



(a) sequential



(b) could cache

Figure 7.19: Two examples where the detector was trained to recognize I/O behavior from an intermediate representation, accessible to both humans as well as compatible to established and well researched classification and recognition algorithms.

Training: Figure 7.20 documents the training process of a detector which can generate bounding boxes such as those illustrated in Figure 7.19. After an initialization phase for the first 250 iterations, the network converges quickly until about iteration 600 where the loss starts to stabilize.

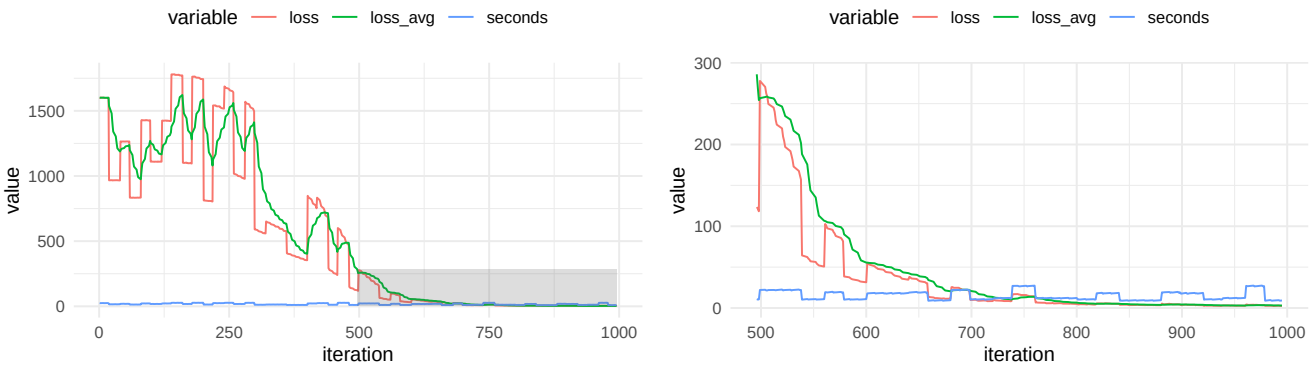


Figure 7.20: Monitoring displaying how an iterative machine learning-based detector is trained from a small set of training examples. The right plot is a zoomed in view into the area highlighted in grey of the right plot.

The detector in this case is based on the third iteration of a method designed for real-time object detection from streaming data, named YOLO [Redmon and Farhadi, 2016; 2018]. Training the network on a small corpus of data containing only about 100 labeled samples takes about 2-3 hours with two Nvidia K80 GPUs. Detection times range from 12-25 seconds when provided a single CPU core. On a GPU, detection times are much shorter within a 200-500 ms timeframe.

Conclusion: This classification experiments demonstrate that it is viable to adapt existing architectures to tasks in I/O characterization. While a more extensive study with many more motifs should be conducted to justify the complexity of the network, this preliminary evaluation can be used to establish an automated labeling helper for relatively complex patterns. For the moment it is anticipated that such decision aids are mostly used offline, but further research into reducing model complexity might allow to consider online approaches.
As it is anticipated the analysis of such features in many cases is performed offline, it is possible to batch feature extraction making it worthwhile to run this on GPUs despite short individual recognition times.

7.3.3 *Decision Aid: File Type Clustering and Classification*

Revisiting the traces for the ICON climate application from Section 7.2 (Use Case: Analysis of the ICON Climate Model), it becomes apparent that there is routinely too much information than can be reasonably visualized. Figure 7.18 for example shows a visualization of dataflows from and to files of a simulation run of the ICON climate model which easily interacts with hundreds of files. As more snapshots are written this number increases.

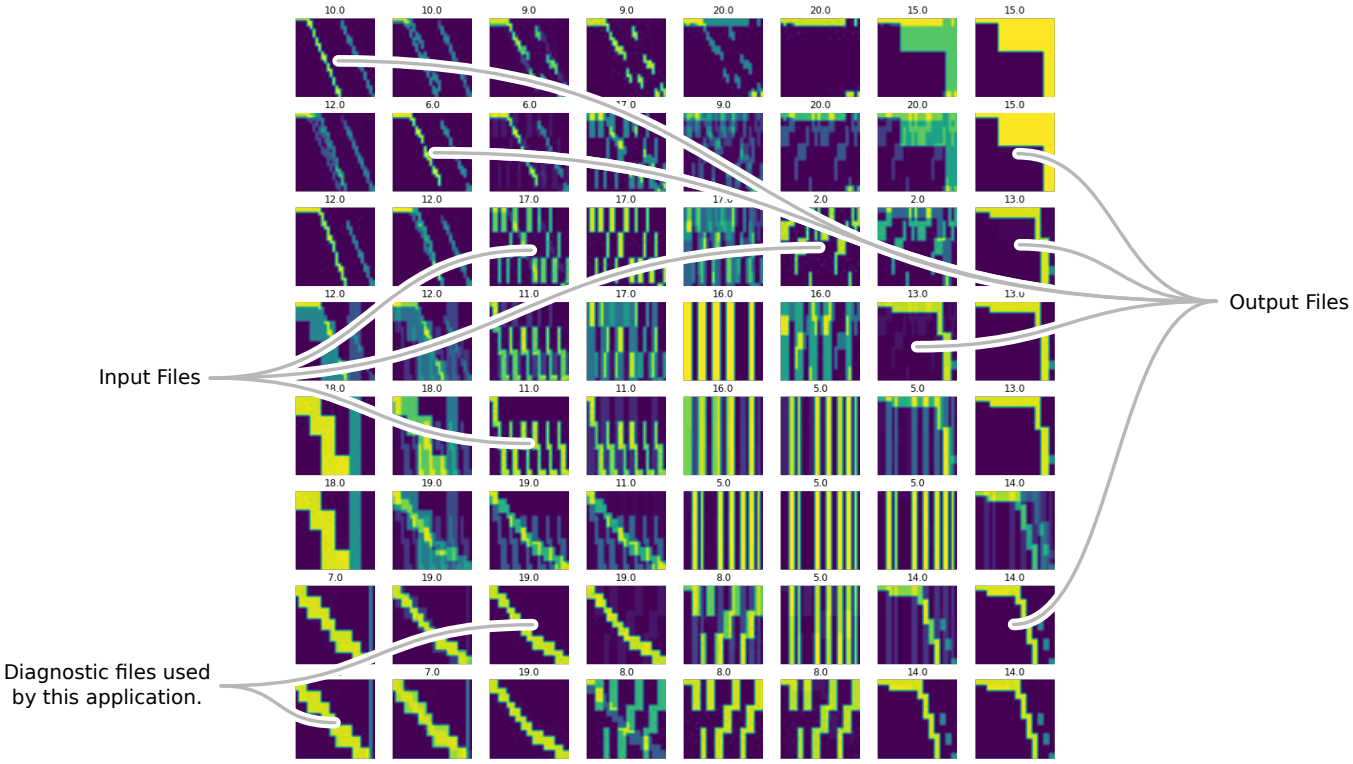


Figure 7.21: Interpolations of access fingerprints generated from a self-organizing map clustering. The numbers on top refer to the closest actual fingerprint. The arrangement demonstrates that despite different underlying file/dataset sizes access patterns for input or output files for a particular application are often quite distinct and can be grouped using unsupervised methods.

Considering all file interactions is often not necessary, so a mechanism to filter by different criteria is required. This is true for automated decision-making as well as for visualization. This use case implements a decision aid that can be used to filter and then help to classify files or objects based on a fingerprint generated from trace information. As an unsupervised learning method is used no labeled data needs to be provided.

Fingerprinting: A very naive approach to obtain and normalize fingerprints is to reuse the visual trace representations introduced earlier but scale them to have matching size. In practice, an auto-encoder based approach will of-

ten be more adequate and conservative about resources in the latent space. For the discussion here, however, using a visual representation is advantageous because it is more intuitive to reason about similarity.

Clustering: Based on these fingerprints a Self-Organizing Map (SOM) is used to obtain a clustering. SOMs are introduced in detail in Section 2.6.3 (Self-Organizing Maps). An example clustering is shown in Figure 7.21, which displays an interpolation of the closes matching fingerprints and the closest matching ID displayed on top. The SOM, in this case, was configured to use an 8x8 regular grid as a base topology.
A number of files (input, output, and diagnostic) are marked to highlight three classes to show where similar file-types were clustered together correctly even though their file size varied considerably.

Clustering Conclusion: This case study demonstrates that clustering can be viable to find and group small-scale I/O motifs on a per-file basis. Input, output, and diagnostic files are separated and grouped close to at least another representative which is also input, output, or diagnostic respectively. This clustering considered traces only collected from a single application/-workflow, but in this case, is sufficient to support a workflow optimization. It remains unclear if there is an opportunity to find transferable similarities across a large user base to train more general classifiers.
The fingerprinting mechanism used here uses trace files stored on application shutdown, but the fingerprint could also be generated on the fly, thus eliminating memory and storage overheads that currently occur.

7.4 Component Assessment: Decision Aid Propagation Service

The Decision Aid Propagation Service (DAPS) simplifies the distribution of decision aids to interested subscribers. It is responsible for the efficient and topology-aware propagation of decision aids from publishers, which generate a decision aid, to a decision component, which utilizes a decision aid. It is essential to evaluate its performance characteristics for expected message sizes in the workflow-aware storage decision support framework to confirm the feasibility of this approach and determine the frequency at which decisions can be propagated through the system.
In this assessment, DAPS utilizes ZeroMQ, a lightweight asynchronous messaging library for distributed applications, as our prototype transport for DAPS because it specifically implements communications patterns such as publish/subscribe while allowing to fine-tune and adapt for high-performance. In this section, we evaluate ZeroMQ in terms of bandwidth and latency to understand the parameter space for our framework.
The latency analysis considers cold-start times and periodic updates. It also demonstrates the existence of jitter and finds a latency spectrum in respect to the underlying topology. The message size analysis shows the relationship between message size and frequency at which a decision aid can be offered.
ZeroMQ is an open-source library for building message-oriented middleware when low-latency and concurrency are important. It supports a variety of transports, like IPC, TCP, UDP, multicast, or WebSockets but does not support MPI. While this might appear odd in the context of HPC environments, it is assumed not critical for the anticipated decision aids used with

DAPS. This benchmark verifies these assumptions and it is used to inform the implementation of DAPS, but also serves as a performance test.

Benchmark Setup: Desired are estimates approximating worst-case behavior for the latency and message size measurements. For this reason and because many providers of decision aids are anticipated to be scripted in Python, the benchmark also uses Python on top of ZeroMQ and MPI. MPI, being a common denominator in HPC stacks, is mainly used to conveniently gather all available nodes provided by an allocation across different sites and scheduling systems. MPI is also used for some coordination within this allocation, such as determining which ranks/nodes are to spawn ZeroMQ clients or servers.
The benchmark then executes different measurement kernels for point-to-point and message size measurements and collects the results as illustrated in Figure 7.22. Transfers on different nodes can be triggered this way, allowing the analysis of various propagation related system characteristics. All of the following measurements are using ZeroMQs default TCP based transport. This too is done to obtain a slightly pessimistic estimate.

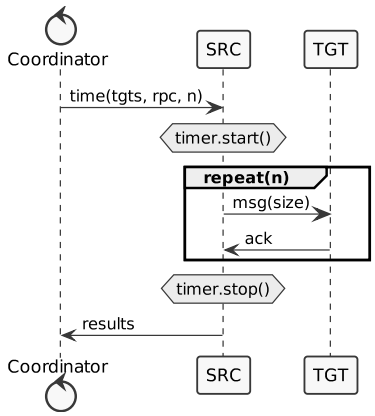


Figure 7.22: Structure of the measurements performed by the benchmark. Here a roundtrip is repeated but the experiment kernel can be swapped out for arbitrary remote-procedure calls.

7.4.1 *Latency Evaluation*

In this subsection, the results for a latency evaluation in a variety of scenarios are discussed. All measurements presented in this subsection use small message sizes, the effect of message size is evaluated separately even though latency, throughput, and operation counts per second are intimately related. The following scenarios are being considered as they determine which performance users of DAPS can expect with ZeroMQ as a transport.

1. *Cold-start:* A single small message transfer for an uninitialized instance ($n = 1$) of DAPS is measured. This scenario informs cold start latencies and is especially relevant for DAs which are static within an allocation. Figure 7.23 shows the cumulative distribution of latencies on a log scale on the x-axis for different point-to-point communications within a 20 node allocation using different repetition counts. The view is split into multiple facets with the number of repetitions used noted on the right side of each plot. For the cold-start scenario, only the first facet is relevant which shows that most messages can be expected to be received within 3-4 ms. As more communication takes place the average message latency approaches 0.2 ms.

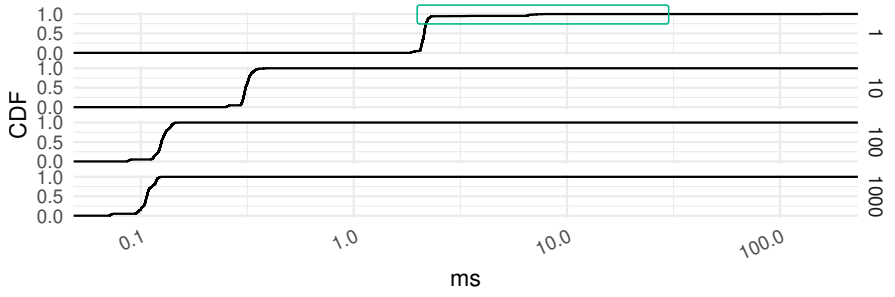


Figure 7.23: Cummulative distribution of latencies for the cold-start scenario: A small message is send n times (right labels in plot) directly after starting up.

2. *Periodic:* A small message is repeatedly ($n = 1, 10, 100, 1000$ over a period of 10 minutes) transferred and measurements are plotted over time. This scenario is relevant for dynamic DAs which repeatedly update recommendations within the execution context.

In comparison to the cold-start scenario the distribution appears to be stable also when running the same benchmark periodically multiple times as plotted in Figure 7.24. A small difference appears for single point-to-point messages which is missing the nodge between the 3.0 ms and 7.0 ms marks. The experiment shows, that for the moment there is no need to optimize the cold-start case as it does not appear to impose a significant performance panelty.

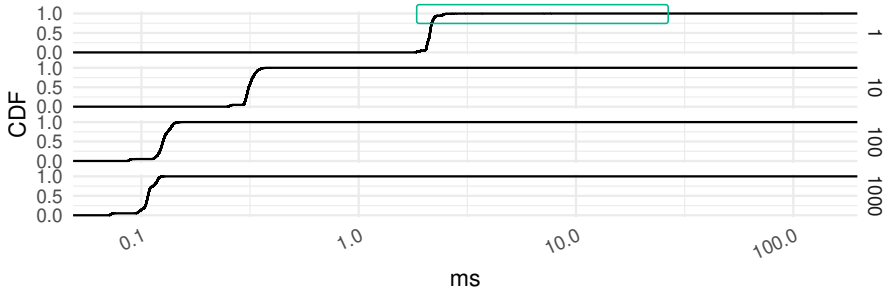
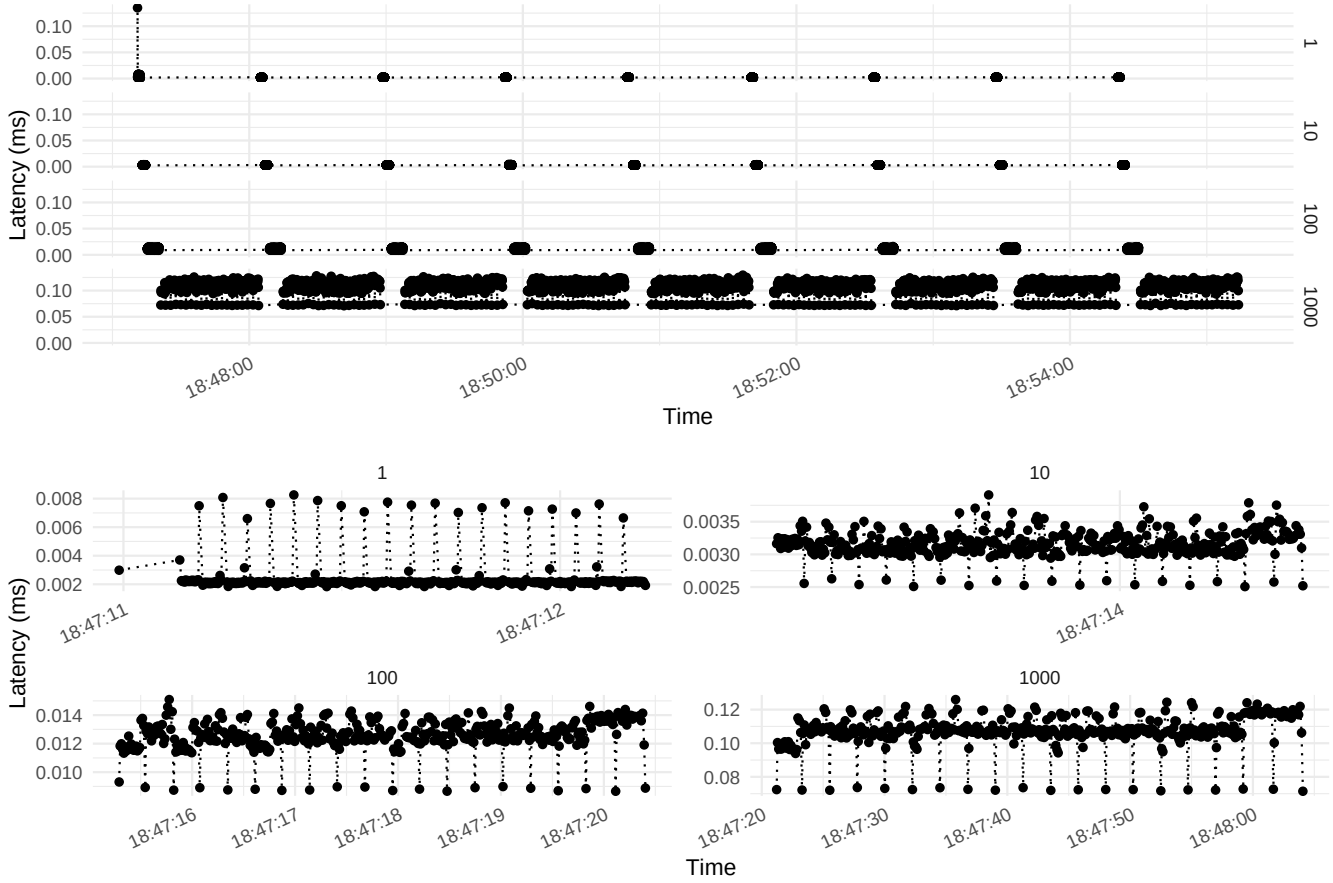


Figure 7.24: Cummulative distribution of latencies for the cold-start scenario: A small message is periodically send n times (right labels in plot) to different targets over the course of 10 minutes.

Figure 7.25 shows multiple zoomed-in portions of the periodic measurement scenario as time-series plots. Again, the observations are faceted by message repetition counts which helps to explain the similar behavior for both the cold-start and the periodic message scenarios.



Especially apparent are the effects of buffering for 1000 messages which also suggest that the buffer size slowly increases (stair steps). While this increases overall throughput, it increases latency for individual messages. A similar pattern emerges for the 100 messages case, but only barely for the 10 message case. An unexpected outlier is the second message for the single point-to-point exchange case which persisted across different runs.

Figure 7.25: Different zoomed in views as timeseries plots for the periodic scenario. The bottom plots are all showing the first iteration for the 1, 10, 100 and 1000 exchanges cases. The outlier for the single point-to-point exchange was removed for the plot to scale.

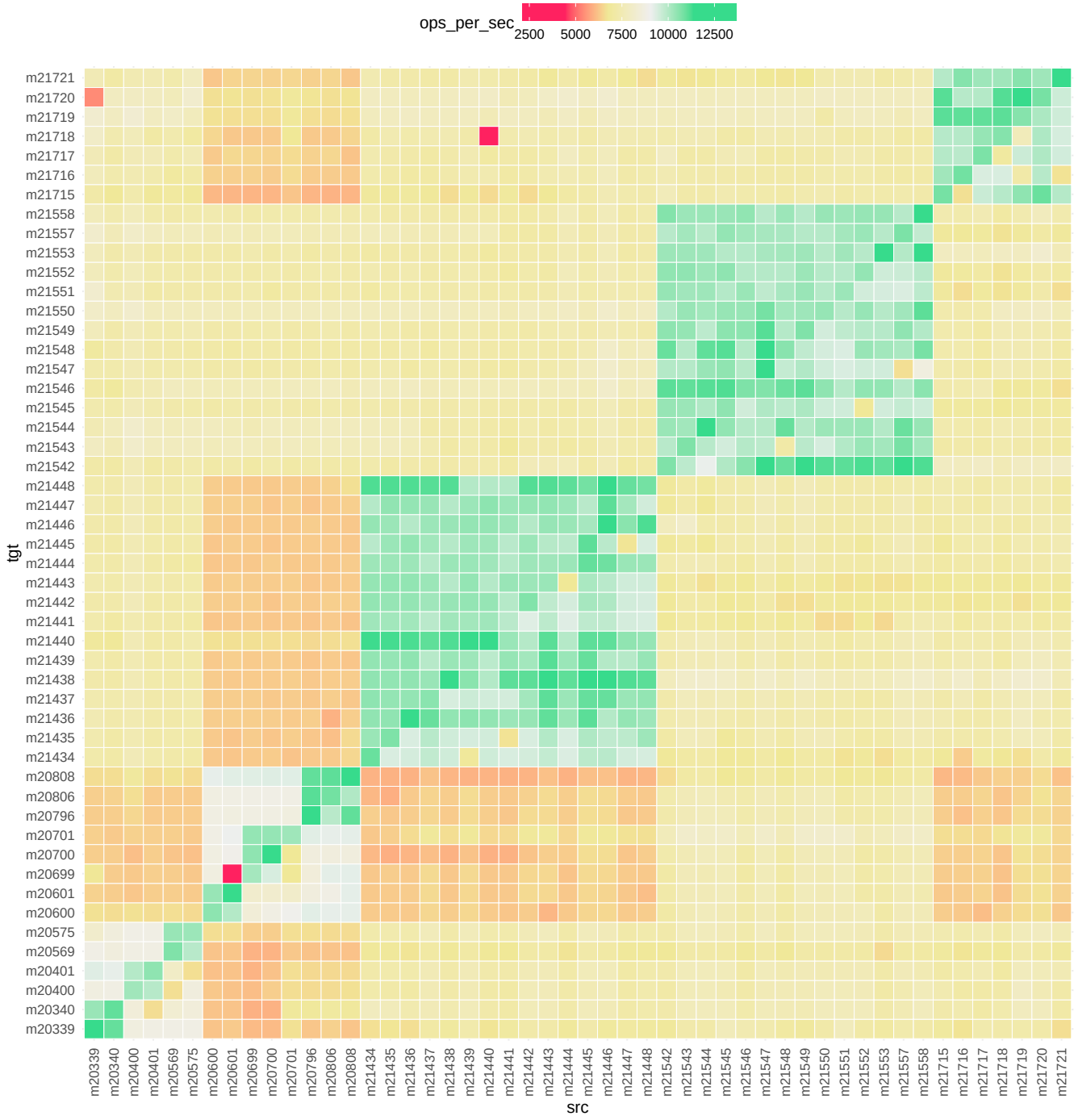


Figure 7.26: Point-to-point communication benchmark using ZeroMQ as a transport. This latency matrix, on purpose does not average multiple requests to highlight performance fluctuations existent also within groups of closely connected nodes.

3. *Topology:* Latency is profoundly impacted by the network topology. To quantify the effect imposed by the topology of the network, multiple small point-to-point communications are measured and then visualized in two different ways: As a point-to-point communication matrix and as a topology overlay which considers the actual network of connections.

   The analysis shows that especially for DAs within larger allocations there might be the need to consider hierarchic decision coordination.

Point-to-point Communication Matrix: Besides considering average performance for propagation times, it is useful to consider point-to-point latency behavior. Figure 7.26 display the point-to-point measurement matrix between 50 nodes. Color-coded is the performance, where greener is better.

The matrix clearly shows that various nodes can be grouped into nodes with particular low latency between each other. This is an artifact of the underlying network topology. The plot on purpose does not use the average performance across multiple runs to illustrate that communication even between close nodes is subject to latency variability. Both the topology and the variability hold implications for the latency expectations when it comes to the propagation of decision aids.

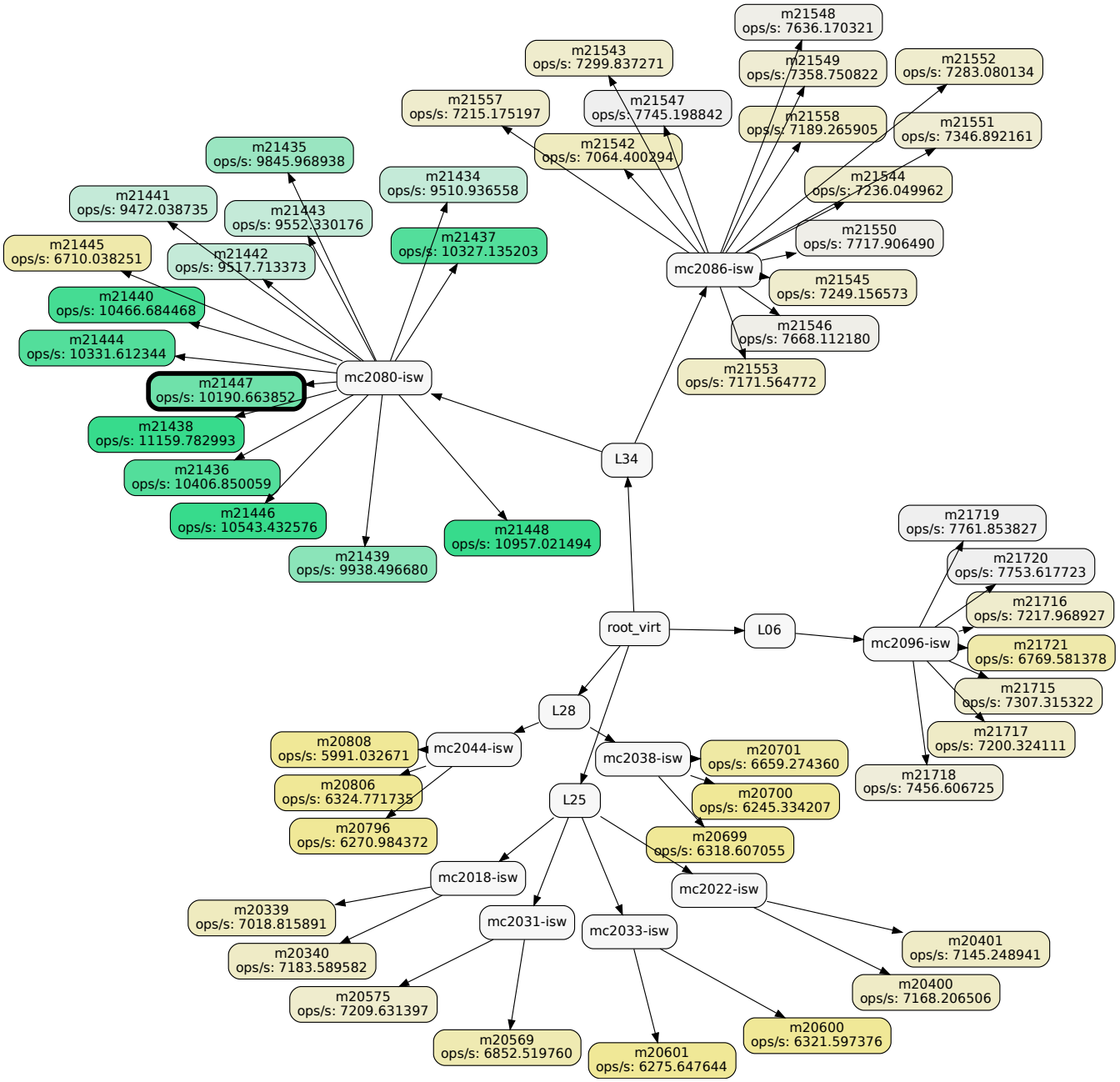


Figure 7.27: Topology overlay of a column in the point-to-point latency matrix. With node `m21447` as the message origin the visualization shows how latency/operations per second degrades as the hop count increases.

Topology Overlay: As an explanation for the observable patterns in Figure 7.26 it is useful to consider the measured latency values relative to the network topology. Figure 7.27 shows a graph of the network topology obtained from Slurm. Nodes in white are physical and virtual switches, while the colored nodes are individual hosts in the allocation. The graph can be interpreted as a single column from Figure 7.26 being projected onto the network graph. The reference origin, here node `m21447` is indicated by a thick black border. The displayed value indicates completed 4 byte messages.

The topology overlay confirms the relationship between latency and network topology and shows the different latency domains which unfold as a result of increasing hop counts. It also demonstrates that performance can fluctuate a lot even within closely connected groups. As a result, a decision aid might need to drop its available update frequency to the slowest rank in this group. As such it supports the notion of hierarchically organizing some collective decisions as stated in Chapter 6 (Architecture for Workflow-Aware Decision Components).

7.4.2 *Effect of Message Size*

This benchmark series was conducted to quantify how the message size influences propagation properties for decision aids. For this, the point-to-point measurements were repeated using different message sizes from 32 bytes to 4 MiB. This spectrum of message sizes is relevant because decision aids might be very small if a decision component might require only a boolean value which sways a decision on the one hand, or quite large up to several megabytes when loading trained weights for a neural network in a decision component.
The measurements were conducted on two different systems one based on an Infiniband interconnect and one based on the now-discontinued Omnipath interconnect. Interestingly, the general performance profiles are very similar despite the different underlying technologies. A more nuanced analysis shows a number of differences, for example, when considering the distribution clusters with similar throughput and latency dynamics. These to a large extent can be explained as being artifacts from the network topology in combination with the allocated compute nodes.

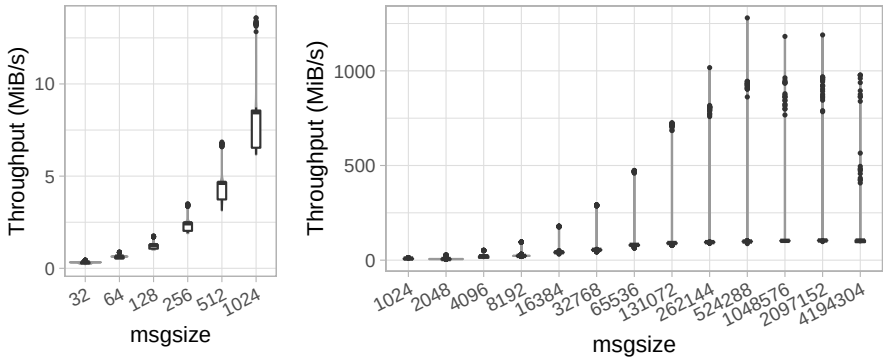


Figure 7.28: Expected throughput as the message size increases on DKRZ Mistral.

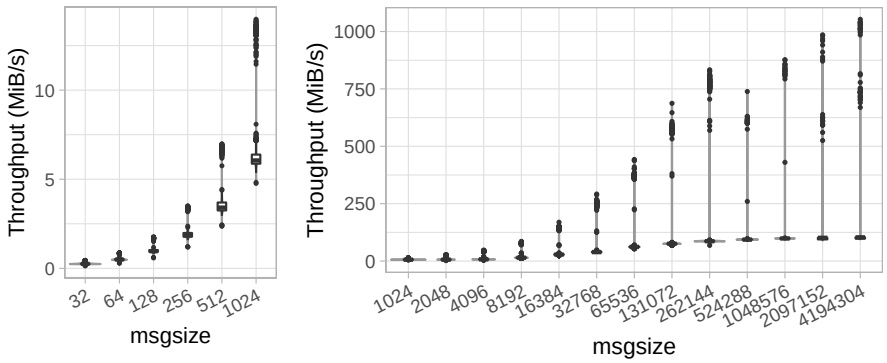


Figure 7.29: Expected throughput as the message size increases on ANL's LCRC Bebob.

Throughput dynamics for different message sizes: Figure 7.28 and Figure 7.29 plot the throughput in MiB/s measured on the Mistral and Bebob supercomputers, respectively. For small message sizes the throughput on Mistral experiences higher variability than on Bebob, this in part can be explained

by the higher hop count from node to node on Mistral. For large messages, Mistral achieves higher performance for some outliers, but the bulk of measurements remain below 125 MiB/s on both systems. Both systems experience multiple groups of measurements with comparable performance as an artifact of the system topology.

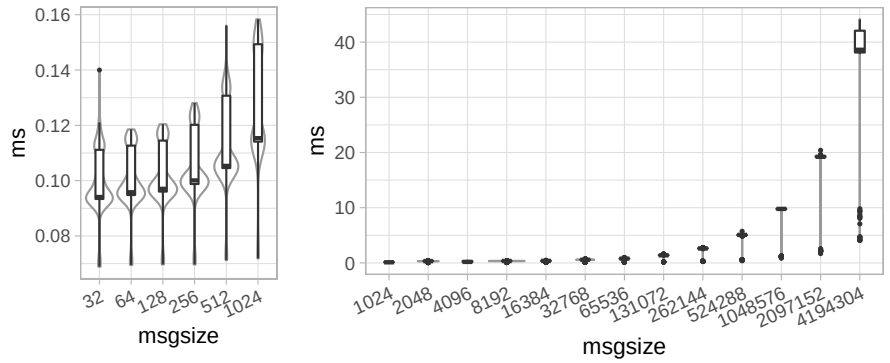


Figure 7.30: Expected latency as the message size increases on DKRZ Mistral.

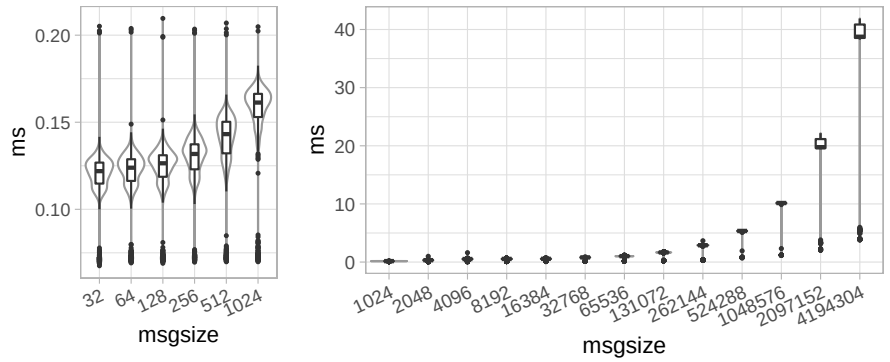


Figure 7.31: Expected latency as the message size increases on ANL's LCRC Bebob.

Latency dynamics for different message sizes: Figure 7.30 and Figure 7.31 plot the latency in ms measured on the Mistral and Bebob supercomputers, respectively. As latency and throughput are intricately linked, the higher variability already observed for throughput are also found for the latency plot. Latency on Mistral is slightly lower on average between 0.10 ms and 0.12 ms for messages between 32 and 1024 bytes. Notable on Bebob is the higher spread for outliers also into the 0.20 ms regime which can not be observed on Mistral. For 1, 2 and 4MiB large messages both systems peak out at about 20-40 ms.

7.4.3 *Conclusion for Throughput and Latency Boundaries*

The performance analysis demonstrates that ZeroMQ can be a viable transport for DAPS even when using an unfavorable configuration based on TCP for most decision aids discussed. The benchmark shows that the resulting performance is sufficient for most discussed use cases of propagating decision aids in the context of scientific workflows. This is the case because many common persistency APIs and state systems require a decision at a far lower rate. Even the relatively fast past paced HDFS, for example, has a default heartbeat interval of three seconds [Shvachko et al., 2010] which is used to inform data replication.
Only partially assessed was the impact of long-term performance variability, as demonstrated by [Lockwood et al., 2018] which shows that there can be considerable changes in performance expected over the course of a day/week/month/year or after system updates.

7.5 End-to-end Proof of Concept

This section demonstrates how the different components such as decision aids and the Decision Aid Propagation Service (DAPS) have to come together to achieve a workflow-aware storage optimization within HPC environments. Mimicking the ICON climate model from Section 7.2 (Use Case: Analysis of the ICON Climate Model) introduced earlier, a proxy application was developed to reduce dependencies and for better control over tuning parameters. In particular, the proxy application is structured so that it simplifies performing parameter sweeps for verification.

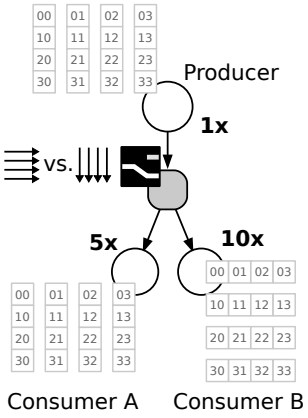


Figure 7.32: Structure of the workflow used for the proof-of-concept with a single writer, and two readers with different magnitude and access pattern.

Experimental Setup: The underlying workflow uses a single-producer/multiple-consumers setup which is commonly found in scientific workflows. The workflow, also illustrated in Figure 7.32, is structured as follows:

1. A producer P generates data using HDF5 and stores it to a parallel file system. It is assumed P defaults to passing tuning parameters for chunk-size and striping optimized for the write case. The task is periodically repeated at the beginning of each workflow execution.
2. Multiple consumers A and B depend on this data. But they do favor different domain decomposition and one consumer is executed more frequently than the other.
   (a) Consumer A favors a row-wise domain decomposition and is run only at a frequency f_A of 5 times per workflow.
   (b) Consumer B favors a column-wise domain decomposition and is run more often at a frequency of f_B of 10 times per workflow.

Example Dataset: The structure of the dataset used for measurements corresponds to the output in Listing 7.3, albeit using larger data volumes. The shared file in the experiments is 10 GiB. For illustration purposes, only a two-dimensional dataset is considered.

Listing 7.3: Excerpt from `h5dump` illustrating the general structure of the dataset used in this benchmark. The data section is populated by different processes each writing its rank a to a subset of the 16x16 dataspace using a chunked data layout.

```
GROUP "/" {
        DATASET "mydataset" {
                DATATYPE  H5T_STD_I8LE
                DATASPACE  SIMPLE { ( 16, 16 ) / ( 16, 16 ) }
                STORAGE_LAYOUT {
                        CHUNKED ( 2, 16 )
                        SIZE 256
                }
                FILLVALUE {
                        FILL_TIME H5D_FILL_TIME_IFSET
                        VALUE  H5D_FILL_VALUE_DEFAULT
                }
                ALLOCATION_TIME {
                        H5D_ALLOC_TIME_EARLY
                }
                DATA {
                        (0,0): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        (1,0): 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                        (2,0): 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                        (3,0): 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                        :
                        (12,0): 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
                        (13,0): 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
                        (14,0): 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
                        (15,0): 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7
                }
        }
}
```

Instrumentation: Instrumentation is provided by Darshan with extended tracing capabilities enabled. While this might be a burden on large workflows the results in Section 7.3.3 (Decision Aid: File Type Clustering and Classification) demonstrate that more light-weight detectors can be constructed that are fast enough to allow decisions to be performed in-band.

7.5.1 HDF5 and Lustre Tuning Parameters

HDF5 features a wide range of tuning parameters when used in combination with parallel file systems. Optimizing parallel I/O for a multi-node application with HDF5 can be a complex task due to the interplay of different optimization factors. This section briefly introduces the tuning parameters which have to be considered at different layers and how they interfere. A number of best practices to tune HDF5 I/O when using a Lustre filesystem have been analyzed by [Howison et al., 2012]. A workflow perspective, however, is not taken. All performed measurements in this evaluation have HDF5 chunksize and Lustre stripe size coincide.

Domain Decomposition: Provided by the resource allocation is a number of ranks scattered across multiple nodes. For this benchmark computational considerations for the domain decomposition are disregarded, but in practice, the effective domain decomposition does depend on the computational intensity of the problem.

Dataset and Hyperslabs: Writing and accessing subsets of data in HDF5 is achieved using hyperslabs. These settings affect application-level I/O calls, although HDF5 typically will internally transform and then dispatch data depending on a variety of settings as is illustrated in Figure 7.33.

Dataset and Hyperslabs: Datasets in HDF5 can be arbitrary sized, which requires breaking up datasets into so-called chunks. Chunking settings have to balance the combination of nodes involved as well as with potential stripesize and stripecount configurations on lower levels.

Alignment: By default HDF5 files are packed tightly to conserve space, compare Figure 7.34. For parallel access semantics this can be a disadvantage: Even small HDF5 objects stored on a parallel storage system might end up on two different targets, effectively requiring clients to issue two requests instead of one. HDF5 allows users to provide an alignment parameter which while increasing file size, distributes objects in a more predictable pattern across storage targets.

Stripesize and Stripecount (Lustre): The stripesize determines the number of bytes to be stored to a particular target per stripe as illustrated in Figure 7.35. An optimal value for the stripesize depends on the latency and throughput dynamics of the system. Lustre, for example, requires stripe sizes to be multiples of 64 KiB. Very small stripe sizes are typically not useful due to request latency. Disregarding latency, small stripe sizes in combination with a high stripecount works favorable for overall throughput but increases request counts and load across the storage system. As a result the optimal stripesize, stripecount combination also depends on the degree of parallelism on the data generator and consumer sides.

Other: A number of additional optimization strategies can be applied but are not included in the following benchmarks. For HDF5 datasets with many objects, metadata can impact I/O performance, as HDF5 has to collect metadata information spread across different index and tree structures. HDF5 offers a possibility to set the number of B-tree entries.

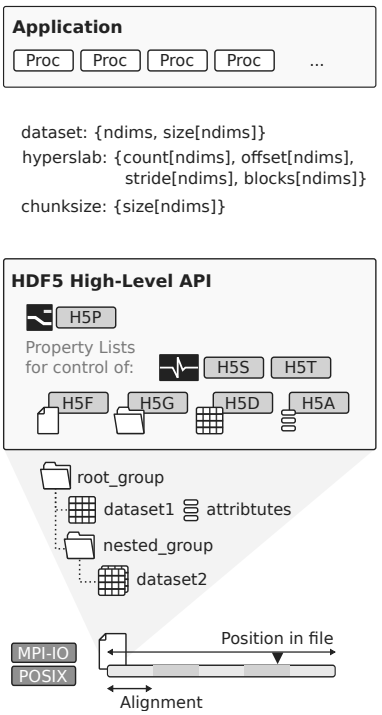


Figure 7.33: A multiprocess application with the tuneables to describe the datasets and the storage layout which the HDF5 library then projects into the byte-sequence of the HDF5 file.

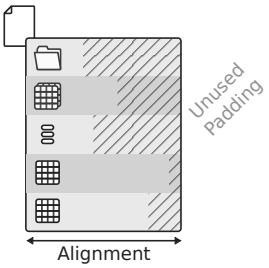


Figure 7.34: Effect of the alignment tuning parameter which adds paddings so HDF5 objects are easier to align with the parallel file system/object storage.

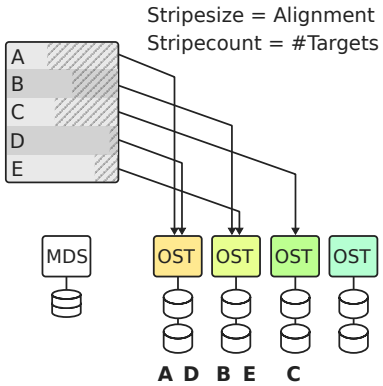


Figure 7.35: Mapping of the HDF5 file to different object storage targets controlled by the stripesize and stripecount tunables to allow parallel I/O.

7.5.2 Workflow Discovery and Workflow-specific Decision Aid

With the understanding that data producers influence the performance characteristics of subsequent data consumers, the case for workflow-aware performance optimization is established. As these optimizations are highly application and workflow-specific, they further motivate the introduction of specialized decision aids as an interchangeable abstraction.
In Section 7.3.1 (Decision Aid: Augmenting Workflow Graphs with I/O Behavior) obtaining a workflow data dependency graph for different workflow engines was demonstrated. In this section, a secondary decision aid to optimize storage I/O from such a task-data dependency graph is derived. For this example, the decision aid is generated starting from a shared dataset for which participating tasks and a common set of tunables are considered. Table 7.3 shows the data structure necessary for an exhaustive bookkeeping scenario from which a decision component will pick a winner.
The underlying heuristic is an implementation detail and could be swapped out for different strategies on a case to case basis. For this example, the tables are structured to allow the calculations of confidence and total workflow cost under different tuneable configurations.
The total workflow cost is calculated by summing over the product of task frequency f_{task} and an approximation of the average task cost $\varnothing_{task}$. The *total_workflow_cost* under a specific tuning scenario with a set of tunables denoted as $tune_X$ is defined as follows:

$$total\_workflow\_cost(tune_P, tune_A, tune_B) = \sum_{task}^{P,A,B} f_{task} * \varnothing_{task}$$

The bookkeeping tables, typically, would not be exhaustive but might include user-defined or a generated list of considered combinations. The DAs sizes in this discussion are given as follows for exhaustive tables:

$$\texttt{sizeof}(DA) = log_2(tuneables^{tasks}) + \sum_{variable}^{\#,\varnothing,\sigma^2,\ldots} \texttt{sizeof}(variable)$$

This also demonstrates that additional filtering is necessary to limit sampling and decision aid size. An example heuristic applicable for this scenario could be based on the factors of how often individual tasks are being executed, where low-frequency tasks are ignored.
By preserving the sum and the number of observations, or preferably using Welford's Method [Welford, 1962; Chan et al., 1982] to calculate running averages and variances, a confidence score to ensure a sufficient number of measurements can be calculated. This confidence score, here simply based on the variance σ^2, is then used as a termination criterion to stop searching as illustrated in Listing 7.4. Robustness of the score has to be balanced against the requirements of being light-weight on memory and effective for the particular scenario. Integration into the decision process for this demonstration invokes a DAPS wrapper from a job script and passes the recommended tuneable to the application as a command-line argument.

Table 7.3: Workflow-specific decision aid for tracking performance combinations. Here requiring 188 bytes to allow cost estimates and confidence calculations: 12 bytes for task frequency counts, 72 bytes for task performance and 104 for workflow performance.

Task Frequencies:

f_P	f_A	f_B
`int`	`int`	`int`

Total workflow performance/cost estimates as the core decision aid to which a workflow would subscribe to:

Stor	Mem	#	$\varnothing, \sigma^2, \ldots$
R	A|R	`int`	`float[2]`
R	A|C	`int`	`float[2]`
C	A|R	`int`	`float[2]`
C	A|C	`int`	`float[2]`
R	B|R	`int`	`float[2]`
R	B|C	`int`	`float[2]`
C	B|R	`int`	`float[2]`
C	B|C	`int`	`float[2]`

Individual task performance/cost estimates to avoid invalidating all gathered experience if the workflow is changed:

P	A	B	#	$\varnothing, \sigma^2, \ldots$
R	-	R	`int`	`float[2]`
R	-	C	`int`	`float[2]`
R	R	-	`int`	`float[2]`
R	C	-	`int`	`float[2]`
C	-	R	`int`	`float[2]`
C	-	C	`int`	`float[2]`
C	R	-	`int`	`float[2]`
C	C	-	`int`	`float[2]`

Listing 7.4: Pseudo code for a Monte-Carlo sampling approach to first sample the performance of different tuneable combinations and then settle on a recommendation after confidence exceeds a certain threshold.

```
import daps, random, sys
da = daps.load_da('wdfhol')     # aka Workflow Decision for HDF5 on Lustre
da.topic_subscribe(['wf_perf')  # implicitly subscribes to task_perf and task_freq
recs = da.get_recommendations(sys.argv[1])
recs.sort(key='performance'))
if recs[0].confidence > threshold:
    print(recs[0])
else:
    print(random.choices(recs.options())
```

7.5.3 *Behavior Adaptation over Time*

With these decision aids in place, it becomes possible to perform workflow-aware storage I/O optimization using tuning parameters exposed by HDF5 and Lustre. This section demonstrates how a decision aid after considering telemetry from multiple workflow executions, is able to settle on a tuneable recommendation which optimized total workflow cost. The discussion then proceeds to demonstrate that a decision aid constructed this way is further robust to changing task frequencies.
Table 7.4 documents the behavior change over time when delegating the storage layout decision across multiple executions to a workflow-specific decision component. After the first full workflow execution the workflow is known and the workflow task-data dependency graph can be assumed to be known together with the workflows task frequencies. As the workflow is being executed repeatedly, performance observations are collected which are then preserved in reduced form in the workflow-specific decision aid. The decision component uses a search termination criterion to determine if a decision is good enough. In this example, the DA settles on a tuneable recommendation if the standard deviation for all three performance estimates used to calculate total workflow cost falls below 250 Mib/s. The threshold should be based on the storage systems' performance variability.

	Producer		Consumer A		Consumer B	
Run #	Tuneable	Performance MiB/s	Tuneable	Performance MiB/s	Tuneable	Performance MiB/s
Run 1	R	2018.938				
			A|R	197.705	B|R	101.108
			A|R	570.779	B|R	106.633
			A|R	195.070	B|R	102.129
			A|R	179.213	B|R	88.211
			A|R	200.777	B|R	108.609
					:	:
	R	∅ 2018.938 ± 0.000	A|R	∅ 268.709 ± 151.218	B|R	∅ 101.383 ± 5.287
Workflow discovery completed: Producer (Factor f_P: 1), Consumer A (Factor f_A: 5), Consumer B (Factor f_B: 10)						
Run 2	C	3333.741				
			A|C	99.147	B|C	166.705
			A|C	109.927	B|C	248.555
			:	:	:	:
	C	∅ 3333.741 ± 0.000	A|C	∅ 110.393 ± 13.799	B|C	∅ 178.072 ± 24.517
Run 3	R	∅ 1984.361 ± 34.576	A|R	∅ 387.576 ± 257.670	B|R	∅ 107.134 ± 10.121
	C	∅ 3333.741 ± 0.000	A|C	∅ 110.394 ± 13.799	B|C	∅ 178.072 ± 24.517
Run 4	R	∅ 1984.362 ± 34.576	A|R	∅ 387.576 ± 257.670	B|R	∅ 107.134 ± 10.121
	C	∅ 2742.075 ± 591.666	A|C	∅ 88.535 ± 23.941	B|C	∅ 130.702 ± 50.444
Decision aid is set to settle once performance deviation is below 250 MiB/sec (with zero not allowed) for winning tuneable setting:						
Run 5	R	∅ 1862.071 ± 175.234	A|R	∅ 341.225 ± 220.370	B|R	∅ 113.029 ± 13.090
	C	∅ 2742.075 ± 591.666	A|C	∅ 88.535 ± 23.941	B|C	∅ 130.702 ± 50.444

Table 7.4: Measurements over time with the system exploring different configurations and then settling on the one with the highest performance among multiple measured workflow runs.

While many different sampling strategies can be pursued, in this example, the decision aid only obtains a tuneable once at the beginning of each workflow execution. For this reason, the averages and standard deviation for either R or C remain the same between every other workflow run. In

practical deployments, more elaborate sampling strategies, for example, on a per task granularity would allow for faster convergence. In ensemble pipelines, as used in UQ workflows, for example, multiple producers would allow exploring multiple storage layout representations in parallel. In fact, the decision aid is flexible enough already to do so, as it was deliberately constructed to work well with parallel adaptive sampling methods such as proposed and implemented by [Nijholt et al., 2019].

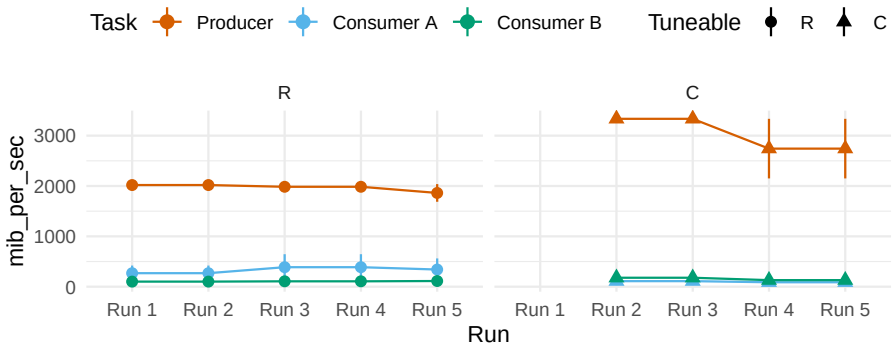


Figure 7.36: Changing state of the decision aid, tracking individual task throughput prediction across multiple runs. On the left the prediction of time including uncertainty for tasks under the row-wise tuneable are listed, analog to that the right plots covers the column-wise tuneable behavior. Note how the column-wise tuneable due to I/O variability acquires uncertainty in Run 4.

Converging Confidence: The decision aid is constructed on the assumption of the central limit theorem extending also to collected I/O observations. Depending on the probability distribution it is, therefore, possible to allow to constrain the decision aid by a confidence threshold it has to surpass before it is allowed to sway a decision. For non-normal distributed performance metrics different search termination criteria have to be explored. Figure 7.36 illustrates how confidence evolves over multiple runs until the threshold condition is met. The measurements demonstrate that also task I/O is subject to I/O variability, as is especially apparent for Run 4 for the column-wise producer which actually acquires additional uncertainty as measurements are used to update the decision aid.
Thus while for a sufficient number of measurements sensitivity to fluctuations like this is vanishing, it can still be adequate to explore adaptive approaches to improve decision quality, such as windowed approaches, calibration measurements, or models that can cope with concept drift [Madireddy et al., 2019].

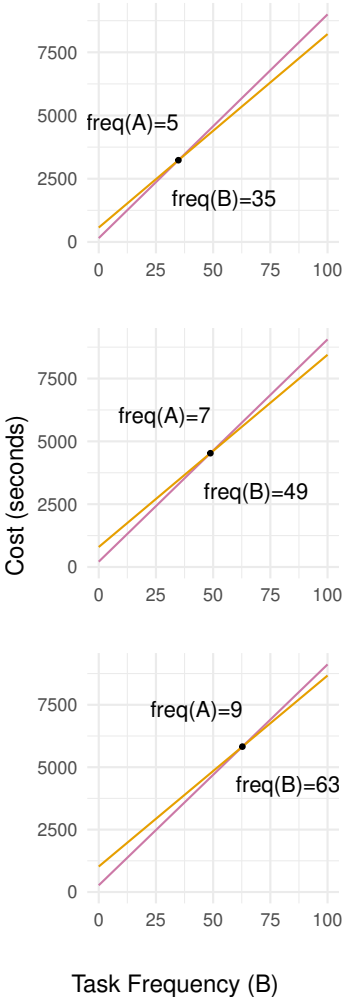


Figure 7.37: Predicted decision boundaries to switch which consumer task type (A or B) the producer should cater for. Based on total workflow cost calculated using estimated cost and task frequencies of each task.

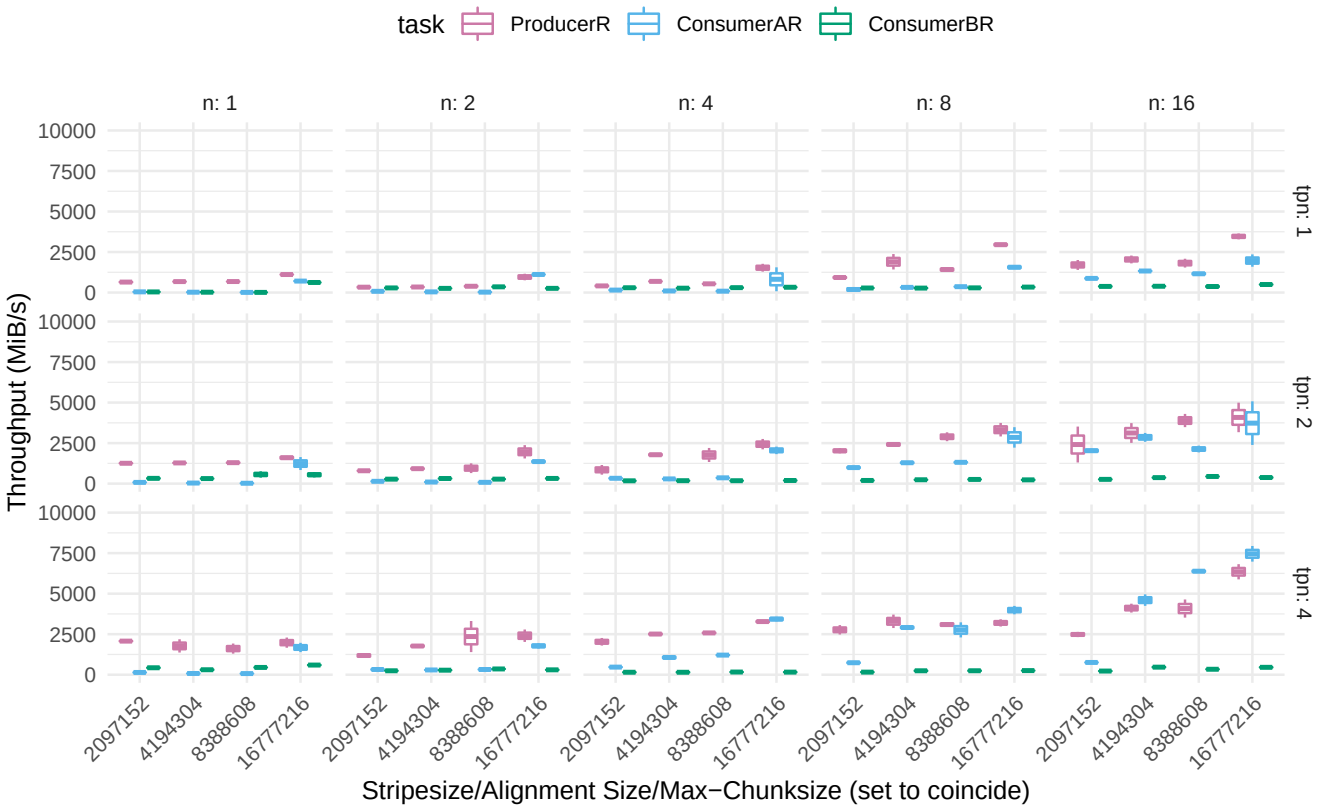
Predicting Decision Boundaries: The decision aid used in this example was designed to be robust to changing task frequencies. Why this is relevant, becomes apparent by considering the decision boundaries when constraining task frequencies. For the task frequencies of $f_A = 5$ and $f_B = 10$ workflow cost is estimated at 1036 seconds (row-wise) and 1333 seconds (column-wise). Consequently, the decision aid settled on the row-wise layout for the producer.
As the ratio of task frequencies changes, so can the decision for the storage layout. For example, the decision aid switches to a column-wise layout when the frequency of task B increases to $f_B >= 35$. Depending on the structure of the decision aid, these decision boundaries can be determined analytically as illustrated in Figure 7.37. For three different task frequencies, f_A for Task A, the decision switch points are predicted based on the available state of the decision aid after Run 5.

7.5.4 Parameter Study to Investigate Performance Behavior

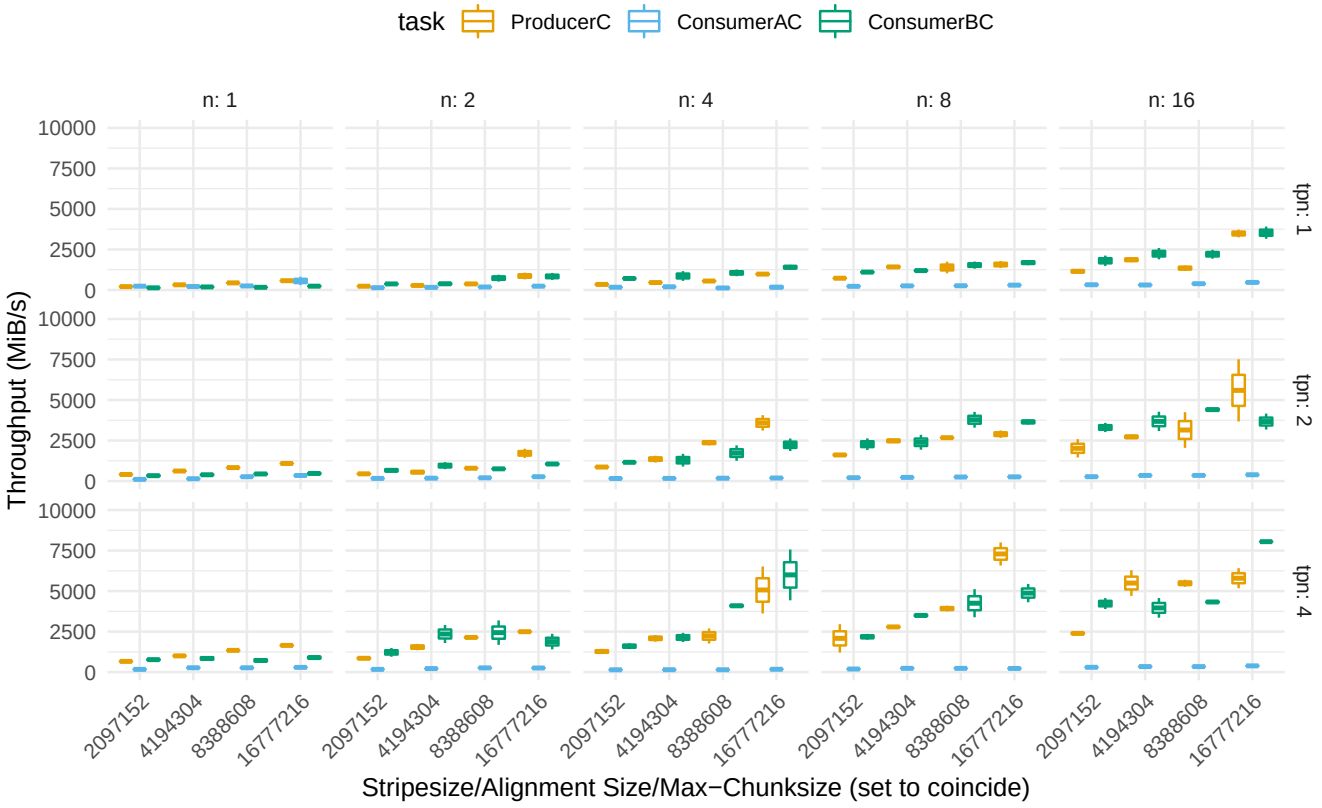
This section provides a detailed analysis of the performance relationship between producers and consumers and shows that severe penalties can be imposed when there is a mismatch. By considering the results it can be explained why the previously discussed decision aid is effective.
A second objective for this section is to demonstrate that I/O performance can be conceptualized as a performance landscape with some performance variability. This view motivates the feasibility and utility of approximating this performance landscape using neural networks or using optimization to find local minima. Finally, the measurements are used for verification and decision quality assessments for which they serve as baseline measurements to better understand the performance landscape for the following tuning parameters:

- Task type: Whether the task is a data producer (P) or a data consumer.
- Access-Pattern: Whether the access pattern is using a row-wise (A) or column-wise (B) domain decomposition on the compute side.
- Storage-Layout: Row-wise (R) or column-wise (C) storage layout. Many other storage layouts are possible in principle but they are not considered for this benchmark.
- n: Number of nodes used to decompose the domain.
- tpn: Number of tasks per node used to decompose the domain.
- *blocksize*: The block size used for chunks, alignment and stripesize.

The results of the parameter study are summarized in Figure 7.38 and Figure 7.39. The measurements are colored by their task and configuration. Producers R and C are row- and column-wise writers respectively. ConsumersAR is a measurement when reading row-wise from a row-wise file, while ConsumerAC is reading row-wise from a column-wise file layout. Similarly, ConsumerBR and ConsumerBC are, respectively, reading column-wise from row- and column-wise file layouts.
Confirming the task-data relationship hypothesis, the producer-consumer relationship imposes a performance penalty if there is a mismatch. A row-wise consumer performs better on a row-wise file layout, and a column-wise consumer performs better on a column-wise file layout.
In Figure 7.38 (a), for example, this becomes apparent for the ConsumerBR (green) which does not scale as node count or task per nodes are increased. When switching the storage layout to cater to B, as illustrated in Figure 7.38 (b), now the ConsumerAC (blue) experiences no performance gains.
Considering producer and consumer performance behavior independently, Figure 7.39 plots the data from a different perspective to allow comparing producers and best-performing consumers. Overall the column-wise producers perform slightly better than the row-wise producers but the performance difference likely has to be attributed to performance variability, see Figure 7.39 (a). A similar effect can be observed for consumers, in Figure 7.39 (b) but only including the well-performing consumer configurations. Comparing row-wise and the column-wise consumer, again column-wise consumers feature slightly better performance. Spuriously, some configurations experience higher performance variability, likely because of network or storage contention.

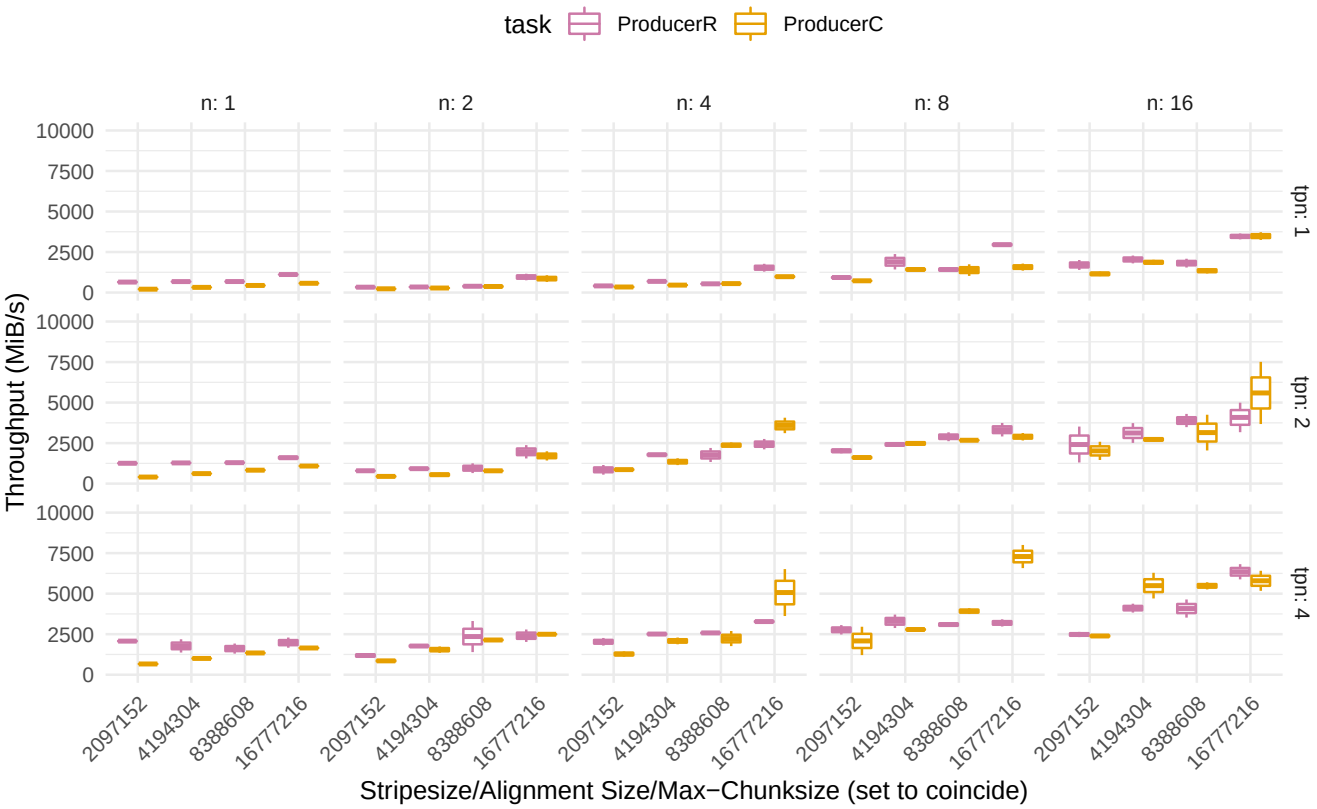


(a) Workflow on row-wise storage layout for shared data file. ConsumerAR scales while Consumer BR does not.

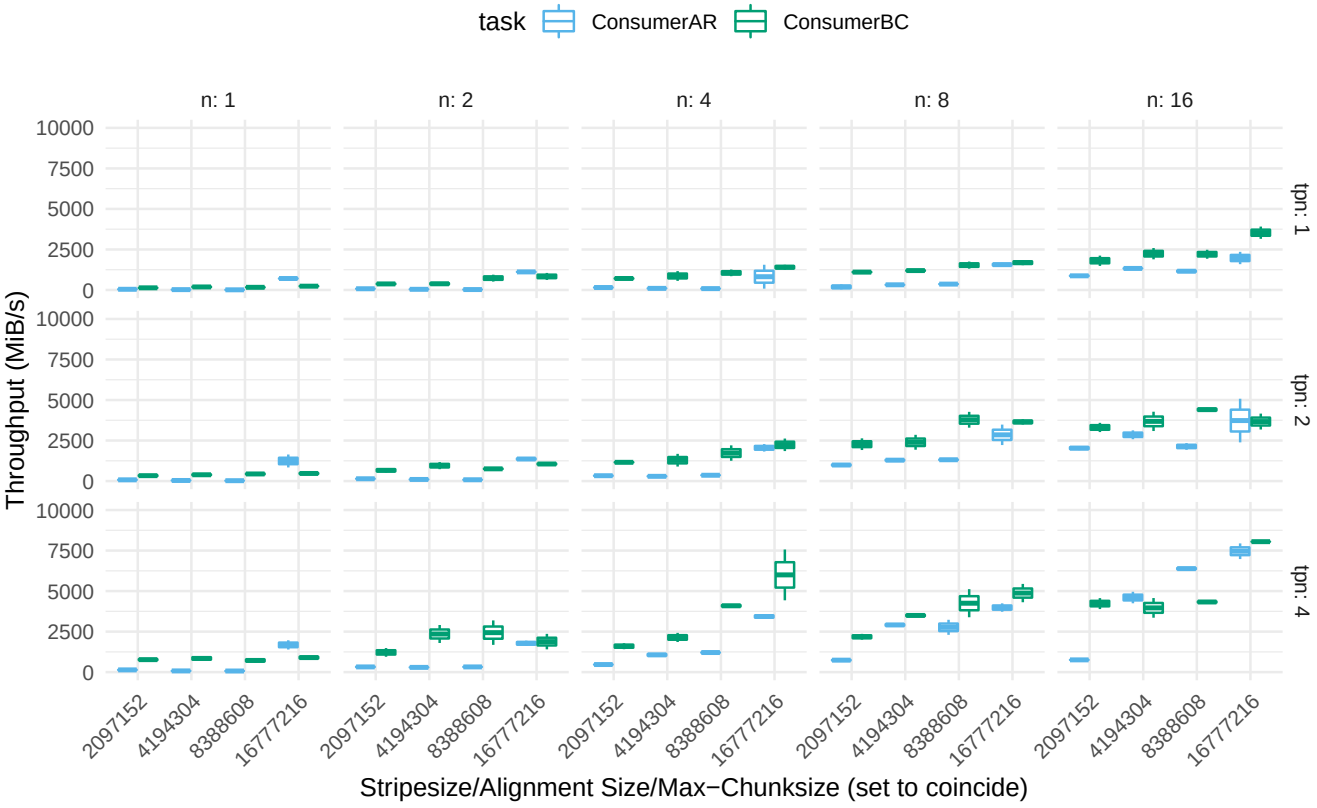


(b) Workflow on column-wise storage layout for shared data file. ConsumerBC scales while ConsumerAC does not.

Figure 7.38: Parameter study to understand the impact of different tuning parameters such as node count, stripesize, alignment, and chunksize on task performance. Take note of how the producer always only caters towards one of the two consumers, and how the situation is inverted as the producer changes strategy.



(a) Comparison of writers performances for different tuneable.



(b) Comparison of best performing readers .

Figure 7.39: Parameter study to reveal the influence different node counts, task per node, stripesize, alignment, and domain decomposition have on the performance of producers and consumers. (a) compares producer behavior and shows that there is only minor differences. Similarly, (b) compares the performance of well-performing consumers side by side.

7.5.5 HDF5 File-Layout: Two Serialization Scenarios

To explain the observed performance penalty if a producer and consumer mismatch is present, it is necessary to consider the effect the different tunables have on the storage layout serialization.
Typically, a combination of the tuning parameters introduced in the previous section conspires to realize performance improvements for a particular system configuration and a number of narrowly defined access patterns. This is the case because various of the tuning parameters control how data is represented in the serialized file layout which is passed to storage systems. The storage system in turn is responsible to map the serial representation across the different targets if it is a parallel distributed storage system.
To explain the performance measurements in the next section consider the positive/negative impact that these tunables can have for the simplified HDF5 dataset illustrated previously in Listing 7.3.
Consider the raw sequence for this HDF5 file is illustrated in Figure 7.40. It was created by an MPI application with 16 ranks and each rank is writing a small portion of data, here it's own rank. The example also illustrates the organization of the HDF5 file, with raw data here in chunks and the alignment parameter set here to only 128 bytes for demonstration purposes. The alignment parameter of HDF5 allows to ensure that file system stripe size and HDF5 object boundaries coincide.

```
1  0000000:   894844460d0a1a0a  0000000000080800  0400100000000000  0000000000000000    .HDF.....................
2  0000020:   ffffffffffffffff  4819000000000000  ffffffffffffffff  0000000000000000    ........H.....................
3  :
4  0000100:   5452454500000100  ffffffffffffffff  ffffffffffffffff  00d0000000000000    TREE.........................
5  0000120:   0018000000000000  0800000000000000  0000000000000000  0000000000000000    ............................
6  :
7  0000380:   4845415000000000  5800000000000000  1800000000000000  a003000000000000    HEAP....X....................
8  00003a0:   0000000000000000  6d79646174617365  7400000000000000  0100000000000000    ........mydataset............
9  :
10 0000580:   5452454501000800  ffffffffffffffff  ffffffffffffffff  2000000000000000    TREE................... .......
11 00005a0:   0000000000000000  0000000000000000  0000000000000000  0010000000000000    ............................
12 :
13 0001000:   0000000000000000  0000000000000000  0101010101010101  0101010101010101    ............................
14 0001020:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
15 0001040:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
16 0001060:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
17
18 0001080:   0202020202020202  0202020202020202  0303030303030303  0303030303030303    ............................
19 00010a0:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
20 00010c0:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
21 00010e0:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
22
23 :          Chunks
24
25 0001400:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
26 0001420:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
27 :
28 0001800:   534e4f4401000100  0800000000000000  0004000000000000  0000000000000000    SNOD........................
29 0001820:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
30 :
31 0001900:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
32 0001920:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ............................
33 0001940:   0000000000000000                                                         ........
```

Figure 7.40: Annotated excerpt of a HDF5 file using the `xxd` command-line utility. Note the different HDF5 data structures in the beginning, followed by chunks of raw data.

A continuous non-interleaved serialization: Chunking affects the serialization of raw data in the file. To demonstrate this, consider the raw data chunks in Figure 7.40 (line 13 or line 18) for which each process writes its current rank, thus marking which values have been touched by which process. In

this case, each process is accessing continuous regions in the file, which typically yields optimal I/O performance for sequential access.

```
:
0001080:   0203020302030203  0203020302030203  0203020302030203  0203020302030203    ................................
00010a0:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ................................
00010c0:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ................................
00010e0:   0000000000000000  0000000000000000  0000000000000000  0000000000000000    ................................
:
```

Figure 7.41: A chunk serialization when using potentially suboptimal chunk settings.

A worst-case serialization: Sometimes chunk settings can lead to I/O activity from different nodes to become interleaved as illustrated in Figure 7.41. While this is not necessarily a problem, as the data is adjacent as far as the HDF5 dataspace is concerned, it might contribute to performance degradation depending on the application or workflow access pattern. Often this will result in very low I/O performance because more data then necessary has to be loaded or because target nodes experience additional load.

7.5.6 End-to-End Proof-of-Concept Conclusion

The end-to-end proof-of-concept demonstrates that it is possible to allow workflow-aware decision-making to optimize storage performance. It also demonstrates that without a supporting ecosystem in place, doing so is very complicated often requiring considerable engineering effort.
The approach shows that after analyzing a real-world application it is possible to apply an abstract decision aid to address optimization opportunities for the specific workflow.
A precondition for workflow-aware decision-making, however, is to acquire a task-data dependency graph which allows identifying producer/consumer relationships as were exploited to optimize the workflow in this end-to-end example. From this available tunables can be determined, which then allows exploring their effect on I/O performance which gives rise to a performance landscape. This performance landscape may be obtained from benchmarks, but can also be sampled from workflow or task executions. It seems likely that hybrid approaches for this might be the most plausible deployment model.
By analyzing the behavior change over time and collapsing the information into a decision aid, it becomes both possible and efficient to inform workflow storage-tuneable decisions. In fact, if the decision aid is constructed accordingly, for example by ensuring higher granularity, the decision remains useful beyond a particular workflow configuration and can be robust to, for example, different task frequencies.

Chapter Summary

This section evaluated multiple aspects of the proposed design of Chapter 6 (Architecture for Workflow-Aware Decision Components) and demonstrated that the approach is feasible. The discussion starts with the documentation of software dependencies as well as needed modifications to run experiments. It follows a description of systems measured on as a reference for expected performance.
The evaluation focused on three aspects first considering multiple case studies and assessments before discussing a benchmark for the Decision Aid Propagation Service (DAPS) and a proof of concept demonstrating the feasibility of including workflow knowledge for optimization.

The first aspect performs a case study analysis of optimization opportunities in a large climate code and demonstrates how some of the developed tools help with their discovery. It follows an assessment for a variety of decision aids demonstrating the workflow graph reconstruction from telemetry information, as well as the development of *secondary decision aids* useful for automatic characterization of I/O access patterns and filetype inference from clustering I/O fingerprints.

Which decisions can be improved using these decision aids depends on the characteristics of the compute, network, and storage systems deployed at a site. To quantify expected performance and acceptable payloads for decision aids a benchmark was developed and results for two different systems are discussed, suggesting that even propagation within a 10-20 millisecond window is feasible even for decision aids in the order of 2MiB in size.

Finally, revisiting the analysis of the climate application, an end-to-end proof-of-concept is evaluated which demonstrates the feasibility of decision aids for workflow I/O optimization. To do so two decision aids are constructed which allow adapting individual task tuneable configurations in response to the observed performance. For better control and follow-up investigation, a proxy application is developed which can be also used as a benchmark for producer/consumer workflows that use HDF5 for result data. The analysis then takes different tunables such as stripe size, stripe count, and HDF5 chunking into account.

The evaluation could show that by taking performance measurements across multiple workflow executions, the decision aid system can be used to settle on a parameter combination with the highest estimated overall workflow performance while being robust to changing frequencies of subtasks of the workflow if limited additional bookkeeping can be afforded.

8
Summary & Conclusion

This chapter summarizes the dissertation's findings in Section 8.1 and concludes with an outlook of future work in Section 8.2.

8.1 Summary

Applications on supercomputers are increasingly driven by large-scale computational workflows. Many scientific applications executed on HPC resources are fundamentally constrained by I/O bottlenecks due to contention, sub-optimal data layout, as well as divergent technological advancements of compute and storage technologies. To make most of the available technologies it becomes essential to take the I/O behavior of scientific workflows as well as the dynamics of the underlying storage and network technologies into account when deciding how to store data.
This thesis introduces a decision support framework for workflow-aware high-performance storage optimization. The resulting architecture is informed by two extensive technology surveys, the first is assessing the state of HPC workflow orchestration and the second traverses the storage stack for a holistic overview of *information sources* and *decision points* to allow workflow I/O optimization. A novel API and a new service are proposed: The Decision Support API allows to define and delegate performance-sensitive decisions so they can be made workflow-aware, and the Decision Aid Propagation Service (DAPS) dramatically simplifies the collection and distribution of workflow advice.

The first two chapters set the stage for a discussion about workflow-awareness in compute and storage systems by highlighting many of the related challenges faced by the HPC community. Chapter 2 (Background) refines loosely used terminology and describes many of the technical systems which together form compute and storage systems, including their smaller building blocks. Also introduced are some common forms of scientific data, as well as concepts and algorithms which became useful to reason, formalize, and evaluate the architecture proposal from the main chapter. Very briefly, a number of key methods and an intuition regarding their application from the fields of control theory and machine learning are introduced which matured in the last decades to a point that they have become reliable tools instead of mathematical curiosities.

Chapter 3 (Scientific Workflows in High Performance Computing) surveys the landscape of workflow management tools and transforms a formal workflow definition typically limited to a perspective, that aims to better exploit compute resources, into one which accounts for storage systems. The chapter focuses on a practical perspective which also includes examples to demonstrate the spectrum from early coarse-grained workflows to increasingly fine-grained definitions. The chapter ends with an outlook: For one,

the workflow community is consolidating what was learned over decades into more generic workflow definitions as observed in initiatives such as Common Workflow Language (CWL). More importantly, workflows are likely at the core of many upcoming challenges in regards to performance exploitation and data management but also in regards to how science can be executed to be reproducible and accessible across disciplines.

Chapter 5 (Information Sources and Decision Points) takes a technical perspective and explores the storage stack for opportunities to reconstruct the incomplete and abstract workflow descriptions used by workflow management systems. In particular, the performance implications of executing a workflow are hard to reconstruct, but they form the basis for any decision-making attempting to optimize the same. In fact, the quick turnaround time of HPC systems, differences across sites and the pace in which new technologies are introduced, make it unfeasible to get more explicit. Instead, a common language and self-learning methods would be needed which allows to relate information from the user perspective with the technical perspective. Self-learning does not mean to seek a magic bullet which somehow obtains conscious decision-making capabilities, but to find a framework which allows replacing generic and thus potentially inappropriate heuristics with a two-phase process which first learns to recognize context after being offered a number of examples, and then also learns to fit a function to map between context and best performance.

Chapter 6 (Architecture for Workflow-Aware Decision Components) combines what was learned into an architecture proposal, deliberately avoiding to propose a system which would radically replace current systems, but instead permits gradually augmenting capabilities of current systems and software libraries by applying a few surgical changes. A core problem is that contextual information is typically not passed from one system to another as data are being handled by systems which, today, have no choice but to be ignorant about the structure of data and its embedding into workflows. To overcome this, an architecture consisting of the following core building blocks is proposed:

- Discovery tools are required to combine, reduce, and visualize various artifacts collected from workflow engines and different physical systems to reconstruct the performance implications of executing a workflow.

- A new interface, the Decision API, for applications, middleware, and systems to expose decision points and as a mechanism to delegate making a decision to an exchangeable component, thus allowing to provide and adapt a decision component exclusively for a particular workflow, user, or application.

- A new service, the Decision Aid Propagation Service (DAPS) which offers a publish/subscribe interface to allow efficient distribution of decision aids. By decoupling the generation of advice, from the distribution and from consumers, changes to existing software remain minimal while the architecture is flexible to integrate with new systems.

- Decision aids, which often are aggregation units that add value to, for example, raw telemetry by reducing it into information which sways a decision component to conclude one action over another.

Chapter 7 (Evaluation) discussed different aspects of the proposed architecture and shows that workflow aware decision-making in respect to data storage is a viable concept and can in fact be used to improve performance and save resources. The evaluation starts with the analysis of a global climate model (ICON), using tools developed as part of this work and proceeds with the identification of various workflow-related optimization opportunities.
Based on these opportunities three decision aids to support workflow-aware decision-making are implemented and their feasibility is assessed. As a primary decision aid the discovery of workflow I/O behavior from instrumentation data is automated, yielding both, the task-data relationships as well as the magnitude of dataflows. Two secondary decision aids demonstrate how machine learning approaches can be leveraged for complex I/O motif recognition and clustering in support processing of large amounts of workflow artifacts.
To inform the implementation of the Decision Aid Propagation Service (DAPS) a benchmark was developed to verify that for many of the envisioned decision aids, a transport layer relying on ZeroMQ even when using TCP-based propagation, results in latencies within acceptable bounds.
Finally, in an end-to-end proof-of-concept, the feasibility of the decision support framework is demonstrated. For a single-producer multiple-consumer workflow a proxy application based on the I/O behavior of ICON is developed, to simplify the construction of I/O intensive workflows and to gain more direct control over tuning parameters. Based on this proxy workflow a generic decision aid for producer-consumer workloads is evaluated, which allowed adapting the I/O tunable configuration for a data-producing task while optimizing for total workflow performance and being robust to changing task frequencies.

8.1.1 Conclusion

To a large extent, current data centers are not ready to accommodate and account for workflow behavior in storage systems. This dissertation extensively surveyed scientific workflows and traversed the storage stack to discover opportunities to change this. The result is an architecture proposal that demonstratively can be viable for many decision processes that slowly evolve or even remain stable for a given system and workflow.
It becomes apparent that obtaining and executing a decision automatically can become prohibitively expensive with current technologies. In particular, significant effort has to be spent collecting and enhancing workflow artifacts to a point that they can serve as a decision aid. Besides engineering for interoperability, many decisions can not be made in real-time but require post-hoc analysis thus requiring data centers to feature services which allow inquiring about past application or workflow behavior with compact inexpensive to propagate knowledge representations. At the same time, research into new specialized hardware and accelerators for data processing promises to make adaptable and currently sometimes intractable interventions a possibility along the data path both out-of-band and in-band.
To remove friction when applying adaptive I/O optimization strategies that accommodate workflows, it becomes necessary for the community to agree on common concepts to report data dependencies and performance. In addition, it is necessary to expose control to change internal behavior

of, for example, scheduling strategies, data placement decisions, and data representations.

8.2 Outlook & Future Work

With current systems significant opportunity for workflow-aware optimization remains untapped. With the viability of the proposed approach demonstrated, further maturing the architecture lays out a plausible roadmap to change this. Consequently, in future work, additional case studies with production workflows will be conducted which also consider the challenges associated with in situ and machine learning workflows in HPC environments.

Besides applying the existing set of decision aids and decision components developed as part of this thesis to new workflows, the integration with advanced machine learning methodologies but also the exploration of coupling training with systems simulation and reinforcement learning is planned. This not only promises to speed up the training of decision aids but also appears to be an important incubator for the development of more capable decision components in support of workflow-aware high-performance storage systems.

Throughout performing experiments and writing the manuscript various topics deserving additional research have been identified. Two main themes become apparent: 1) Research into decision aids, their representations and their training and 2) research informing the implementation of services to expose information about and propagate advice to decision points. A few areas which are particularly relevant include:

- Research the specifics to inform the implementation of optimizations opportunities discussed for the Decision Aid Propagation Service (DAPS).
- "Standardize" the Decision API or an equivalent with the community.
- Build collections of more generalized neural heuristics and decision components that can then be refined by a particular site.
- Development of end-to-end approaches that couple training and tuning of decision components with system simulation.

In addition, a wide range of tooling and software libraries was developed as became necessary for prototyping and experiments. Where adequate, contributions, and feedback were passed on to developers of existing tools. Planned improvements to tools, that are made available as open-source in hopes that others might find them useful for their research too, include:

- The generalization of the visualization and interactive tools for compatibility beyond custom dashboards or Jupyter Notebooks.
- Python bindings and wrappers to handle workflow-related artifacts such as Darshan records, workflow graphs, and other service or system logs.
- Port existing surrogate models used to simulate hierarchical storage systems to more optimized implementations of discrete event simulators to take advantage of multi-processing.

Bibliography

7-cpu.com (2020). 7-Zip LZMA Benchmark. https://www.7-cpu.com/ (accessed 2020-02-07).

Abdelzaher, T., Diao, Y., Hellerstein, J. L., Lu, C., and Zhu, X. (2008). Introduction to Control Theory And Its Application to Computing Systems. In Liu, Z. and Xia, C. H., editors, *Performance Modeling and Engineering*, pages 185–215. Springer US, Boston, MA. https://doi.org/10.1007/978-0-387-79361-0_7 (accessed 2020-02-26).

Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., and Tallent, N. R. (2009). HPCTOOLKIT: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a. http://doi.wiley.com/10.1002/cpe.1553 (accessed 2020-03-02).

Adhianto, L. and Taffet, P. (2016). Addressing Challenges in Visualizing Huge Call-Path Traces. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*, pages 319–328, Philadelphia, PA, USA. IEEE. http://ieeexplore.ieee.org/document/7576482/ (accessed 2020-03-03).

ADIOS2 (2018). Next generation of ADIOS developed in the Exascale Computing Program: Ornladios/ADIOS2. ADIOS. https://github.com/ornladios/ADIOS2 (accessed 2018-09-09).

Afgan, E., Baker, D., Batut, B., van den Beek, M., Bouvier, D., Čech, M., Chilton, J., Clements, D., Coraor, N., Grüning, B. A., Guerler, A., Hillman-Jackson, J., Hiltemann, S., Jalili, V., Rasche, H., Soranzo, N., Goecks, J., Taylor, J., Nekrutenko, A., and Blankenberg, D. (2018). The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research*, 46(W1):W537–W544. https://academic.oup.com/nar/article/46/W1/W537/5001157 (accessed 2019-05-23).

Agelastos, A., Allan, B., Brandt, J., Cassella, P., Enos, J., Fullop, J., Gentile, A., Monk, S., Naksinehaboon, N., Ogden, J., Rajan, M., Showerman, M., Stevenson, J., Taerat, N., and Tucker, T. (2014). The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 154–165, New Orleans, LA, USA. IEEE. http://ieeexplore.ieee.org/document/7013000/ (accessed 2019-04-22).

Al-Kiswany, S., Costa, L. B., Yang, H., Vairavanathan, E., and Ripeanu, M. (2017). A cross-layer optimized storage system for workflow applications. *Future Generation Computer Systems*, 75:423–437. https://linkinghub.elsevier.com/retrieve/pii/S0167739X17302959 (accessed 2020-02-23).

Al-Kiswany, S., Vairavanathan, E., Costa, L. B., Yang, H., and Ripeanu, M. (2013). The case for cross-layer optimizations in storage: A workflow-optimized storage system. *CoRR*, abs/1301.6195. http://arxiv.org/abs/1301.6195.

Albrecht, M., Donnelly, P., Bui, P., and Thain, D. (2012). Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies - SWEET '12*, pages 1–13, Scottsdale, Arizona. ACM Press. http://dl.acm.org/citation.cfm?doid=2443416.2443417 (accessed 2020-02-26).

Almende B.V. (2018). Vis.js - A dynamic, browser based visualization library. http://visjs.org/ (accessed 2018-08-14).

Altair (2019). PBS: Portable Batch System. https://www.pbsworks.com/Default.aspx (accessed 2019-05-22).

Alted, F. (2010). Why Modern CPUs Are Starving and What Can Be Done about It. *Computing in Science & Engineering*, 12(2):68–71. http://ieeexplore.ieee.org/document/5432301/ (accessed 2020-03-10).

Altintas, I., Berkley, C., Jaeger, E., Jones, M. B., Ludäscher, B., and Mock, S. (2004). Kepler: An extensible system for design and execution of scientific workflows. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004), 21-23 June 2004, Santorini Island, Greece*, pages 423–424. IEEE Computer Society. https://doi.org/10.1109/SSDM.2004.1311241.

Altintas, I., Block, J., de Callafon, R., Crawl, D., Cowart, C., Gupta, A., Nguyen, M., Braun, H.-W., Schulze, J., Gollner, M., Trouve, A., and Smarr, L. (2015). Towards an Integrated Cyberinfrastructure for Scalable Data-driven Monitoring, Dynamic Prediction and Resilience of Wildfires. *Procedia Computer Science*, 51:1633–1642. http://www.sciencedirect.com/science/article/pii/S1877050915011047 (accessed 2020-01-25).

Amstutz, P., Crusoe, M. R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., Ménager, H., Nedeljkovich, M., Scales, M., Soiland-Reyes, S., and Stojanovic, L. (2016). Common Workflow Language, v1.0. page 5921760 Bytes. https://figshare.com/articles/Common_Workflow_Language_draft_3/3115156/2 (accessed 2019-05-27).

Andreadis, G., Versluis, L., Mastenbroek, F., and Iosup, A. (2018). A Reference Architecture for Datacenter Scheduling: Design, Validation, and Experiments. page 15.

Ang, J. A. (2015). High Performance Computing Co-Design Strategies. In *Proceedings of the 2015 International Symposium on Memory Systems*, pages 51–52. ACM. http://dl.acm.org/citation.cfm?id=2818959 (accessed 2017-05-10).

ANL/LCRC (2020). Bebop - Laboratory Computing Resource Center. https://www.lcrc.anl.gov/systems/resources/bebop/ (accessed 2020-02-09).

Anwar, S., Hwang, K., and Sung, W. (2015). Structured Pruning of Deep Convolutional Neural Networks. *arXiv:1512.08571 [cs, stat]*. http://arxiv.org/abs/1512.08571 (accessed 2020-02-17).

Apache Foundation (2019a). Airflow: Contribute to apache/airflow development by creating an account on GitHub. The Apache Software Foundation. https://github.com/apache/airflow (accessed 2019-05-27).

Apache Foundation (2019b). Flink: Stateful Computations over Data Streams. https://flink.apache.org/ (accessed 2019-05-27).

Armstrong, T. G., Wozniak, J. M., Wilde, M., and Foster, I. T. (2014). Compiler techniques for massively scalable implicit task parallelism. In *Proc. SC*.

Backblaze (2016). Hard Drive Test Data. https://kaggle.com/backblaze/hard-drive-test-data (accessed 2020-08-09).

Backblaze (2020). Backblaze Hard Drive Stats. https://www.backblaze.com/b2/hard-drive-test-data.html (accessed 2020-08-09).

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H. (2019). PETSc: Portable, Extensible Toolkit for Scientific Computation. http://www.mcs.anl.gov/petsc.

Bautista-Gomez, L., Tsuboi, S., Komatitsch, D., Cappello, F., Maruyama, N., and Matsuoka, S. (2011). FTI: High performance fault tolerance interface for hybrid systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*, page 1, Seattle, Washington. ACM Press. http://dl.acm.org/citation.cfm?doid=2063384.2063427 (accessed 2020-04-01).

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, Montreal, Quebec, Canada. Association for Computing Machinery. https://doi.org/10.1145/1553374.1553380 (accessed 2020-06-16).

Bent, J., Thain, D., Arpaci-Dusseau, A. C., Arpaci-Dusseau, R. H., and Livny, M. (2004). Explicit control in the batch-aware distributed file system. In Morris, R. T. and Savage, S., editors, *1st Symposium on Networked Systems Design and Implementation (NSDI 2004), March 29-31, 2004, San Francisco, California, USA, Proceedings*, pages 365–378. USENIX. http://www.usenix.org/events/nsdi04/tech/bent.html.

Binkert, N. et al. (2006). Reinhardt, SK. *The M5 Simulator: Modeling Networked Systems*, pages 52–60.

Biven, L., Peterka, T., Bard, D., Bennett, J., Bethel, W., Oldfield, R., Pouchard, L., Sweeney, C., and Wolf, M. (2019). ASCR Workshop on In Situ Data Management - Pre-Workshop Document. https://www.orau.gov/insitudata2019/ISDM-pre-workshop-document-final.pdf (accessed 2019-06-04).

Boehme, D., Gamblin, T., Beckingsale, D., Bremer, P.-T., Gimenez, A., LeGendre, M., Pearce, O., and Schulz, M. (2016). Caliper: Performance Introspection for HPC Software Stacks. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 550–560.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59:291–294.

Brandt, J. M., Gentile, A. C., Mayo, J., Pébay, P. P., Roe, D. C., Thompson, D. C., and Wong, M. (2009). Resource monitoring and management with OVIS to enable HPC in cloud computing environments. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, pages 1–8. IEEE. https://doi.org/10.1109/IPDPS.2009.5161234.

Broad Institute (2019). Workflow Description Language: Specification and Implementations - openwdl/wdl. openwdl. https://github.com/openwdl/wdl (accessed 2019-05-22).

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42. http://arxiv.org/abs/1611.08097 (accessed 2020-08-15).

Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., and Namyst, R. (2010). Hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 180–186, Pisa. IEEE. http://ieeexplore.ieee.org/document/5452445/ (accessed 2020-03-02).

Calder, B., Grunwald, D., Jones, M. P., Lindsay, D. C., Martin, J. H., Mozer, M., and Zorn, B. G. (1997). Evidence-based static branch prediction using machine learning. *ACM Trans. Program. Lang. Syst.*, 19(1):188–222. https://doi.org/10.1145/239912.239923.

Carns, P., Latham, R., Ross, R., Iskra, K., Lang, S., and Riley, K. (2009). 24/7 Characterization of petascale I/O workloads. pages 1–10. IEEE. http://ieeexplore.ieee.org/document/5289150/ (accessed 2018-07-13).

Carns, P. H., Harms, K., Allcock, W. E., Bacon, C., Lang, S., Latham, R., and Ross, R. B. (2011). Understanding and improving computational science storage access through continuous characterization. In Brinkmann, A. and Pease, D., editors, *IEEE 27th Symposium on Mass Storage Systems and Technologies, MSST 2011, Denver, Colorado, USA, May 23-27, 2011*, pages 1–14. IEEE Computer Society. https://doi.org/10.1109/MSST.2011.5937212.

Carreras, R. M. (2018). Towards Energy-efficient Exascale Computing: A Use-case Applying READEX to Alya. page 28.

Carter, S., Armstrong, Z., Schubert, L., Johnson, I., and Olah, C. (2019). Activation Atlas. *Distill*, 4(3):e15. https://distill.pub/2019/activation-atlas (accessed 2020-01-11).

Caruana, R. A. (1993). Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Machine Learning Proceedings 1993*, pages 41–48. Morgan Kaufmann, San Francisco (CA). http://www.sciencedirect.com/science/article/pii/B9781558603073500125 (accessed 2020-07-26).

CEA/LSCE and CNRS/IPSL (2019). XIOS: XML I/O Server. http://forge.ipsl.jussieu.fr/ioserver/wiki (accessed 2019-05-27).

Celery Project (2019). Celery: Distributed Task Queue. Celery. https://github.com/celery/celery (accessed 2019-05-27).

Chaarawi, M. and Koziol, Q. (2014). HDF5 RFC: Virtual Object Layer.

Chan, T. F., Golub, G. H., and LeVeque, R. J. (1982). Updating Formulae and a Pairwise Algorithm for Computing Sample Variances. In Caussinus, H., Ettinger, P., and Tomassone, R., editors, *COMPSTAT 1982 5th Symposium Held at Toulouse 1982*, pages 30–41. Physica-Verlag HD, Heidelberg. http://link.springer.com/10.1007/978-3-642-51461-6_3 (accessed 2020-06-10).

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58. https://doi.org/10.1145/1541880.1541882 (accessed 2020-07-26).

Cheng, P., Lu, Y., Du, Y., and Chen, Z. (2018). Accelerating Scientific Workflows with Tiered Data Management System. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 75–82, Exeter, United Kingdom. IEEE. https://ieeexplore.ieee.org/document/8622780/ (accessed 2020-02-27).

Clark, A., Donahue, J., and Simonyan, K. (2019). Efficient Video Generation on Complex Datasets. *arXiv:1907.06571 [cs, stat]*. http://arxiv.org/abs/1907.06571 (accessed 2019-08-28).

Coen, J. L. and Schroeder, W. (2017). Coupled weather-fire modeling: From research to operational forecasting. 75(1):60.

collectd (2020). Collectd – The system statistics collection daemon. https://collectd.org/ (accessed 2020-03-24).

Conway, M., Moore, R., Rajasekar, A., and Nief, J.-Y. (2011). Demonstration of Policy-Guided Data Preservation Using iRODS. In *2011 IEEE International Symposium on Policies for Distributed Systems and Networks*, pages 173–174.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, Shaker Heights, Ohio, USA. Association for Computing Machinery. https://doi.org/10.1145/800157.805047 (accessed 2020-07-31).

Costa, L. B., Al-Kiswany, S., Yang, H., and Ripeanu, M. (2014). Supporting storage configuration for I/O intensive workflows. In Bode, A., Gerndt, M., Stenström, P., Rauchwerger, L., Miller, B. P., and Schulz, M., editors, *2014 International Conference on Supercomputing, ICS'14, Muenchen, Germany, June 10-13, 2014*, pages 191–200. ACM. https://doi.org/10.1145/2597652.2597679.

Crusoe, M. R., Pope, B., and Pellman, J. (2019). Repository for the CWL standards. https://s.apache.org/existing-workflow-systems (accessed 2019-05-28).

Daly, J. (2006). A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312. http://linkinghub.elsevier.com/retrieve/pii/S0167739X04002213 (accessed 2018-06-20).

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.-H., Vahi, K., and Livny, M. (2004). Pegasus: Mapping scientific workflows onto the grid. In Dikaiakos, M. D., editor, *Grid Computing, Second European across Grids Conference, AxGrids 2004, Nicosia, Cyprus, January 28-30, 2004, Revised Papers*, volume 3165 of *Lecture Notes in Computer Science*, pages 11–20. Springer. https://doi.org/10.1007/978-3-540-28642-4_2.

Deelman, E., Peterka, T., Altintas, I., Carothers, C. D., van Dam, K. K., Moreland, K., Parashar, M., Ramakrishnan, L., Taufer, M., and Vetter, J. (2018). The future of scientific workflows. *The International Journal of High Performance Computing Applications*, 32(1):159–175. http://journals.sagepub.com/doi/10.1177/1094342017704893 (accessed 2018-06-12).

Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P. J., Mayani, R., Chen, W., Ferreira da Silva, R., Livny, M., and Wenger, K. (2015). Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35. http://linkinghub.elsevier.com/retrieve/pii/S0167739X14002015 (accessed 2018-08-05).

Deelman, E., Vahi, K., Rynge, M., Mayani, R., da Silva, R. F., Papadimitriou, G., and Livny, M. (2019). The Evolution of the Pegasus Workflow Management Software. *Computing in Science & Engineering*, 21(4):22–36. https://ieeexplore.ieee.org/document/8725518/ (accessed 2020-02-28).

DFG (2013). Sicherung guter wissenschaftlicher Praxis / Safeguarding Good Scientific Practice. page 112.

Di, S., Guo, H., Pershey, E., Snir, M., and Cappello, F. (2019). Characterizing and Understanding HPC Job Failures Over The 2K-Day Life of IBM BlueGene/Q System. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 473–484.

DOE and NISA (2017). Exascale Computing Project (ECP). https://www.exascaleproject.org/ (accessed 2018-02-04).

Donnell, N. M. (2016). D7.3 Inventory of Exascale Tools and Techniques.

Donnelly, P. and Thain, D. (2013). Design of an active storage cluster file system for DAG workflows. In Sun, X.-H., Chen, Y., and Roth, P. C., editors, *Proceedings of the 2013 International Workshop on Data-Intensive Scalable Computing Systems, DISCS 2013, Denver, Colorado, USA, November 18, 2013*, pages 37–42. ACM. https://doi.org/10.1145/2534645.2534656.

Dorier, M., Dreher, M., Peterka, T., Wozniak, J. M., Antoniu, G., and Raffin, B. (2015). Lessons Learned from Building In Situ Coupling Frameworks. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization - ISAV2015*, pages 19–24, Austin, TX, USA. ACM Press. http://dl.acm.org/citation.cfm?doid=2828612.2828622 (accessed 2019-03-04).

Dorier, M., Ibrahim, S., Antoniu, G., and Ross, R. (2014). Omnisc'IO: A Grammar-Based Approach to Spatial and Temporal I/O Patterns Prediction. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 623–634, New Orleans, LA, USA. IEEE. http://ieeexplore.ieee.org/document/7013038/ (accessed 2020-02-28).

DPDK Project (2020). Data Plane Development Kit (DPDK). https://www.dpdk.org/ (accessed 2020-01-12).

Dreher, M. and Peterka, T. (2017). Decaf: Decoupled Dataflows for In Situ High-Performance Workflows. Technical Report ANL/MCS–TM-371, 1372113. http://www.osti.gov/servlets/purl/1372113/ (accessed 2019-03-04).

Duan, R., Nadeem, F., Wang, J., Zhang, Y., Prodan, R., and Fahringer, T. (2009). A Hybrid Intelligent Method for Performance Modeling and Prediction of Workflow Activities in Grids. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 339–347.

Duro, F. R., Blas, F. J. G., Isaila, F., Carretero, J., Wozniak, J. M., and Ross, R. (2015). Exploiting data locality in Swift / T workflows using Hercules.

Eker, J., Janneck, J., Lee, E., Jie Liu, Xiaojun Liu, Ludvig, J., Neuendorffer, S., Sachs, S., and Yuhong Xiong (2003). Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144. http://ieeexplore.ieee.org/document/1173203/ (accessed 2020-02-28).

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211. https://doi.org/10.1207/s15516709cog1402_1.

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99. http://www.sciencedirect.com/science/article/pii/0010027793900584 (accessed 2020-06-16).

EOSC (2019). EOSC: European Open Science Cloud. https://www.eosc-portal.eu/ (accessed 2019-05-22).

ESGF (2019). ESGF: Earth System Grid Federation. https://esgf.llnl.gov/index.html (accessed 2019-06-05).

ESiWACE (2016). Centre of Excellence in Simulation of Weather and Climate in Europe. https://www.esiwace.eu/ (accessed 2019-04-22).

Fangohr, H., Aplin, S., Barty, A., Beg, M., Bondar, V., Boukhelef, D., Brockhauser, S., Danilevski, C., Ehsan, W., Esenov, S., Flucke, G., Giovanetti, G., Goeries, D., Hauf, S., Heisen, B., Hickin, D., Khakhulin, D., Klimovskaia, A., Kluyver, T., Kuhn, M., Kuster, M., Lang, P.-M., Maia, L., Mariani, V., Mekinda, L., Michelat, T., Parenti, A., Previtali, G., Santos, H., Silenzi, A., Sztuk-Dambietz, J., Szuba, J., Teichmann, M., Weger, K., Wiggins, J., Wrona, K., and Xu, C. (2018). Data Analysis Support in Karabo at European XFEL. *Proceedings of the 16th Int. Conf. on Accelerator and Large Experimental Control Systems*, ICALEPCS2017:8 pages, 3.041 MB. http://jacow.org/icalepcs2017/doi/JACoW-ICALEPCS2017-TUCPA01.html (accessed 2020-01-10).

Fayek, H. M., Cavedon, L., and Wu, H. R. (2020). Progressive learning: A deep learning framework for continual learning. *Neural Networks*, 128:345–357. http://www.sciencedirect.com/science/article/pii/S0893608020301817 (accessed 2020-09-05).

Fleming, M. (2017). A thorough introduction to eBPF [LWN.net]. https://lwn.net/Articles/740157/ (accessed 2020-01-11).

Frank Röder (2019). Analysis of I/O Patterns. Technical report, Universität Hamburg.

Freund, Y. and Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139. http://www.sciencedirect.com/science/article/pii/S002200009791504X (accessed 2020-08-30).

Frigo, M. and Johnson, S. G. (2005). FFTW: C subroutine library for computing the discrete Fourier transform. http://www.fftw.org/ (accessed 2017-07-04).

Fujimoto, R. M. (1990). Performance of Time Warp under synthetic workloads. *Proceedings of the SCS Multiconference on Distributed Simulations, 1990*, 22(1):23–28. https://ci.nii.ac.jp/naid/10022194420/en/.

Ganglia Developers (2019). Ganglia Monitoring System. http://ganglia.info/ (accessed 2019-04-22).

Gao, J. (2014). Machine learning applications for data center optimization.

Geimer, M., Wolf, F., Wylie, B. J. N., Ábrahám, E., Becker, D., and Mohr, B. (2010). The Scalasca performance toolset architecture. *Concurr. Comput. Pract. Exp.*, 22(6). https://doi.org/10.1002/cpe.1556.

Glenn K. Lockwood, Shane Snyder, George Brown, Kevin Harms, Philip Carns, and Nicholas J. Wright (2018). TOKIO on ClusterStor: Connecting Standard Tools to Enable Holistic I/O Performance Analysis. In *In Proceedings of the 2018 Cray User Group*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*. http://arxiv.org/abs/1406.2661 (accessed 2017-12-17).

Google (2008). Protocol Buffers | Google Developers. https://developers.google.com/protocol-buffers (accessed 2020-10-27).

Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2.

Grafana Labs (2019). Grafana - The open platform for analytics and monitoring. https://github.com/grafana/grafana (accessed 2018-08-14).

Gregg, B. (2014). *Systems Performance: Enterprise and the Cloud*. Prentice Hall, Upper Saddle River, New Jersey.

Gregg, B. (2015). eBPF: One Small Step. http://www.brendangregg.com/blog/2015-05-15/ebpf-one-small-step.html (accessed 2020-01-11).

Gregg, B. (2019). Linux eBPF Tracing Tools. http://www.brendangregg.com/ebpf.html (accessed 2020-01-11).

Guest, M. (2012). *Prace: The Scientific Case for HPC in Europe*. Insight publishers, Bristol.

Guo, J., Nomura, A., Barton, R., Zhang, H., and Matsuoka, S. (2018). Machine learning predictions for underestimation of job runtime on HPC system. In Yokota, R. and Wu, W., editors, *Supercomputing Frontiers - 4th Asian Conference, SCFA 2018, Singapore, March 26-29, 2018, Proceedings*, volume 10776 of *Lecture Notes in Computer Science*, pages 179–198. Springer. https://doi.org/10.1007/978-3-319-69953-0_11.

Gupta, T. (2019). CS170 Lab 6: File System. https://sites.cs.ucsb.edu/~trinabh/classes/w19/labs/lab6.html (accessed 2020-04-27).

Hadian, A. and Heinis, T. (2019). Interpolation-friendly B-trees: Bridging the Gap Between Algorithmic and Learned Indexes. page 4.

Hammer Lab (2015). Monitoring Spark with Graphite and Grafana · Hammer Lab. http://www.hammerlab.org/2015/02/27/monitoring-spark-with-graphite-and-grafana/ (accessed 2018-08-16).

Hammond, J. (2019). Jhammond/lltop. https://github.com/jhammond/lltop (accessed 2019-09-06).

Haskins, K., Wofford, Q., and Bridges, P. G. (2019). Workflows for performance predictable and reproducible HPC applications. In *2019 IEEE International Conference on Cluster Computing, CLUSTER 2019, Albuquerque, NM, USA, September 23-26, 2019*, pages 1–2. IEEE. https://doi.org/10.1109/CLUSTER.2019.8891043.

HDF Group (2019). HDF5: Hierarchical Data Format. https://www.hdfgroup.org/hdf5/ (accessed 2017-07-04).

Hendrix, V., Fox, J., Ghoshal, D., and Ramakrishnan, L. (2016). Tigres Workflow Library: Supporting Scientific Pipelines on HPC Systems. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 146–155.

Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A., and Stanley, K. S. (2005). An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423.

HGST (2017a). Spec Sheet: Ultrastar-Hs14-DS. https://www.hgst.com/sites/default/files/resources/Ultrastar-Hs14-DS.pdf (accessed 2018-02-20).

HGST (2017b). Spec Sheet: Ultrastar-SN200. https://www.hgst.com/sites/default/files/resources/Ultrastar-SN200-Series-datasheet.pdf (accessed 2018-02-20).

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366. http://www.sciencedirect.com/science/article/pii/0893608089900208 (accessed 2020-07-26).

Howison, M., Koziol, Q., Knaak, D., Mainzer, J., and Shalf, J. (2012). Tuning HDF5 for Lustre File Systems. page 10.

Hruska, J. (2012). DARPA summons researchers to reinvent computing - ExtremeTech. http://www.extremetech.com/computing/116081-darpa-summons-researchers-to-reinvent-computing (accessed 2019-04-12).

Intel (2017a). QuickAssist Technology. https://01.org/intel-quickassist-technology (accessed 2020-03-24).

Intel (2017b). Spec Sheet: Optane SSD DC P4800X. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-ssd-dc-p4800x-brief.pdf (accessed 2018-02-20).

Intel, The HDF Group, EMC, and Cray (2014). Fast Forward Storage and I/O.

Intel Software (2019). MKL (Math Kernel Library). https://software.intel.com/en-us/intel-mkl (accessed 2017-05-11).

IO Visor (2020). XDP. https://www.iovisor.org/technology/xdp (accessed 2020-01-12).

iRODS Consortium (2019). iRODS. https://irods.org/ (accessed 2019-05-28).

ITRS (2015). International Technology Roadmap for Semiconductors - 2.0. Technical report.

Jaderberg, M., Czarnecki, W., Dunning, I., Marris, L., and Graepel, T. (2018a). Capture the Flag: The emergence of complex cooperative agents. https://deepmind.com/blog/capture-the-flag/ (accessed 2019-04-15).

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., and Graepel, T. (2018b). Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. *arXiv:1807.01281 [cs, stat]*. http://arxiv.org/abs/1807.01281 (accessed 2019-04-15).

jaegertracing.io (2020). Jaegertracing/jaeger-ui. Jaeger - Distributed Tracing Platform. https://github.com/jaegertracing/jaeger-ui (accessed 2020-02-11).

Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.-M., Hautier, G., Gunter, D., and Persson, K. A. (2015). FireWorks: A dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, 27(17):5037–5059. http://dx.doi.org/10.1002/cpe.3505.

Jain, A., Ong, S. P., Hautier, G., Chen, W., Richards, W. D., Dacek, S., Cholia, S., Gunter, D., Skinner, D., Ceder, G., and a. Persson, K. (2013). The Materials Project: A materials genome approach to accelerating materials innovation. *APL Materials*, 1(1):011002. http://link.aip.org/link/AMPADS/v1/i1/p011002/s1&Agg=doi.

Jenkins, J., Zou, X., Tang, H., Kimpe, D., Ross, R., and Samatova, N. F. (2014). RADAR: Runtime Asymmetric Data-Access Driven Scientific Data Replication. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Kobsa, A., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Terzopoulos, D., Tygar, D., Weikum, G., Kunkel, J. M., Ludwig, T., and Meuer, H. W., editors, *Supercomputing*, volume 8488, pages 296–313. Springer International Publishing, Cham. http://link.springer.com/10.1007/978-3-319-07518-1_19 (accessed 2020-01-05).

Jimenez, D. and Lin, C. (2001). Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206, Monterrey, Mexico. IEEE Comput. Soc. http://ieeexplore.ieee.org/document/903263/ (accessed 2019-06-10).

Jr., P. D. B., Carothers, C. D., Jefferson, D. R., and LaPre, J. M. (2013). Warp speed: Executing time warp on 1, 966, 080 cores. In Loper, M. L. and Wainer, G. A., editors, *SIGSIM Principles of Advanced Discrete Simulation, SIGSIM-PADS '13, Montreal, QC, Canada, May 19-22, 2013*, pages 327–336. ACM. https://doi.org/10.1145/2486092.2486134.

Karlsson, M. and Töpel, B. (2018). The Path to DPDK Speeds for AF_XDP.

Karniavoura, F. and Magoutis, K. (2019). Decision-making approaches for performance QoS in distributed storage systems: A survey. *IEEE Trans. Parallel Distrib. Syst.*, 30(8):1906–1919. https://doi.org/10.1109/TPDS.2019.2893940.

Kepler, P. (2016). The Kepler Project - Kepler. https://kepler-project.org/ (accessed 2018-08-16).

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*. http://arxiv.org/abs/1312.6114 (accessed 2020-02-10).

Kingston (2017). Spec Sheet: KSM24LQ4/64HAI. https://www.kingston.com/dataSheets/KSM24LQ4_64HAI.pdf (accessed 2018-02-20).

Kitware (2016). In situ | ParaView. https://www.paraview.org/in-situ/ (accessed 2019-06-04).

Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M. S., and Nagel, W. E. (2008). The vampir performance analysis tool-set. In Resch, M. M., Keller, R., Himmler, V., Krammer, B., and Schulz, A., editors, *Tools for High Performance Computing - Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008, HLRS, Stuttgart*. Springer. https://doi.org/10.1007/978-3-540-68564-7_9.

Knüpfer, A., Rössel, C., an Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A. D., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., and Wolf, F. (2011). Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, TAU, and vampir. In Brunst, H., Müller, M. S., Nagel, W. E., and Resch, M. M., editors, *Tools for High Performance Computing 2011 - Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing, ZIH, Dresden, September 2011*. Springer. https://doi.org/10.1007/978-3-642-31476-6_7.

Koethe, D. (2017). The DOE Exascale Computing Project. https://www.youtube.com/watch?v=5or4L41Vjl4.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. (2017). The Case for Learned Index Structures. In *arXiv:1712.01208 [Cs]*. http://arxiv.org/abs/1712.01208 (accessed 2017-12-11).

Król, D., da Silva, R. F., Deelman, E., and Lynch, V. E. (2017). Workflow Performance Profiles: Development and Analysis. In Desprez, F., Dutot, P.-F., Kaklamanis, C., Marchal, L., Molitorisz, K., Ricci, L., Scarano, V., Vega-Rodríguez, M. A., Varbanescu, A. L., Hunold, S., Scott, S. L., Lankes, S., and Weidendorfer, J., editors, *Euro-Par 2016: Parallel Processing Workshops*, volume 10104, pages 108–120. Springer International Publishing, Cham. http://link.springer.com/10.1007/978-3-319-58943-5_9 (accessed 2020-02-13).

Kubernetes (2017). Using eBPF in Kubernetes. https://kubernetes.io/blog/2017/12/using-ebpf-in-kubernetes/ (accessed 2020-01-12).

Lampa, S. (2014). Cheat sheets for working wirh the iRODS data system (Work in progress): Samuell/irods-cheatsheets. https://github.com/samuell/irods-cheatsheets (accessed 2019-05-23).

Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169. https://doi.org/10.1145/279227.279229.

Landry, T. (2019). Interoperable Machine Learning for Earth Observation and Climate in Federated Cyberinfrastructures.

LANL, NERSC, and SNL (2016). APEX Workflows. https://www.nersc.gov/assets/apex-workflows-v2.pdf (accessed 2018-06-12).

Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. (1979). Basic Linear Algebra Subprograms for Fortran Usage. *ACM Trans. Math. Softw.*, 5(3):308–323. http://doi.acm.org/10.1145/355841.355847 (accessed 2019-04-11).

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Denker, J. S., and Solla, S. A. (1989). Optimal brain damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 598–605. Morgan Kaufmann. http://papers.nips.cc/paper/250-optimal-brain-damage.

Leibovici, T. (2015). Taking back control of HPC file systems with Robinhood Policy Engine. *arXiv:1505.01448 [cs]*. http://arxiv.org/abs/1505.01448 (accessed 2020-02-13).

Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867. http://www.sciencedirect.com/science/article/pii/S0893608005801315 (accessed 2020-07-26).

Li, J., Liao, W.-k., Choudhary, A. N., Ross, R. B., Thakur, R., Gropp, W., Latham, R., Siegel, A. R., Gallagher, B., and Zingale, M. (2003). Parallel netCDF: A high-performance scientific I/O interface. In *Proceedings of the ACM/IEEE SC2003 Conference on High Performance Networking and Computing, 15-21 November 2003, Phoenix, AZ, USA, CD-Rom*, page 39. ACM. https://doi.org/10.1145/1048935.1050189.

Liu, Q., Logan, J., Tian, Y., Abbasi, H., Podhorszki, N., Choi, J. Y., Klasky, S., Tchoua, R., Lofstead, J., Oldfield, R., Parashar, M., Samatova, N., Schwan, K., Shoshani, A., Wolf, M., Wu, K., and Yu, W. (2014). Hello ADIOS: The Challenges and Lessons of Developing Leadership Class I/O Frameworks. *Concurr. Comput. : Pract. Exper.*, 26(7):1453–1473. http://dx.doi.org/10.1002/cpe.3125.

Liu, Z., Kettimuthu, R., Foster, I., and Beckman, P. H. (2018). Toward a smart data transfer node. *Future Generation Computer Systems*, 89:10–18. https://linkinghub.elsevier.com/retrieve/pii/S0167739X18302346 (accessed 2019-06-10).

LLNL (2015). SCR: Scalable Checkpoint/Restart for MPI. https://computing.llnl.gov/projects/scalable-checkpoint-restart-for-mpi (accessed 2020-02-06).

Lockwood, G. K., Wang, T., Byna, S., Wright, N. J., Snyder, S., and Carns, P. (2018). A year in the life of a parallel file system. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC18)*.

Lockwood, G. K., Yoo, W., Byna, S., Wright, N. J., Snyder, S., Harms, K., Nault, Z., and Carns, P. (2017). UMAMI: A Recipe for Generating Meaningful Metrics Through Holistic I/O Performance Analysis. In *Proceedings of the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, PDSW-DISCS '17, pages 55–60, New York, NY, USA. ACM. http://doi.acm.org/10.1145/3149393.3149395 (accessed 2018-01-13).

Lofstead, J. F., Klasky, S., Schwan, K., Podhorszki, N., and Jin, C. (2008). Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In Kim, Y. and Li, X., editors, *6th International Workshop on Challenges of Large Applications in Distributed Environments, CLADE@HPDC 2008, Boston, MA, USA, June 23, 2008*, pages 15–24. ACM. https://doi.org/10.1145/1383529.1383533.

Lordan, F., Tejedor, E., Ejarque, J., Rafanell, R., Álvarez, J., Marozzo, F., Lezzi, D., Sirvent, R., Talia, D., and Badia, R. M. (2014). ServiceSs: An Interoperable Programming Framework for the Cloud. *Journal of Grid Computing*, 12(1):67–91. http://link.springer.com/10.1007/s10723-013-9272-5 (accessed 2019-05-23).

Ludwig, T. and Geyer, B. (2019). Reproduzierbarkeit. *Informatik Spektrum*, 42(1):48–52. http://link.springer.com/10.1007/s00287-019-01149-2 (accessed 2019-06-03).

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

Lustre (2017). Kvaps/lustre. https://github.com/kvaps/lustre (accessed 2020-04-24).

Lüttgau, J. (2016). Modeling and simulation of tape libraries for hierarchical storage management systems. Master's thesis, Universität Hamburg.

Lüttgau, J., Betke, E., Perevalova, O., Kunkel, J., and Kuhn, M. (2017). Adaptive tier selection for NetCDF and HDF5. In *SC17: The International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, USA. https://sc17.supercomputing.org/SC17Archive/tech_poster/index.html.

Lüttgau, J., Kuhn, M., Duwe, K., Alforov, Y., Betke, E., Kunkel, J., and Ludwig, T. (2018a). A Survey of Storage Systems for High-Performance Computing. *Supercomputing Frontiers and Innovations*, pages 31–58. http://superfri.org/superfri/article/view/162.

Lüttgau, J. and Kunkel, J. (2017). Simulation of Hierarchical Storage Systems for TCO and QoS. In *High Performance Computing - ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^3MA, VHPC, Visualization at Scale, WOPSSS, Frankfurt, Germany, June 18-22, 2017, Revised Selected Papers*, pages 132–144. https://doi.org/10.1007/978-3-319-67630-2_12.

Lüttgau, J. and Kunkel, J. (2018). Cost and Performance Modeling for Earth System Data Management and Beyond. In Yokota, R., Weiland, M., Shalf, J., and Alam, S., editors, *High Performance Computing*, volume 11203, pages 23–35. Springer International Publishing, Cham. http://link.springer.com/10.1007/978-3-030-02465-9_2 (accessed 2020-01-13).

Lüttgau, J., Snyder, S., Carns, P., Wozniak, J. M., Kunkel, J., and Ludwig, T. (2018b). Toward Understanding I/O Behavior in HPC Workflows. In *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, Dallas, TX, USA. IEEE. https://ieeexplore.ieee.org/document/8638425/ (accessed 2019-04-23).

Madireddy, S., Balaprakash, P., Carns, P., Latham, R., Lockwood, G. K., Ross, R., Snyder, S., and Wild, S. M. (2019). Adaptive Learning for Concept Drift in Application Performance Modeling. In *Proceedings of the 48th International Conference on Parallel Processing - ICPP 2019*, pages 1–11, Kyoto, Japan. ACM Press. http://dl.acm.org/citation.cfm?doid=3337821.3337922 (accessed 2019-09-06).

Madireddy, S., Balaprakash, P., Carns, P., Latham, R., Ross, R., Snyder, S., and Wild, S. (2018a). Modeling I/O Performance Variability Using Conditional Variational Autoencoders. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 109–113, Belfast. IEEE. https://ieeexplore.ieee.org/document/8514864/ (accessed 2019-06-10).

Madireddy, S., Balaprakash, P., Carns, P. H., Latham, R., Ross, R. B., Snyder, S., and Wild, S. M. (2018b). Machine learning based parallel I/O predictive modeling: A case study on lustre file systems. In Yokota, R., Weiland, M., Keyes, D. E., and Trinitis, C., editors, *High Performance Computing - 33rd International Conference, ISC High Performance 2018, Frankfurt, Germany, June 24-28, 2018, Proceedings*, volume 10876 of *Lecture Notes in Computer Science*, pages 184–204. Springer. https://doi.org/10.1007/978-3-319-92040-5_10.

Mandal, A., Ruth, P., Baldin, I., Krol, D., Juve, G., Mayani, R., Silva, R. F. D., Deelman, E., Meredith, J., Vetter, J., Lynch, V., Mayer, B., Wynne, J., Blanco, M., Carothers, C., Lapre, J., and Tierney, B. (2016). Toward an End-to-End Framework for Modeling, Monitoring and Anomaly Detection for Scientific Workflows. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Chicago, IL, USA. IEEE. http://ieeexplore.ieee.org/document/7530026/ (accessed 2020-02-13).

Mapbox Inc. (2019). Map Tiles from Mapbox. https://www.mapbox.org (accessed 2019-12-04).

Marshall, W. (2017). Mission 1 Complete! https://www.planet.com/pulse/mission-1/ (accessed 2019-05-01).

Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Comput.*, 30(5-6):817–840. https://doi.org/10.1016/j.parco.2004.04.001.

McCanne, S. and Jacobson, V. (1993). The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the Usenix Winter 1993 Technical Conference, San Diego, California, USA, January 1993*, pages 259–270. USENIX Association. https://www.usenix.org/conference/usenix-winter-1993-conference/bsd-packet-filter-new-architecture-user-level-packet.

McCulloch, W. S. and Pitts, W. (1988). A logical calculus of the ideas immanent in nervous activity. In *Neurocomputing: Foundations of Research*, pages 15–27. MIT Press, Cambridge, MA, USA.

McInnes, L., Healy, J., and Melville, J. (2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv:1802.03426 [cs, stat]*. http://arxiv.org/abs/1802.03426 (accessed 2020-02-10).

McKee, S. A. (2004). Reflections on the Memory Wall. In *Proceedings of the 1st Conference on Computing Frontiers*, CF '04, pages 162–, New York, NY, USA. ACM. http://doi.acm.org/10.1145/977091.977115 (accessed 2017-05-11).

Mell, P. and Grance, T. (2011). 800-145: The NIST definition of cloud computing. http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf (accessed 2016-08-02).

Mellanox (2018). ConnectX-5 VPI IC. https://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX-5_VPI_IC.pdf (accessed 2020-02-09).

Mendez, S. (2019). Best Practice Guide - Parallel I/O. page 55.

MGI (2011). Materials Genome Initiative. https://www.mgi.gov/ (accessed 2019-12-06).

Michelogiannakis, G., Shalf, J., and Vasudevan, D. (2019). PARADISE. https://crd.lbl.gov/departments/computer-science/cag/research/paradise/ (accessed 2019-03-06).

Miles, S., Groth, P., Deelman, E., Vahi, K., Mehta, G., and Moreau, L. (2008). Provenance: The Bridge Between Experiments and Data. *Computing in Science Engineering*, 10(3):38–46.

Minsky, M. L. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press. https://www.amazon.com/Perceptrons-Introduction-Computational-Geometry-Minsky/dp/0262130432?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0262130432.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2017). Pruning Convolutional Neural Networks for Resource Efficient Inference. *arXiv:1611.06440 [cs, stat]*. http://arxiv.org/abs/1611.06440 (accessed 2020-02-17).

Molnar, I. (2015). LKML: Ingo Molnar: [GIT PULL] perf updates for v4.1. https://lkml.org/lkml/2015/4/14/232 (accessed 2020-01-11).

Monjalet, F. and Leibovici, T. (2019). Predicting File Lifetimes with Machine Learning. In Weiland, M., Juckeland, G., Alam, S., and Jagode, H., editors, *High Performance Computing*, Lecture Notes in Computer Science, pages 288–299, Cham. Springer International Publishing.

MPI Forum (2019). MPI (Massage Passing Interface). http://mpi-forum.org/ (accessed 2017-07-04).

Mubarak, M., Carothers, C. D., Ross, R. B., and Carns, P. H. (2014). A case study in using massively parallel simulation for extreme-scale torus network codesign. In Jr., J. A. H., Riley, G. F., and Fujimoto, R. M., editors, *SIGSIM Principles of Advanced Discrete Simulation, SIGSIM-PADS '14, Denver, CO, USA, May 18-21, 2014*, pages 27–38. ACM. https://doi.org/10.1145/2601381.2601383.

Nagios Enterprise (2019). Nagios - The Industry Standard In IT Infrastructure Monitoring. https://www.nagios.org/ (accessed 2018-08-14).

NCBI (2020). BLAST: Basic Local Alignment Search Tool. https://blast.ncbi.nlm.nih.gov/Blast.cgi (accessed 2020-01-25).

Nellans, D., Kadaru, V. K., and Brunvand, E. (2004). *ASIM – An Asynchronous Architectural Level Simulator*.

NERSC (2017). TaskFarmer. http://www.nersc.gov/users/data-analytics/workflow-tools/taskfarmer/ (accessed 2018-05-21).

Newhouse, S. (2011). EGI-InSPIRE: European Grid Infrastructure – an Integrated Sustainable Pan-European Infrastructure for Researchers in Europe. https://documents.egi.eu/public/RetrieveFile?docid=201&version=6&filename=EGI-paper-D23-v7.pdf (accessed 2019-05-29).

NEXTGenIO (2018). NEXTGenIO: Next Generation I/O for the Exascale. http://www.nextgenio.eu/ (accessed 2017-07-04).

Nijholt, B., Weston, J., Hoofwijk, J., and Akhmerov, A. (2019). Adaptive: Parallel active learning of mathematical functions.

NOPP (2009). MEDEA Program Overview. https://www.nopp.org/wp-content/uploads/2010/06/85.pdf (accessed 2019-09-01).

Oeste, S., Kluge, M., Soysal, M., Streit, A., Vef, M.-A., and Brinkmann, A. (2016). Exploring Opportunities for Job-temporal File Systems with ADA-FS. page 5.

OFIWG (2020). Libfabric OpenFabrics. https://ofiwg.github.io/libfabric/ (accessed 2019-09-05).

Oliver, H. J., Shin, M., Fitzpatrick, B., Clark, A., Sanders, O., Bartholomew, S. L., Valters, D., Kinoshita, B. P., Kerry Smout-Day, challurip, Trzeciak, T., Matthews, D., Wales, S., lhuggett, Williams, J., wxtim, Hatcher, R., Osprey, A., Reinecke, A., ivorblockley, Dix, M., Pulo, K., and Huang, A. (2019). Cylc/cylc: Cylc-7.8.1. Zenodo. https://zenodo.org/record/2549475 (accessed 2019-04-11).

ONXX (2019). Open Neural Network Exchange (ONNX). https://onnx.ai/ (accessed 2020-11-01).

Open Grid Computing Inc. (2019). OVIS High Performance Computing monitoring, analysis, and visualization project. NOTE: V4 is a branch (not master).: Ovis-hpc/ovis. https://github.com/ovis-hpc/ovis (accessed 2019-04-22).

OpenMP (1997). OpenMP. http://www.openmp.org/ (accessed 2017-07-04).

OpenStack (2013). Mistral: Workflow service for OpenStack. https://www.openstack.org/software/releases/train/components/mistral (accessed 2020-02-08).

OpenStack (2018). OpenStack Docs: Design/Logical Architecture. https://docs.openstack.org/arch-design/design.html (accessed 2020-02-09).

OpenStreetMap Contributors (2019). Map Data received from OpenStreetMap. https://www.openstreetmap.org (accessed 2019-12-04).

OpenTelemetry (2020). OpenTelemetry. https://opentelemetry.io/ (accessed 2020-10-31).

OpenTracing (2018). The OpenTracing project. https://opentracing.io/ (accessed 2020-10-31).

OpenUCX (2019). UCX Documentation. https://www.openucx.org/documentation/ (accessed 2021-04-30).

OpenUCX (2021). Openucx/ucx. UCX. https://github.com/openucx/ucx (accessed 2021-04-28).

OpenZipkin (2015). OpenZipkin · A distributed tracing system. https://zipkin.io/ (accessed 2020-02-11).

ORNL (2017). ADIOS. https://www.olcf.ornl.gov/center-projects/adios/ (accessed 2017-07-04).

Palazzo, C., Mariello, A., Fiore, S., D'Anca, A., Elia, D., Williams, D. N., and Aloisio, G. (2015). A workflow-enabled big data analytics software stack for escience. In *2015 International Conference on High Performance Computing Simulation (HPCS)*, pages 545–552.

Peterka, T., Bard, D., Bennett, J., Bethel, E. W., Oldfield, R., Pouchard, L., Sweeney, C., and Wolf, M. (2019). ASCR Workshop on In Situ Data Management: Enabling Scientific Discovery from Diverse Data Sources. Technical report, USDOE Office of Science (SC) (United States). https://www.osti.gov/biblio/1493245-ascr-workshop-situ-data-management-enabling-scientific-discovery-from-diverse-data-sources (accessed 2020-03-10).

Peterka, T., Cappello, F., and Lofstead, J. (2018). Decaf: High-Performance Decoupling of Tightly Coupled Flows | Argonne National Laboratory. http://www.mcs.anl.gov/project/decaf-high-performance-decoupling-tightly-coupled-flows (accessed 2018-06-21).

Pietri, I., Juve, G., Deelman, E., and Sakellariou, R. (2014). A Performance Model to Estimate Execution Time of Scientific Workflows on the Cloud. In *2014 9th Workshop on Workflows in Support of Large-Scale Science*, pages 11–19.

Plimpton, S. (1995). Fast Parallel Algorithms for Short–Range Molecular Dynamics. page 42.

Pouchard, L., Baldwin, S., Elsethagen, T., Gamboa, C., Jha, S., Raju, B., Stephan, E., Tang, L., and Van Dam, K. K. (2018). Use Cases of Computational Reproducibility for Scientific Workflows at Exascale. *arXiv:1805.00967 [cs]*. http://arxiv.org/abs/1805.00967 (accessed 2019-06-03).

Povh, J. (2016). EINFRA-11-2016: Support to the next implementation phase of Pan- European High Performance Computing Infrastructure and Services (PRACE).

PRACE (2013). Performance prediction. http://www.prace-ri.eu/performance-prediction/ (accessed 2019-04-11).

ProfiT-HPC (2017). Profiling Toolkit for HPC Applications. https://profit-hpc.de/ (accessed 2021-05-12).

Prometheus Authors (2019a). Prometheus - Monitoring system & time series database. https://prometheus.io/ (accessed 2019-04-21).

Prometheus Authors (2019b). Prometheus - Monitoring system & time series database. https://github.com/prometheus (accessed 2019-04-21).

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). ROS: An open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434 [cs]*. http://arxiv.org/abs/1511.06434 (accessed 2020-08-16).

Raffin, B. and Dreher, M. (2014). FlowVR: Download. http://flowvr.sourceforge.net (accessed 2019-05-27).

Redmon, J. and Farhadi, A. (2016). YOLO9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*.

Redmon, J. and Farhadi, A. (2018). YOLOv3: An incremental improvement. *arXiv*.

Regenscheid, A. (2018). Platform firmware resiliency guidelines. Technical Report NIST SP 800-193, National Institute of Standards and Technology, Gaithersburg, MD. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-193.pdf (accessed 2019-03-07).

Rew, R. and Davis, G. (1990). NetCDF: An interface for scientific data access. *IEEE Computer Graphics and Applications*, 10(4):76–82.

Rocklin, M. (2015). Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Python in Science Conference*, pages 126–132, Austin, Texas. https://conference.scipy.org/proceedings/scipy2015/matthew_rocklin.html (accessed 2019-03-12).

Rodrigues, A. F., CooperBalls, E., Jacob, B., Hemmert, K. S., Barrett, B. W., Kersey, C., Oldfield, R., Weston, M., Risen, R., Cook, J., and Rosenfeld, P. (2011). The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37. http://portal.acm.org/citation.cfm?doid=1964218.1964225 (accessed 2018-03-25).

Rosenblatt, F. (1957). *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory. https://books.google.de/books?id=P_XGPgAACAAJ.

Ross, R. B., Amvrosiadis, G., Carns, P., Cranor, C. D., Dorier, M., Harms, K., Ganger, G., Gibson, G., Gutierrez, S. K., Latham, R., Robey, B., Robinson, D., Settlemyer, B., Shipman, G., Snyder, S., Soumagne, J., and Zheng, Q. (2020). Mochi: Composing Data Services for High-Performance Computing Environments. *Journal of Computer Science and Technology*, 35(1):121–144. http://link.springer.com/10.1007/s11390-020-9802-0 (accessed 2020-02-19).

Ruprecht, D., Emmett, M., and Speck, R. (2019). PinT: Parallel in Time. https://parallel-in-time.org/ (accessed 2019-05-28).

Santoalla, D. V. and Bonet, A. (2019). ecFlow - Workflow Package to Run Large Numbers of Programs. https://confluence.ecmwf.int/display/ECFLOW/ecflow+home (accessed 2019-03-09).

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227. https://doi.org/10.1007/BF00116037 (accessed 2020-08-30).

SchedMD (2019a). Slurm Workload Manager - Documentation. https://slurm.schedmd.com/ (accessed 2019-05-22).

SchedMD (2019b). Slurm Workload Manager - Slurm Burst Buffer Guide. https://slurm.schedmd.com/burst_buffer.html (accessed 2020-02-10).

Score-P Developer Community (2019). Scalable performance measurement infrastructure for parallel codes (Score-P). https://zenodo.org/record/3356706 (accessed 2020-02-13).

Seagate (2017). Spec Sheet: Nytro3000. https://www.seagate.com/files/www-content/datasheets/pdfs/nytro-3000-sas-ssdDS1950-2-1711DE-de_DE.pdf (accessed 2018-02-20).

Seagate (2018). Spec Sheet: Nytro 5910. https://www.seagate.com/files/www-content/datasheets/pdfs/nytro-5910-nvme-ssdDS1953-3-1801DE-de_DE.pdf (accessed 2018-02-20).

Segers, P. and Nominé, J.-P. (2017). D2.5 Updated Stakeholder Management in PRACE 2. http://www.prace-ri.eu/IMG/pdf/D2.5_4ip.pdf (accessed 2019-04-11).

Shainer, G. (2011). MPI Collectives Offloads. https://downloads.openfabrics.org/Media/Monterey_2011/Apr4_mpicollectivesoffloads.pdf (accessed 2020-02-09).

Shamis, P., Venkata, M. G., Lopez, M. G., Baker, M. B., Hernandez, O., Itigin, Y., Dubman, M., Shainer, G., Graham, R. L., Liss, L., et al. (2015). UCX: An open source framework for HPC network APIs and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 40–43. IEEE.

Shehabi, A., Smith, S., Sartor, D., Brown, R., Herrlin, M., Koomey, J., Masanet, E., Horner, N., Azevedo, I., and Lintner, W. (2016). United States Data Center Energy Usage Report. Technical Report LBNL–1005775, 1372902. http://www.osti.gov/servlets/purl/1372902/ (accessed 2018-12-07).

Shende, S. and Malony, A. D. (2006). The tau parallel performance system. *IJHPCA*, 20(2):287–311. https://doi.org/10.1177/1094342006064482.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The Hadoop Distributed File System. In Khatib, M. G., He, X., and Factor, M., editors, *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7, 2010*, pages 1–10. IEEE Computer Society. https://doi.org/10.1109/MSST.2010.5496972.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144. http://www.sciencemag.org/lookup/doi/10.1126/science.aar6404 (accessed 2019-04-13).

Sim, H., Vallée, G., Kim, Y., Vazhkudai, S. S., Tiwari, D., and Butt, A. R. (2019). An analysis workflow-aware storage system for multi-core active flash arrays. *IEEE Trans. Parallel Distrib. Syst.*, 30(2):271–285. https://doi.org/10.1109/TPDS.2018.2865471.

Sonnenburg, S., Braun, M. L., Ong, C. S., Bengio, S., Bottou, L., Holmes, G., LeCun, Y., Muller, K.-R., Pereira, F., Rasmussen, C. E., Ratsch, G., Scholkopf, B., Smola, A., Vincent, P., Weston, J., and Williamson, R. C. (2007). The Need for Open Source Software in Machine Learning. page 25.

Soumagne, J., Kimpe, D., Zounmevo, J., Chaarawi, M., Koziol, Q., Afsahi, A., and Ross, R. (2013). Mercury: Enabling remote procedure call for high-performance computing. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8.

spack.io (2020). Spack. https://spack.io/ (accessed 2020-01-17).

Spafford, K. and Vetter, J. S. (2012). Aspen: A domain specific language for performance modeling. In Hollingsworth, J. K., editor, *SC Conference on High Performance Computing Networking, Storage and Analysis, SC '12, Salt Lake City, UT, USA - November 11 - 15, 2012*, page 84. IEEE/ACM. https://doi.org/10.1109/SC.2012.20.

Spark, A. (2018). RDD Lineage — Logical Execution Plan · Mastering Apache Spark. https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd-lineage.html (accessed 2018-06-15).

Spotify (2019). Luigi is a Python module that helps you build complex pipelines of batch jobs. It handles dependency resolution, workflow management, visualization etc. It also comes with Hadoop support built in. .. Spotify. https://github.com/spotify/luigi (accessed 2019-05-27).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Stanford-Clark, A. and Nipper, A. (1999). MQTT (MQ Telemetry Transport). http://mqtt.org/ (accessed 2019-02-13).

Stukowski, A. (2009). Visualization and analysis of atomistic simulation data with OVITO–the Open Visualization Tool. *Modelling and Simulation in Materials Science and Engineering*, 18(1):015012. https://doi.org/10.1088/0965-0393/18/1/015012 (accessed 2021-04-25).

Sun, Q., Parashar, M., Jin, T., Romanus, M., Bui, H., Zhang, F., Yu, H., Kolla, H., Klasky, S., and Chen, J. (2015). Adaptive data placement for staging-based coupled scientific workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15*, pages 1–12, Austin, Texas. ACM Press. http://dl.acm.org/citation.cfm?doid=2807591.2807669 (accessed 2020-01-05).

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *arXiv:1409.3215 [cs]*. http://arxiv.org/abs/1409.3215 (accessed 2020-07-26).

Swift Team (2015). Swift User Guide: Section 5.3: Monitoring Swift. http://swift-lang.org/guides/release-0.96/userguide/userguide.html.

Tejedor, E., Becerra, Y., Alomar, G., Queralt, A., Badia, R. M., Torres, J., Cortes, T., and Labarta, J. (2017). PyCOMPSs: Parallel computational workflows in Python. *The International Journal of High Performance Computing Applications*, 31(1):66–82. https://doi.org/10.1177/1094342015594678 (accessed 2019-05-22).

Top500 (2019). Top500 Supercomputing Sites. http://www.top500.org/.

Toshiba (2017). Spec Sheet: eSSD-PX05SHB. https://toshiba.semicon-storage.com/content/dam/toshiba-ss/asia-pacific/docs/product/storage/product-manual/eSSD-PX05SHB-product-overview.pdf (accessed 2018-02-20).

Treibig, J., Hager, G., and Wellein, G. (2010). LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. *arXiv:1004.4431 [cs]*. http://arxiv.org/abs/1004.4431 (accessed 2020-03-02).

Tuncer, O., Ates, E., Zhang, Y., Turk, A., Brandt, J. M., Leung, V. J., Egele, M., and Coskun, A. K. (2019). Online diagnosis of performance variation in HPC systems using machine learning. *IEEE Trans. Parallel Distrib. Syst.*, 30(4):883–896. https://doi.org/10.1109/TPDS.2018.2870403.

UCSC (2017). Toil architecture — Toil 3.7.1a1 documentation. https://toil.readthedocs.io/en/releases-3.7.x/architecture.html (accessed 2020-02-03).

UCSC CGL (2017). Toil. http://toil.ucsc-cgl.org/ (accessed 2019-05-22).

Ulmer, C., Mukherjee, S., Templet, G., Levy, S., Lofstead, J. F., Widener, P. M., Kordenbrock, T., and Lawson, M. (2018). Faodel: Data management for next-generation application workflows. In Costan, A., Nicolae, B., Duplyakin, D., II, G. F. L., and Dayal, J., editors, *Proceedings of the 9th Workshop on Scientific Cloud Computing, ScienceCloud@HPDC 2018, Tempe, AZ, USA, June 11, 2018*, pages 8:1–8:6. ACM. https://doi.org/10.1145/3217880.3217888.

Unidata (2019). NetCDF: Network Common Data Form. https://www.unidata.ucar.edu/software/netcdf/ (accessed 2017-07-04).

Unidata (2020). Software for Manipulating or Displaying NetCDF Data. https://www.unidata.ucar.edu/software/netcdf/software.html (accessed 2020-03-20).

Urquizo, A. (2011). PID controller overview. https://commons.wikimedia.org/wiki/File:PID_en.svg (accessed 2020-08-08).

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605. http://www.jmlr.org/papers/v9/vandermaaten08a.html.

Vef, M.-A., Moti, N., Süß, T., Tacke, M., Tocci, T., Nou, R., Miranda, A., Cortes, T., and Brinkmann, A. (2020). GekkoFS — A Temporary Burst Buffer File System for HPC Applications. *Journal of Computer Science and Technology*, 35(1):72–91. https://doi.org/10.1007/s11390-020-9797-6 (accessed 2020-02-23).

Verbitskiy, I., Thamsen, L., Renner, T., and Kao, O. (2018). CoBell: Runtime prediction for distributed dataflow jobs in shared clusters. In *2018 IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2018, Nicosia, Cyprus, December 10-13, 2018*, pages 81–88. IEEE Computer Society. https://doi.org/10.1109/CloudCom2018.2018.00029.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning - ICML '08*, pages 1096–1103, Helsinki, Finland. ACM Press. http://portal.acm.org/citation.cfm?doid=1390156.1390294 (accessed 2020-02-10).

Wadhwa, B., Paul, A. K., Neuwirth, S., Wang, F., Oral, S., Butt, A. R., Bernard, J., and Cameron, K. W. (2019). Iez: Resource Contention Aware Load Balancing for Large-Scale Parallel File Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 610–620.

Wang, H., Fu, X., Xu, J., and Lu, H. (2019). Learned index for spatial queries. In *20th IEEE International Conference on Mobile Data Management, MDM 2019, Hong Kong, SAR, China, June 10-13, 2019*, pages 569–574. IEEE. https://doi.org/10.1109/MDM.2019.00121.

Wang, T., Byna, S., Dong, B., and Tang, H. (2018a). UniviStor: Integrated Hierarchical and Distributed Storage for HPC. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 134–144.

Wang, T., Snyder, S., Lockwood, G., Carns, P., Wright, N., and Byna, S. (2018b). IOMiner: Large-Scale Analytics Framework for Gaining Knowledge from I/O Logs. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 466–476, Belfast. IEEE. https://ieeexplore.ieee.org/document/8514906/ (accessed 2019-06-10).

Wang, Y., Lu, P., and Kent, K. B. (2015). WaFS: A Workflow-Aware File System for Effective Storage Utilization in the Cloud. *IEEE Transactions on Computers*, 64(9):2716–2729.

Wang, Z. J., Turko, R., Shaikh, O., Park, H., Das, N., Hohman, F., Kahng, M., and Chau, D. H. (2020). CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization. *arXiv:2004.15004 [cs]*. http://arxiv.org/abs/2004.15004 (accessed 2020-06-30).

Weil, S., Brandt, S., Miller, E., and Maltzahn, C. (2006). CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In *ACM/IEEE SC 2006 Conference (SC'06)*, pages 31–31, Tampa, FL, USA. IEEE. http://ieeexplore.ieee.org/document/4090205/ (accessed 2020-04-24).

Weiner, J. (2018). Psi: Pressure stall information for CPU, memory, and IO v2 [LWN.net]. https://lwn.net/Articles/759658/ (accessed 2019-06-11).

Welford, B. P. (1962). Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics*, 4(3):419–420. https://amstat.tandfonline.com/doi/abs/10.1080/00401706.1962.10490022 (accessed 2020-06-10).

Widener, P. M., Ulmer, C., Levy, S., Kordenbrock, T., and Templet, G. (2019). Mediating data center storage diversity in HPC applications with FAODEL. In Weiland, M., Juckeland, G., Alam, S. R., and Jagode, H., editors, *High Performance Computing - ISC High Performance 2019 International Workshops, Frankfurt, Germany, June 16-20, 2019, Revised Selected Papers*, volume 11887 of *Lecture Notes in Computer Science*, pages 275–287. Springer. https://doi.org/10.1007/978-3-030-34356-9_22.

Wikipedia contributors (2020). List of datasets for machine-learning research. *Wikipedia*. https://en.wikipedia.org/w/index.php?title=List_of_datasets_for_machine-learning_research&oldid=966061832 (accessed 2020-08-09).

Wilde, M., Hategan, M., Wozniak, J. M., Clifford, B., Katz, D. S., and Foster, I. (2011). Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., Gonzalez-Beltran, A., Gray, A. J., Groth, P., Goble, C., Grethe, J. S., Heringa, J., 't Hoen, P. A., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S. J., Martone, M. E., Mons, A., Packer, A. L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M. A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., and Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4792175/ (accessed 2019-12-06).

Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., Bhagat, J., Belhajjame, K., Bacall, F., Hardisty, A., Nieva de la Hidalga, A., Balcazar Vargas, M. P., Sufi, S., and Goble, C. (2013). The Taverna workflow suite: Designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561. https://academic.oup.com/nar/article/41/W1/W557/1094153 (accessed 2019-05-27).

Wozniak, J. M., Armstrong, T. G., Maheshwari, K., Lusk, E. L., Katz, D. S., Wilde, M., and Foster, I. T. (2013a). Turbine: A distributed-memory dataflow engine for high performance many-task applications. *Fundamenta Informaticae*, 28(3).

Wozniak, J. M., Armstrong, T. G., Wilde, M., Katz, D. S., Lusk, E., and T. Foster, I. (2013b). Swift/T: Scalable data flow programming for distributed-memory task-parallel applications. In *Proc. CCGrid*.

Wozniak, J. M., Chan, A., G. Armstrong, T., Wilde, M., Lusk, E., and Foster, I. T. (2013c). A model for tracing and debugging large-scale task-parallel programs with MPE. In *Proc. Workshop on Leveraging Abstractions and Semantics in High-performance Computing (LASH-C) at PPoPP*.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2020). A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21. http://arxiv.org/abs/1901.00596 (accessed 2020-08-15).

Xu, C., Snyder, S., Kulkarni, O., Venkatesan, V., Carns, P., Byna, S., Sisneros, R., and Chadalavada, K. (2017). DXT: Darshan eXtended Tracing.

Yildiz, O., Ejarque, J., Chan, H., Sankaranarayanan, S., Badia, R. M., and Peterka, T. (2019). Heterogeneous Hierarchical Workflow Composition. *Computing in Science & Engineering*, 21(4):76–86. https://ieeexplore.ieee.org/document/8723160/ (accessed 2019-07-17).

Zhao, D., Zhang, Z., Zhou, X., Li, T., Wang, K., Kimpe, D., Carns, P., Ross, R., and Raicu, I. (2014). FusionFS: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 61–70, Washington, DC, USA. IEEE. http://ieeexplore.ieee.org/document/7004214/ (accessed 2020-02-20).

Zilberman, N., Watts, P. M., Rotsos, C., and Moore, A. W. (2015). Reconfigurable Network Systems and Software-Defined Networking. *Proceedings of the IEEE*, 103(7):1102–1124.

Quick Reference for Acronyms and Terms

Appendix

A.1 List of Publications

- Lüttgau, J. and Kunkel, J. (2017). Simulation of Hierarchical Storage Systems for TCO and QoS. In *High Performance Computing - ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^3MA, VHPC, Visualization at Scale, WOPSSS, Frankfurt, Germany, June 18-22, 2017, Revised Selected Papers*, pages 132–144. https://doi.org/10.1007/978-3-319-67630-2_12

- Lüttgau, J., Betke, E., Perevalova, O., Kunkel, J., and Kuhn, M. (2017). Adaptive tier selection for NetCDF and HDF5. In *SC17: The International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, USA. https://sc17.supercomputing.org/SC17Archive/tech_poster/index.html

- Lüttgau, J. and Kunkel, J. (2018). Cost and Performance Modeling for Earth System Data Management and Beyond. In Yokota, R., Weiland, M., Shalf, J., and Alam, S., editors, *High Performance Computing*, volume 11203, pages 23–35. Springer International Publishing, Cham. http://link.springer.com/10.1007/978-3-030-02465-9_2 (accessed 2020-01-13)

- Lüttgau, J., Snyder, S., Carns, P., Wozniak, J. M., Kunkel, J., and Ludwig, T. (2018b). Toward Understanding I/O Behavior in HPC Workflows. In *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, Dallas, TX, USA. IEEE. https://ieeexplore.ieee.org/document/8638425/ (accessed 2019-04-23)

- Lüttgau, J., Kuhn, M., Duwe, K., Alforov, Y., Betke, E., Kunkel, J., and Ludwig, T. (2018a). A Survey of Storage Systems for High-Performance Computing. *Supercomputing Frontiers and Innovations*, pages 31–58. http://superfri.org/superfri/article/view/162

A.2 Map of Various Machine Learning Methods

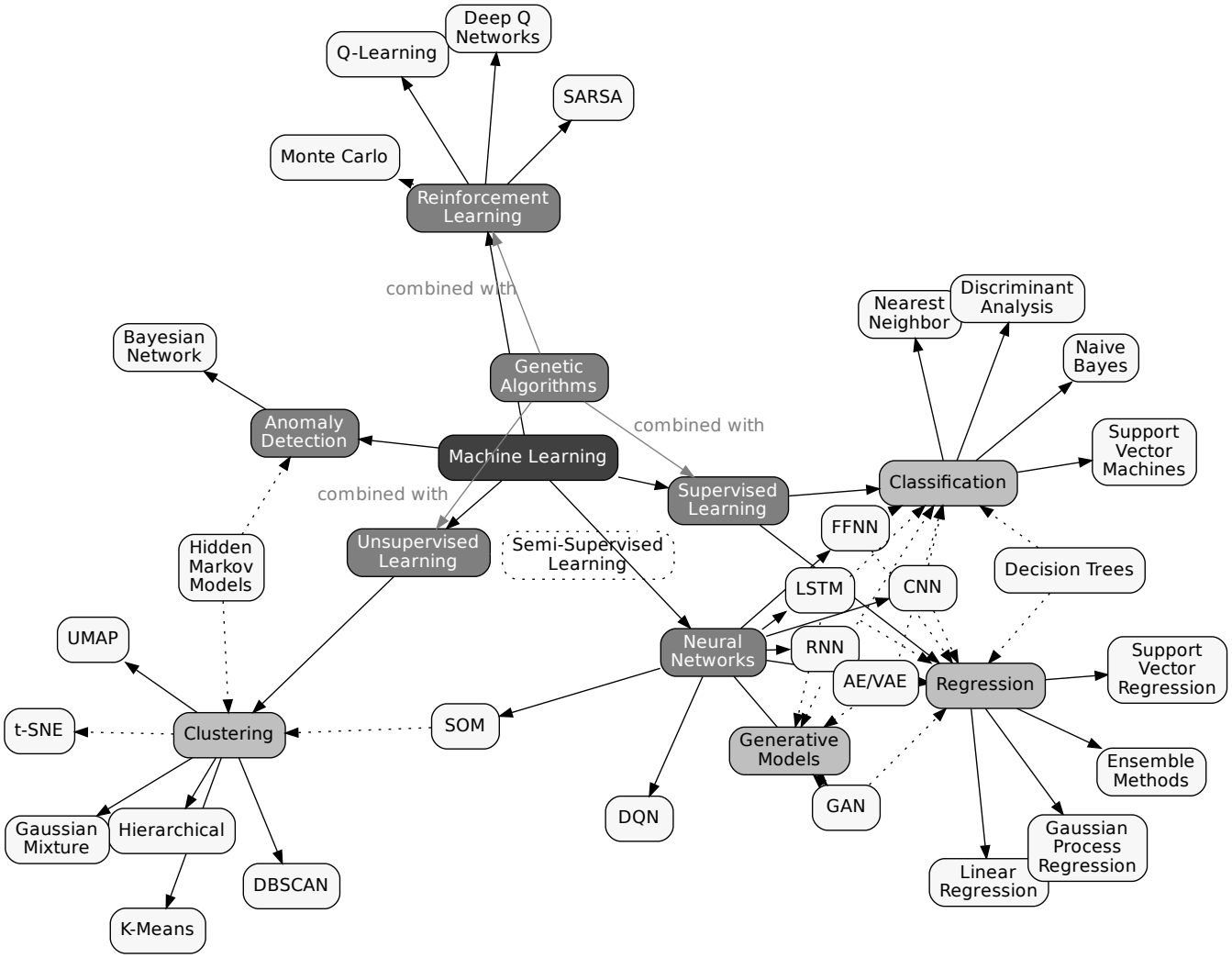


Figure A.1: Overview of various methods in machine learning and their relationship.

A.3 Distribution of Supercomputers Worldwide (2020)

Figure A.2: The Top500 supercopmuting sites as of November 2020. Cartographic data courtesy of `openstreetmap.org`, rendered with tiles from Mapbox using Leaflet to overlay the sites. A sites diameter is determined based on data provided by the Top500 Supercomputing list at `top500.org`.

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den Unterschrift