Lawrence Berkeley National Laboratory

LBL Publications

Title

Investigating User Experiences with Data Abstractions on High Performance Computing Systems

Permalink

https://escholarship.org/uc/item/1j93t5gm

Authors

Paine, Drew

Poon, Sarah

Ramakrishnan, Lavanya

Publication Date

2021-06-30

DOI

10.2172/1805039

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Investigating User Experiences with Data Abstractions on High Performance Computing Systems

Drew Paine, Sarah Poon, and Lavanya Ramakrishnan
Data Science and Technology Department
Lawrence Berkeley National Laboratory
{pained, sspoon, lramakrishnan}@lbl.gov

June 2021

Abstract

Scientific exploration generates expanding volumes of data that commonly require High Performance Computing (HPC) systems to facilitate research. HPC systems are complex ecosystems of hardware and software that frequently are not user friendly. The Usable Data Abstractions (UDA) project set out to build usable software for scientific workflows in HPC environments by undertaking multiple rounds of qualitative user research. Qualitative research investigates how individuals accomplish their work and our interview-based study surfaced a variety of insights about the experiences of working in and with HPC ecosystems. This report examines multiple facets to the experiences of scientists and developers using and supporting HPC systems. We discuss how stakeholders grasp the design and configuration of these systems, the impacts of abstraction layers on their ability to successfully do work, and the varied perceptions of time that shape this work. Examining the adoption of the Cori HPC at NERSC we explore the anticipations and lived experiences of users interacting with this system's novel storage feature, the Burst Buffer. We present lessons learned from across these insights to illustrate just some of the challenges HPC facilities and their stakeholders need to account for when procuring and supporting these essential scientific resources to ensure their usability and utility to a variety of scientific practices.

Keywords — user research, qualitative methods, HPC system, abstraction layers, Burst Buffer
Report Number: LBNL-2001374

1 Introduction

Scientific exploration for decades has produced and relied upon expanding and evolving volumes of data, and improved fidelity and accuracy of simulations. Scientist's ability to scale their work up, typically as part of large projects, frequently requires High Performance Computing (HPC) systems to provide the necessary computing, storage, and networking resources to work with these big datasets. HPC systems are complex computational ecosystems, particularly contrasted with personal computers, that perpetually incorporate new technologies and practices. Novice and experienced users alike can find it challenging to easily utilize different HPC systems in their work due to complex abstraction layers, the need to craft work practices and software to function with high amounts of parallelism, and ongoing changes to the organization and operation of these resources. Building usable, user friendly tools for working with large datasets in HPC environments requires understanding how scientists do their work now as well as how they understand and conceptualize different computing environments as resources. Only by investigating the actual work of scientists using computing systems can we understand how to support scientists and build usable software systems [16].

The Usable Data Abstractions (UDA) project was a multi-year study of these challenges. We investigated how scientists manage data in evolving HPC environments through multiple rounds of semi-structured interviews in our qualitative user research. We engaged with staff at an HPC facility, developers building data management and analysis tools, and domain scientists from a variety of backgrounds over multiple years. Our interviews enabled us to better understand how scientists work across multiple HPC systems and their storage hierarchies, as well as how work changed as a facility transitioned to a new HPC system. We drew upon insights from this work to design new abstractions and workflow systems to support changing scientific work [4].

This report discusses multiple facets to user's experiences working in HPC ecosystems and offers lessons learned for stakeholders who are interested in better supporting the scientific work these resources are built to facilitate. We explore how users grasp the construction of HPC ecosystems as resources for their work, particularly the ways abstraction layers impact their experiences processing and analyzing data. A key facet to these experiences are user's multiple perceptions of time and the ways different components of an HPC impact this experience [3]. Examining the introduction of a new HPC at the National Energy Research Scientific Computing Center (NERSC), we discuss how a next generation storage system was anticipated by stakeholders then experienced in practice. Our insights illustrate a few of the complexities around working in HPC ecosystems and demonstrate concerns that stakeholders should take into account as they design entire systems, software tools, policies, and practices.

2 Background

Our investigation engaged with stakeholders from across the NERSC facility. To situate our findings we first characterize ecosystems of HPC facilities to illustrate some of the varied roles stakeholders hold, the technical resources commonly made available, and some characteristics of working in such environments. We then discuss systems and resources available at the NERSC facility. Finally, we discuss concepts from human computer interaction and user centered design that influence the insights and takeaways from our findings.

2.1 HPC Ecosystems

High Performance Computing (HPC) facilities are essential elements of data-intensive research. This type of facility is common in the funding landscape of the Department of Energy, as well as other agencies like the National Science Foundation. HPC facilities create and sustain a dynamic, diverse ecosystem where stakeholders from varied backgrounds, holding multifaceted roles, can accomplish work. These ecosystems provide cutting edge computing resources, historically oriented towards facilitating large batch processing jobs, as well as large amounts of data storage, and high speed networking. They also commonly pull in software resources from other realms and situate these products within a given ecosystem. The National

Energy Research Scientific Computing Center (NERSC), located at Berkeley Lab, is one HPC facility supporting more than 7,000 scientific users [9].

To contextualize this report's findings we first examine a few aspects of HPC ecosystems. We discuss some of the different roles stakeholders in these ecosystems frequently hold, the types of resources that make advanced scientific work possible, and characteristics inherent to working with HPC systems. We also provide a high-level overview of some of the resources the NERSC facility offers its users which emerge in our findings.

2.1.1 Roles of stakeholders in HPC ecosystems

HPC ecosystems require a large variety of stakeholders in different roles to enable and sustain scientific work. These individuals create and sustain a diverse human infrastructure [8], and over the course of our study a few broad roles emerge in our data, including: Domain Scientists, Computer Engineers, HPC Facility Staff, and Data Analysis & Visualization Tool (DAVTool) Developers. These roles capture different perspectives and challenges for individuals using HPC systems. This list is not exhaustive, but represents core constituencies whose impacts became visible in our study.

Domain Scientists. Domain scientists are researchers in different natural sciences, ranging from climate science to cosmology and material science. These individuals commonly accomplish their research goals as part of multiple scientific collaborations that are composed of complex web of coordinative entities [12]. This work involves varying arrangements of Principal Investigators (PIs), career scientists, postdoctoral researchers, and students. Domain scientists frequently produce novel data analysis software that relies upon packages produced within their collaboration and/or community, as well as the larger data science ecosystem. Depending on the volume, velocity, and variety of data being generated and used this software may be run on systems ranging from personal computers to clusters and HPC systems or the commercial cloud. In our study domain scientists are the primary users of HPC systems being investigated.

Computer Engineers. Computer engineers, in our study, are members of collaborations who ensure that scientific software runs on an HPC system. These individuals help turn the software domain scientists write into reliable and scalable products that can leverage the unique features of different HPC systems. Computer engineer's work ranges from modifying or adapting existing code to utilize features of an HPC system; identifying, installing, and maintaining software from a collaboration or community; or writing new code to adapt a collaboration's software stack to a particular HPC ecosystem.

HPC Facility Staff. HPC facility staff are comprised of individuals in various different roles surrounding the acquisition, operation, and support of HPC systems. A subset of the many varied roles at a facility can include: staff supporting the design and procurement of systems; liaisons between users (domain scientists) of a system and other facility staff to gather information about user needs; engineers who ensure the system is running on a day-to-day basis; and support staff who help end users troubleshoot their use of the system.

Data Analysis & Visulization Tool (DAVTool) Developers. An essential type of scientific software are general purpose tools for data analysis and visualization, such as VisIt [7] or ParaView [14]. In addition, there are endless domain-specific software packages and tools. Developers of general purpose DAV tools are faced with producing resources that work with a variety of data and scientific methods. In our study we see these individuals interact with other stakeholders in an HPC ecosystem, and beyond, in their efforts to support particular scientific collaborations or projects.

2.1.2 Technical Resources in HPC ecosystems

Facilities supporting HPC systems have to procure and support a variety of resources. Hardware resources include an HPC and its components, data storage systems, and advanced networking to connect these elements. Software resources range from particular operating systems to stacks of end user data processing and analysis tools. Here we briefly describe some of the key technical resources provided by HPC facilities that influences our study's findings.

HPC Systems. HPC systems are built out of a series of nodes connected to each other and data storage

systems via high speed network interfaces. Three common types are Login, Compute, and Data Transfer nodes. Nodes have different numbers of processing cores, quantities of memory, and operating system configurations depending on their purpose. Login nodes are where users interactively logon to develop and test code, use interactive computing tools like Jupyter Notebooks, and explore data residing in the system. Compute nodes are used for large batch processing jobs where intensive computations are done in parallel. Data transfer nodes are configured to support the movement of data in and out of the HPC facility, and its systems, from other facilities.

Data Storage Systems. Data storage systems commonly include forms of disk-based storage, increasingly flash memory (SSD) storage, and long-term tape archives for massive quantities of data. Nodes in an HPC typically do not have local disk storage. Users must rely upon the different shared filesystems that are mounted on each HPC node. Filesystems often have different policies and characteristics depending on whether they're meant for short-term storage during computing jobs, medium-term housing of working files, or long-term archive of large shared datasets. In some cases flash memory storage is attached to a subset of an HPC system's nodes to provide local high speed storage for use when executing large compute jobs. Tape storage systems provide long-term archiving of massive datasets that are used infrequently.

Cluster Management & Job Queues. HPC systems are complex and managing the demands of many competing users, and their computing needs, requires the use of software for cluster and resource management and scheduling of computing jobs. Jobs are an allocation of the system's resources (processing cores, memory) for a specified period of time. Management and scheduling software takes the job specifications that a user provides and determines when to execute the work on the system.

Core Software Environments and Tools. The core software environments on HPC systems span users home directories to stacks of software pre-configured for use by workflows and analysis tools, such as compilers for various languages. User environments provide the spaces for home directories and default shells that can be customized. Software stacks provided through modules enable users to access common types of software (e.g., versions of the Python language stack, parallel computing tools, compilers, etc.).

Data Analysis & Visualization Tools (DAVTools). Users frequently work directly with large datasets directly on HPC systems. Facilities support varied tools for doing this work, including general purpose DAVTools like Jupyter Notebooks, VisIt, or Mathematica, as well as domain specific packages.

2.1.3 Characteristics of working in an HPC ecosystem

Working with an HPC system is different from using a personal computer, private cluster, or the commercial cloud. Users must learn how to work with a few fundamental characteristics of these computing environments. Characteristics include how to program for HPC systems, obtaining and using allocations of computing time, and working with batch queuing systems for running jobs.

Programming for HPC. The nodes in HPC systems are composed with varying numbers of processors (or cores, depending on the chip design) and memory that is shared by the processors. Applications designed to execute tasks in serial process data in a specific sequence using a single processor or core. Leveraging the size and scale of an HPC requires crafting programs that work in parallel, splitting tasks up so that many can be completed simultaneously across processors and nodes. These parallelized applications can accomplish more computation in a given timeframe and users must design, and frequently optimize, their scientific software for this computational mode. In addition to parallelizing work, HPC systems mostly rely on shared data storage systems globally available across nodes. Some systems also include some form of local storage, recently flash memory, on a subset of nodes. User applications have to be constructed to manage moving data between systems before, during, and after a computing job to ensure that the work can be completed. Domain scientists often have to work with computer engineers and HPC facility staff to parallelize their software, and optimize data management. These tasks can require understanding the nuances of HPC systems overall as well as one particular machine.

Allocations. HPC facilities manage the shared use of these systems through allocations of computing time. Users apply for time on the machines and allocations are provided to projects/PIs in consultation with the

funding agency. Individuals and groups with time allocations then use this when submitting jobs to be run. Allocations are provided as compute hours and the facility determines an annual rate for jobs using different resources, whether that is the type of processor or priority queue. PIs and their colleagues have to determine the quantity of allocation to request based on their expected computing needs to address their planned science goals. The process of estimating can require science teams coordinate with computer engineers or facility staff to help develop a reasonable request.

Batch Queues. Computing allocations are used on HPC systems through batch queues. This type of software schedules and manages jobs by assigning computing tasks to different system resources. Jobs have a "footprint" based on the number of nodes used and the maximum amount of computing time requested (known as the walltime). Scientists submit a job to a particular system queue with a request for a particular number of nodes and walltime. Facilities frequently setup different queues that provide different quality of service (QOS) which constrains the amount of time that can be requested, number of nodes, priority for a given job to run, and the cost to a user's allocation. The queueing system's scheduler will take a user's choice of queue, number of requested nodes, and walltime to prioritize when, and on what part of the system, to run the computational tasks. These systems are designed so that the many nodes of an HPC are utilized as much as possible at all times. For users requesting more resources (e.g., more nodes and walltime) it is likely that there will be a longer wait time for their job to be run.

2.2 HPC Systems at NERSC

Seeing the variety of stakeholder roles, technical resources, and some characteristics for working in HPC ecosystems, we briefly discuss some of these elements at the NERSC facility. NERSC sustains one to two HPC systems in production simultaneously while also preparing for a next generation replacement of the oldest system. NERSC also runs other infrastructural computing systems including the Spin container-platform for hosting databases and web gateways. In addition, the facility supports different data storage systems and software environments for their users, including interactive computing through Jupyter [13].

At NERSC the HPC systems in production at any given time include nodes with varied purposes, from login nodes that support interactive access to data and batch processing nodes for large computing jobs to data transfer nodes for moving large quantities of data in and out of the facility. Our study spanned the period where the Cori system was provisioned and brought into production. NERSC's previous HPC Edison was in production and available to users as well and its shutdown was planned around the time we conducted our final interviews. Planning for the next-generation system, Perlmutter, also began towards the conclusion of our study. The facility hosted various shared filesystems that users access from different systems, along with some machine specific solutions, including a global disk storage solution and an HPSS tape-based archival system.

Cori was built with 2,388 Intel "Haswell" nodes and 9,688 Intel Xeon Phi "Knights Landing" (KNL) nodes [10] for computing jobs, as well as login nodes for direct user engagement with the system. Over time NERSC added large memory nodes to the system as well. All of the nodes provided access to the shared filesystems and a dedicated Cori scratch filesystem that offers higher performance. The compute nodes included access to the SSD-based Burst Buffer for high throughput storage during computing jobs. For users running compute jobs the nodes built with Intel Haswell processors are similar in design to previous generations of systems and required minimal effort to adapt software to run jobs. In contrast, nodes with KNL processors presented a new architectural design with more cores per processor but less memory shared among them contrasted to Haswell processors. This shift in design often necessitated changes to scientific software to take effectively use these resources.

2.3 Sociotechnical Perspectives on Collaboration and Design

Sociotechnical scholarship in Human Computer Interaction (HCI) and Computer Supported Cooperative Work (CSCW) informs our findings. A sociotechnical viewpoint recognizes and foregrounds that, when studying and designing technical systems or solutions, there are fundamentally many different social and political issues bound up with the technical decisions being made. CSCW work investigating the complex, social and technical dynamics to scientific work, computing, and data (see [5,12,18] for overviews) influences

how we grasp HPC ecosystems, and the ways stakeholders work within these environments. User centered design principles from HCI shape the takeaways we offer about designing usable data abstractions.

Collaboration is a key aspect to successfully accomplishing scientific work in all contexts, and particularly when working with resources at HPC facilities. CSCW research underscores the balancing acts and tensions inherent in multidisciplinary collaboration. A noteworthy element of such work are the inherent tensions between the research and development needs of domain scientists and computer scientists [6, 17]. Domain scientists increasingly have a need for usable software but may not have the expertise or resources to build products needed in their work, requiring collaboration with computer scientists or software engineers [19]. Computer scientists, in contrast, often need to focus on their own novel research rather than engineering stable tools. Tensions like these can end up shaping the experiences HPC users have engaging with DAVTool developers and facility staff.

Lee et al. [8] described the varied nature of the "human infrastructure" of large scientific projects. They emphasize how individuals may hold the multiple roles and frequently lack of clarity on their own membership among different teams. Stakeholders working in and using HPC facilities may encounter multiple diverse human infrastructures depending on the projects they are involved in. Steinhardt and Jackson [22] noted that when trying to cultivate vision in collective practices as part of large, distributed scientific projects stakeholders can end up undertaking "anticipation work." Anticipation work is the practices individuals and teams use to develop collective visions for the work to be done and how to shape and leverage changing technologies. With the longitudinal nature of HPC facilities, stakeholders engage in anticipation work as new systems are envisioned, procured, and provisioned for end users. We see parts of such work in our data with the advent of the Cori HPC at NERSC.

Understanding these issues from collaborative work, we also find our findings are informed by design concepts from HCI and user-centered design [11, 20, 21]. Usability and usefulness are key qualities of interfaces and design research has generated a variety of heuristics and principles to shape the development of systems. Shneiderman et al. [21] note that usability measures including *time to learn* (how long a user takes to understand how to accomplish a task), *rate of error by users* (how many and what types of errors do people make using a system), and *retention over time* (how well do users remember how to accomplish a task after time has passed). Sharp et al. [20] lay out a few common usability goals, including having systems that are effective to use (*effectiveness*), efficient to use (*efficiency*), safe to use (*safety*), easy to learn (*learnability*), and easy to remember how to use (*memorability*). Nielsen [11] specifies ten usability heuristics. He importantly underscores that usable systems make the status of the system visible to users (*visibility*), consistent and standard descriptions and actions across parts of the system (*consistency*), be flexible and efficient to use (*flexibility* and *efficiency*), and ensuring errors users encounter are able to be recognized, diagnosed, and recovered from (*recognizable and recoverable*). Overall, these foundational design principles for systems illustrates varied facets to working with computing systems. While they are most commonly applied to personal and mobile computing, as well as websites, these usability heuristics are important for ensuring HPC systems are usable, useful, and accessible to diverse scientific user bases. Our results illustrate various areas where current and past systems fall short in addressing these heuristics.

3 Research Methods

We conducted qualitative user research at Lawrence Berkeley National Laboratory to gather in depth insights about the ways users work with HPC systems. Qualitative user research approaches can incorporate a range of methods from ethnographic observations, semi-structured interviews, descriptive surveys, collections of artifacts from a field site, and usability evaluations. The UDA project conducted multiple rounds of semi-structured interviews, between 2014-2018, to learn about HPC ecosystems and develop insights that could shape a data abstraction tool [4].

3.1 Semi-Structured Interviews

Semi-structured interviews help researchers learn about the observations and practices of participants. This method helps us develop detailed descriptions of tools and work practices, hear varied perspectives on tasks and challenges, and grasp problems from an overarching perspective [23]. Designing semi-structured

interviews involved crafting a protocol with a set of open-ended questions to guide a conversation while leaving room for exploration of nascent topics. This enables the researcher to probe issues or topics more deeply, expanding beyond their preconceptualized notions.

We conducted 56 interviews between 2014-2018 that on average lasted one hour each. Initial interviews developed a starting sense of user's understanding of HPC systems and experiences working with different software tools and storage systems in their research. Later interviews explored storage system abstraction layers, data analysis and visualization tools, and the challenges with the Cori Burst Buffer in depth. Interviews were transcribed and discussed among the research team.

Initial interviews were designed to learn about a participants research work, use of HPC systems, and the data and software work this entailed. One round of subsequent interviews asked users to show us how they worked with different data analysis and visualization tools in HPC ecosystems. A different round of follow up interviews explored temporal issues in greater depth – asking about user's compute allocations from NERSC, how much time they spend on tasks when working with an HPC, and how they track the amount of time work took. These interviews raised challenges with data storage systems in varied HPC machines as well as the abstraction layers that condition user's interactions with the system. Finally, our last round of interviews asked participants whether they were using the Burst Buffer, what their experiences were with this feature, how their understanding of this storage component evolved over time, and how it compares with other HPC storage systems.

3.2 Data Analysis

Our interviews have been analysed using an iterative qualitative data analysis approach over multiple years. For this report the first author revisited this dataset and open coded each. Open coding [2] is a qualitative data analysis process where insights are systematically drawn out and compared across interviews to identify common themes. In our case, codes and themes emerged about users experiences and understandings of HPC systems, issues with abstraction layer tools, and experiences with the Burst Buffer. The first author also compared the points about temporal issues that re-emerged with findings published in [3] to validate and summarize the points in this report.

4 User Experiences Working in HPC Ecosystems

The findings from our project's interviews surface different aspects to how users understand HPC systems, the utility and challenges of abstraction layers in these ecosystems, and overarching issues regarding time and economics around computing allocations for projects with end users utilizing HPC systems. These insights capture multiple angles to the experiences stakeholders have working with different HPC machines.

4.1 Examining How Users Understand and Experience High Performance Computing

HPC ecosystems are more complex computing environments than everyday computing devices. A recurring theme in our data is the importance of recognizing how users perceive and understand an HPC system as well as the ways they and their collaborations do work. Over the course of our study we saw varying perspectives from users and developers about the perception of the resources available from HPC facilities. Stakeholders in HPC ecosystems bring various worldviews to their work with these systems, including the ways people conduct research and how data, software, and systems are constructed. Domain scientists and computer scientists have different ways of structuring their ideas and these variations end up being expressed through the designs of their software and data products. The social and technical facets to data, software, and systems require understanding and accounting for a variety of different points of view. Many potential mismatches and misalignments between a scientific user's use of an HPC and the modes of computation that the system and facility provide result from the diverse points of view and ways of working in HPC environments. DAVTool developers and facility staff noted that domain scientists often want to treat computing systems as black boxes in their work, and yet these users still must continually work to understand how an HPC system functions to accomplish their work. Learning about and understanding how to

effectively use HPC systems requires a significant amount of knowledge to be able to open up and effectively peer into these black boxes and the way data is stored and moved through them. Teams building DAVTools, furthermore, explained the necessity of fostering relationships between diverse stakeholders to align worldviews and build usable systems.

4.1.1 Commonly black boxed views of computing systems

Computing systems are commonly opaque, filled with unexamined internal details – a black box to users. Some DAVTool developers and HPC facility staff explained that in their interactions with science users there is often a desire to be able to treat any particular computing system as a black box. So long as a given input produces an expected output, domain scientists often do not express significant concern about the inner workings of a computational system. For these scientists, so long as their software functions reasonably well, the details of its internal construction, or the HPC system used to run it, are not at the forefront.

One leader of a bioinformatics DAVTool team noted that the scientists they support have historically been likely to treat their software and computing systems as black boxes. If a result took a day or even a week that was fine so long as running the tool was simple. This individual explained that in recent years they had started emphasizing a need to re-evaluate older tools. This is necessary since the software's performance was not great, and with newer computational architectures the tool could likely be updated and made more efficient. Scientific end users often don't have the time to be able to learn about every detail of a system's construction, such as the number of processor cores and their shared memory, and how their software could perform better by leveraging a new system feature. They want to focus on their research and usually become concerned with computational efficiency when this work is significantly hindered. Scientists in such cases must work to better understand how HPC systems function in different ways.

4.1.2 Perpetual work to understand how HPC systems function

Learning and understanding how HPC systems work is an ongoing process. Usability characteristics such as time to learn, retention of knowledge, and overall learnability of HPC systems is variable and shifts over time. Participants in our study ranged from new to experienced HPC users. Multiple interviewees commented on the need to continuously learn how an HPC system functions as they do their work. This is necessary because alignments between hardware, software, and policy are always evolving, at different rates, in an HPC facility. One interviewee summed up this need, expressing that after more than a year of using one HPC system they felt that they're *"still learning I guess"*. At the same time, users with over 20 years of HPC experience still face difficulties understanding these systems as new designs emerge incorporating new types of hardware into the computing fabric. Interviewees noted these challenges with hardware elements of a given HPC system such as I/O performance of data storage systems, as well as the varying software configurations provided. Domain scientists also face challenges determining how to balance the amount of human time to expend to optimize software for a machine versus the performance benefits obtained.

Uncertainty about I/O performance among data storage systems. Science users noted they often have some amount of uncertainty about the operation of data storage systems in an HPC system. They grasped the overall systems offered for different machines at a facility (e.g., flash memory, disk storage, tape archives), but did not necessarily spend the time to try and learn which data storage system would be best for their current work. Instead they would try different solutions to figure out what worked best for the problem at hand.

One experienced science user described a situation where their Python code unexpectedly began running slowly. His expectation was that a parallel implementation would provide performance gains, yet it was running exceptionally slowly and erratically on the same dataset. Experimenting with the facility's different data storage systems at the time, he found that running the code off a different disk setup lead to significant performance improvements. Reflecting on this experience he commented *"we never did figure out why that was, but when I copied the file over to scratch, all of a sudden it is an order of magnitude faster."* This scientist ended up having to expand their knowledge about the different available disk systems at the HPC facility and the performance implications of each. They assumed that using this more optimized Python implementation would provide benefits no matter what, but it turned out more work was required to understand a bit about the different data storage system's and their performance characteristics.

Another example comes from a scientist who was building tools for their group. Through their software development work they implicitly designed the I/O patterns of their tool with the assumption of smooth, constant performance like their personal computer exhibited. In this case the scientist was incorrectly expecting consistency between the behavior of an HPC system and a laptop or desktop computer. This implicit assumption came into tension with the way HPC systems perform once this scientist pivoted to testing the software in a production context. This individual found that his software's ability to even read files would not operate smoothly since the load on the HPC could be variable depending on how many and what type of jobs were running simultaneously across nodes. Their assumptions were rooted in experiences with personal computers and broke down in a multi-user environment, leading them to determine that they should keep in mind the variability in I/O performance from storage systems when moving between types of computing environments.

Keeping up with software configurations. The software ecosystems configured on HPC systems will vary from one to another, even within one facility, due to differences in the hardware making up each machine. These changing configurations emerged as another challenge impacting learnability, memorability, and consistency for users who have different depths of understanding of a given HPC system.

One example of this emerges with the way software modules are installed and made available to all HPC users at NERSC. The facility has a few options depending on how widely they anticipate the software will be used, as one staff member explained. During our study, NERSC had two HPC systems running, each configured with access to a common data storage system that supported access to home environments and file repositories from either machine. With this configuration users are able to create a global software module that can be employed on either system. The facility itself however creates separate modules, even for the same software, optimized to each HPC and its overall configuration. Configuring the software environments with this delineation makes it clearer for end users to be able to easily recall which computing context they are working in should they get lost in details of their work. A staff member asserted that end users building their own module configurations would benefit from this approach, since doing so would ensure the scientist's workflows run consistently while allowing the software to be tuned to each HPC system's features. Mirroring this implementation approach would support memorability and visibility for scientists when they need to quickly ascertain which environment their software is executing in. The facility staff member noted that their web-based documentation describes this design pattern for configuring software modules as a recommendation to their user base.

The various modules produced by either the facility or scientists encapsulate a particular version of the software. This facility staff member also explained that when the facility updates versions of existing modules the new ones are installed as a separate instantiation while leaving the old default alone and present. Users are then informed via regular updates, whether via email or in announcements on the website, that there is a newer version of the software in the revised module. End users are also provided with a date to expect the now older version to be deprecated and removed. This practice put in place by NERSC ensures that scientist's workflows can remain stable in the short-term, providing consistency from a usability perspective, while their users and maintainers have an opportunity to revise which module they rely upon and test the effects of the change. In contrast, when a new HPC is brought online its default will be set to whatever the latest stable version of a piece of software is. This version may be out of alignment with what is installed in the old system's environment. In these cases, it is expected that users must adjust and realign their workflows and practices to the new HPC and its software configurations. This is an opportunity for scientists and their collaborations to systematically optimize and/or modernize their software stack.

Varying views on effort spent optimizing human versus machine time. Completing scientific work is always a process of optimization and making tradeoffs. Scientists and computer engineers face different optimization choices when building and maintaining scientific software, from machine learning and parallel programming techniques to collaborating with HPC facility staff to ensure that their essential software will work in an efficient manner. One facet we saw above is understanding and managing time when working with data. Determining which file systems in an HPC to store data on based on unclear performance characteristics, along with questions of when and how to retrieve data during development and testing or production runs of jobs, can be difficult to sort out when trying to optimize a workflow. Associated with this issue, as we examine more in a later section, abstraction layers can make it challenging for end users to understand how long it will take for data to be loaded for jobs.

In a different vein, as we noted above, HPC ecosystems periodically shift when facilities update hardware or software that a user's work depends on. This introduces additional work to adapt to any changes in the configuration. Facility staff expend time before deploying upgrades to test and try to identify any potential issues, while recognizing that effort will be required after deployment fixing bugs that surface. Scientific users may need to adapt their software in light of changes, adapting and optimizing their setup to the changed system configuration. Staff at HPC facilities, such as NERSC, offer the expertise to help scientists learn about and better understand how their software performs on a given system in these situations. Collaborations between staff and scientists to optimize software can save computing time and ensure a more reliable product. A concern among some domain scientists in our study was that undertaking this type of collaboration may be labor intensive if they need to teach facility staff or computer engineers about their domain itself. Scientists may not grasp that putting effort into a collaboration up front can pay off in the long run with more stable and better performing tools. We see in particular that DAVTool developers find fostering collaborations and learning the domain science language to be beneficial to getting work done.

4.1.3 Fostering relationships to align scientific and computing worldviews

Domain scientists often treat HPC systems as something of a black box and have to undertake significant work to understand and learn about their changing operation over time. At the same time, a key part of the experience of working in HPC ecosystems is collaborative work. This may be as part of a particular scientific team tackling a shared research problem, or in conjunction with computer scientists building software resources (such as DAVTools) necessary to make science possible. Importantly, the stakeholders in multidisciplinary collaborations all have to work to understand the language and cultures of their colleagues. Research has noted that part of this effort is determining the right amount of, as well as who should be fostering, communication [6].

Participants in our study building DAVTools, in particular, raised the challenge of effectively communicating across disciplinary and technical boundaries. These individuals expect their work is more likely to be successful when they intentionally form relationships and develop shared perceptions of a design space (whether that's the scientific research approach, the software design approach, or the HPC environment, etc.). Computer scientists and domain scientists have different disciplinary languages and points of reference. DAVTool developers interviewed explained how they find that building a relationship between individuals across teams is a fruitful mechanism for knowledge to be exchanged and longer-term efforts sustained. As part of building relationships, a way to help make generic tools (e.g. a generalized data analysis and visualization tool) more usable and friendly to a particular scientific community is by crafting interfaces and abstractions. Crafting interfaces and abstractions, whether graphical or the terminology in APIs, that are aligned with a domain specific language externally and mapped to a tool's internal structures enables domain stakeholders to engage with a system without requiring deep customization.

One visualization tool developer explained the utility of this approach when working with climate scientists. They crafted a graphical workflow that portrayed data by default in geographical terms as a translation on top of the tool's internal representation and computation of the data in a more generic form. Complex aspects of the tool were abstracted away, available for the interested user who digs in, while helping the scientists get their immediate work done. In this case climate scientists were more easily able to engage with the software, helping grow the tool's user base. This developer was able to facilitate this growth in part by building a relationship with the scientists in particular projects, and building an understandable representation and mode of interaction.

4.1.4 Takeaways

- *HPC users may want to treat the system as a black box where they do not have to spend time learning about its internal operations. Collaborations between domain scientists and developers or staff can encourage users to learn more about necessary aspects of a system to ensure their software workflows are optimized and taking advantage of all features.*
- *Users may encounter uncertainty with the I/O performance of different data storage systems if they view these elements as the same as what their personal computer provides. The varied performance of different data storage*

systems furthermore can lead to confusion for users working to implement and optimize their software. This may result in a need for facility staff to support scientists encountering I/O problems.

- *The commonly changing software configurations of HPC systems can be challenging for users to keep track of over time, inhibiting the system's memorability and learnability. Configuring software modules optimized for each HPC system is one way that NERSC already works to make differences and changes visible over time. Teaching users this pattern to constructing elements of their work could increase their ability to track changes to systems.*
- *Scientists should engage with HPC facility staff whose job is to collaborate and help optimize scientific software tools. Collaborating will benefit scientists through more reliable, optimized tools while facility staff learn more about their domain which supports future engagements.*
- *DAVTool developers who build relationships with their user community learn about their worldviews and research practices. This can inform the design of simple abstractions to make a general purpose tool more accessible to a specific community of domain users, so long as these users can then explore underneath the abstraction as desired.*

4.2 How Users Comprehend HPC Systems and Abstraction Layers

Abstraction layers are a key way to manage the complexity of large-scale systems and potentially improve usability for different types of stakeholders. Middleware tools are abstraction layers that can make it easier for scientists to work with an HPC, or other, system as long as they can fairly easily learn and grasp what the abstraction layer is accomplishing. At the same time many of our study's participants conveyed issues that can make them or their colleagues wary of abstractions. This emerges from a tension abstraction layers must address, that between providing simplicity and visibility into system details.

Abstraction layers are fundamental concepts in software development to mask complex system features to provide a simpler user experience through the software design principle of information hiding [15]. Engaging with different types of computing systems may foreground different types of abstractions. Using a personal computer requires minimal examination of the abstractions around storage systems where as the experience with an HPC system and its varied data storage systems introduces a different experience for users. As part of the perpetual work to understand how HPC systems function we see that the memorability and learnability of different abstraction layers utilized in an HPC ecosystem is key to the user experience.

Participants in our study encounter many abstraction layers when building different data systems for scientific work and these tools positively and negatively impact user experiences. Many expressed frustration with the design and functionality of abstraction layers wrapping different HPC components, from data storage systems to parallelization tools for running jobs across many nodes. Often these abstractions are black boxes to their users, sometimes by the user's choice and at others due to the tool's design. The ability of users to open up these black boxes and clearly identify which systems their data is flowing through was too often minimal, making them wary of new abstractions in current HPC systems since breakdowns may be hard to interpret and inspect. This ability or inability to comprehend and account for the fabric of different HPC systems in the software remains a challenge for scientists and developers. At the same time, multiple interviewees noted that abstraction layers can simplify the software they have to build for different projects. At the same time the information hiding principle underlying abstractions can generate challenges when breakdowns take place. Abstractions that do not provide sufficient means for users to peer inside and investigate breakdowns result in a diminished user experience. We see examples of this from the experiences of multiple different scientists.

4.2.1 Abstractions breaking down with architecture shifts

HPC system design is continually evolving as new hardware and software resources are adopted. The evolution of these systems ends up impacting and changing various aspects of usability for users, including consistency as features and behaviors change as well as learnability and the ability for users to identify and recover from errors encountered in their work. Abstraction layers in software tools are one way to smooth over some of these issues for users, but they raise their own concerns in high performance environments.

When HPC systems change and an abstraction layer breaks down due to an architectural shift, the engineers and users working with scientific software encounter new errors and performance slow downs that may require more work to resolve.

One staff member explained this challenge by noting that two decades ago, when the cost of using an HPC system was higher, developing applications for an HPC system required understanding what resources were available and how they would impact the performance of particular computing jobs. Today, it is more common for scientists, and their collaborators, to write software then test to see what performance is like without explicitly factoring in the resources, and their nuances, of a particular system. One computer engineer noted that abstractions are "inviting and helpful for thinking about things" but often end up deterring high performance over time as system's change. This becomes a problem and breaks down when an underlying architecture change disrupts the abstraction and causes poor performance, unless someone takes the time to revise the abstraction layer itself. To this computer engineer the solution is to avoid "fancy abstraction stuff" and to optimize the scientific software they are supporting, but this comes as a trade off since "usability is abstraction" to some extent.

4.2.2 Necessity for user's to have visibility into I/O middleware operations and clear error messages

A key element to usable abstraction layers and systems is the ability of their users to have visibility into the black boxes these components create. Work in HPC ecosystems is particularly impacted by data storage systems and their I/O performance, as we saw above. To make systems usable and efficient for users, visibility into abstractions wrapping data storage systems is important since the underlying devices often have widely varying performance characteristics and this impacts how users perceive the work at hand. Each data storage system may be designed and optimized with different purposes, some short-term and high-throughput for use when running jobs and others long-term and low-throughput when archiving data for sporadic access. Executing a large analysis job may result in a disruption or breakdown if the location of data is hidden behind an abstraction layer that does not provide users with a sense of relative time to access or load particular subsets. Additionally, usability issues can also emerge with the error messages generated by a particular abstraction layer, and how they can be understood by users.

One domain scientist explained such a situation when they were working with a particular HPC system at NERSC. This astronomer was developing a data analysis pipeline and using a middleware tool to handle I/O operations for their data. This middleware tool seamlessly moved files between the system disk and a tape archive. These were data storage systems with very different performance implications, and the middleware was designed so that users didn't need to know where a file was stored when accessing it. When this scientist began running their performance intensive workflow the software ending up breaking down fairly easily. Over time through troubleshooting this scientist realized the breakdown occurred when some of the data was on disks while other pieces were in the tape archive. The middleware tool had completely abstracted away any visibility of the location of an individual piece of data. The middleware also made it difficult to understand the inner workings of this software to isolate the issue since it didn't offer "enough power hooks" where the user could query to filter and show only readily available data products for example. Consequently the variability in access times for various pieces of data would hold up the analysis software, to the point that the scientist's software broke down and caused this individual significant frustration.

A similar case emerged for a domain scientist trying to manage the execution of their computational models at scale on an HPC system. This user decided to turn to a NERSC task management tool. This tool was designed to help users make their computational model execute in parallel across many nodes of a NERSC machine. This scientist expected this tool to handle configuration issues, but they rapidly encountered errors that were "cryptic" enough that this user and their colleagues couldn't figure out why their tasks were failing. Over time through testing they came to the conclusion that the abstraction layer would start off running tasks in parallel before unexpectedly breaking down when trying to read in the various files. For this scientist, their best conclusion was that file I/O was happening in serial and hampering the parallel execution of tasks that the tool was supposed to enable. This abstraction's breakdown was not expected by the scientist, and produced error messages that offered little understandable feedback. Consequently, this user wasn't provided with sufficient insight about the issue emerging and how to fix it. This user needed

the ability to peer into the abstraction's operations but this was not possible.

4.2.3 Takeaways

- *Abstractions can make complex systems more usable for end users but can easily break down as computing architectures shift and must be maintained to remain useful and usable.*
- *Users concerned with the performance of their software should re-evaluate and revise the abstractions they're designing and using when working with new computing architectures to obtain the best performance.*
- *Providing users with the ability to understand how a middleware tool operates internally, and perhaps to specify complex usage scenarios, will enable a better user experience.*
- *User experiences can be simple when relying heavily on default abstractions so long as additional options are provided for more advanced users who need to handle more complex analysis configurations in different HPC environments.*
- *Error messages must be designed to provide clear, concise descriptions of the problem and ideally a pointer to more information. These messages can be essential to helping users open the black box of an abstraction layer.*

4.3 Challenges Scientific Collaborations Face With HPC Allocations & Queues

A fundamental element of HPC ecosystems that domain scientists work with are allocations of computing time. As we described earlier, scientific projects receive allocations that they can spend on running computational jobs through different batch queues. HPC facilities configure batch queues on their systems with different performance characteristics, and consequently costs in terms of allocation usage. Among our study participants we found that projects faced challenges and misalignments around computing allocations as well as with the design and use of queuing systems.

4.3.1 Challenges and misalignments with computing time allocations and policies

Receiving computing allocations is a key aspect to the funding of many large scientific project since the processing and/or analysis of vast quantities of data would not be feasible without an HPC system. HPC facilities end up balancing varied social, political, economic, and technical concerns with their policies for allocating and enabling the use of computing time over the course of a given year. We see two primary challenges around computing allocations and policies for domain scientists. First, that projects and their members require visibility into the quantity of computing allocation they have over the course of a year. Second, the rhythm of collaborative work in a project may be out of alignment with the need to steadily expend allocation.

Projects and their members need to have visibility into the quantity of computing allocation available over the course of the year, from the time a request is made through its use as work is accomplished. Science users face challenges when determining how much of this "virtual currency" to request, and how to manage its use over the course of a year. Projects may end up receiving a smaller allocation of computing time than they perceive is necessary to get their research done if they have a difficult time determining how much is reasonable to request. Shortfalls can emerge sometimes if a request is made based on past experiences running software in HPC environments but these don't align with the current work. This can be a challenging balancing act when large collaborations have software in various states of functionality or varied data collection timelines making the expenditure of computing time uneven over the course of a year. At the same time, the finite amount of computing time available on any given HPC system results in a facility expecting that allocations given to projects, and users, need to be spent steadily throughout the year. NERSC, for example, has a policy emphasizing that allocations will be reduced at set times over an operational year if sufficient quantities of computing time are not being spent by a project. This reduction returns the time to a pool that facility staff can reallocate to other work. This is a mechanism to ensure that the limited time resource is not allowed to go to waste. This policy is outlined on NERSC's website, albeit directed at principal investigators and account managers, breaking down how much will be removed in particular circumstances. Consequently project members may not realize if an allocation they're expected

to use has decreased, or even increased, over the course of a year which can lead to unexpected disruptions to running their software.

We also see that NERSC's policy for using computing allocations can be out of alignment with the rhythm of work in particular teams. Multiple scientists that we interviewed were not always aware, or were unclear on the specifics, that they would lose allocated time if they were not steadily using some amount over the course of a year. These scientists must balance the expected amount of computing time that may be needed to achieve their scientific goals with the effort to get their software to work (an optimization issue). The rhythms of work within a given project, and when large amounts of HPC time is needed, will not necessarily align with the availability of a particular HPC system, or could place a large burden on this shared resource, limiting how many other users can do their work. Teams can spend many months using small amounts of their allocation as they work to get their software running and optimized. Once the software is optimized and functioning as desired the researchers will then be able to go through thousands or millions of hours running jobs in a short amount of human time.

For example, one scientist described their project's allocation of around 40 million hours of computing time before explaining that at that point the project had only used maybe 15 million of those hours. This researcher expected to use maybe ten million hours for an upcoming run of the team's code, but they were still in the process of testing their software before launching such a large job. When we asked whether the remainder of their allocation would be enough they were not sure and expected that they might have to request more time from the facility. In the past their colleagues have run over their allocation and just automatically had their available amount extended without a clear explanation. At the same time this researcher's colleagues also faced situations where they unexpectedly had allocation time taken away because they were not using it rapidly enough. In this case, this scientist is the project PI and therefore responsible for tracking and managing the project's usage. Even he was not entirely clear on the actual ways the policies underlying removal or addition of time unfold in practice. In this brief example this scientist was connected to three different project allocations. Two of these allocations were underutilized, yet only one received a reduction in available time in line with the facility's policy. This left this group of users unsure about the implementation of the policy in practice, leaving a gulf between the described approach and the actual. The team did not want to waste allocated computing time on something that was incorrect which is why they were focused on testing and optimizing their code, resulting in low utilization of their allocation up to this point in time. This scientist noted that he knew of no formal way to tell the facility about their plans to forestall any reduction in their allocation per the time policy. Within the facility ecosystem there may be an implicit approach to achieving this goal, but the pathway to doing so was not clear to this principal investigator.

4.3.2 Challenges with queuing systems

Domain scientists and their collaborations also face challenges understanding and determining which queues to use when running jobs on HPC systems. HPC facilities manage the use of a system's limited resources through processing or job queues where users submit the jobs they want to be run. Queuing systems function as an abstraction and management tool for ensuring the finite compute resources of a given HPC are utilized as efficiently as possible. Queues are designed with different shapes or forms that convey varying amounts of resources available. Designing queues to be efficient and effective requires balancing different concerns — the form of computation needed (e.g. how much process power, memory, time, etc.), the priority of access and other costs to the ecosystem and its users, and so on. Scientists in turn encounter uncertainty when trying to understand how much of an HPC a particular analysis will need, and how long they will have to wait for it to run when using different queues in the HPC ecosystem. We see that the design and structure of a facility or system's queues need to be made understandable to the end users if they are to be helpful and useful. In previous work we noted annoyance "at having to wait in the queue for a quick test during code development or testing" [3] since scientists desire the ability to evaluate a bug fix then move on to other work. We find that the learnability and consistency of different queues impacts how HPC users experience these ecosystems. Facilities in turn recognize such challenges and face their own difficulties balancing how to organize various queues to maximize utilization.

User confusion about queue setups and job execution times. Interviewees noted that queuing systems as

an abstraction can often make it difficult to understand how long they may have to wait until their analysis job is actually run on an HPC system, and then how long it will take to actually complete. Users indicated that they didn't necessarily have a grasp on how a system's job scheduler is balancing different factors when determining which jobs to run in a queue. They also found it difficult to pick a queue when there were many on offer when using a particular HPC system.

For example, one scientist explained that the ability to estimate start times for small jobs helps him decide when to kick off a job then pivot to other work or to wait and see what the results end up being. Sometimes this individual expects a short minute or two wait time and will *"probably type qstat five or six times"* while waiting. In contrast, if the showstart command indicated the job won't start for at least twenty minutes then they will move on to other work (or just go home). In one case twenty minutes was indicated but the job actually took more than an hour to be queued up and run by the system. For this user the commands available to probe a job's status in a queue do give him an idea of whether something will take a long time to actually be run. His experience has given him a sense that if the system reports a few minute wait time then this will be fairly accurate but as the estimated time increases the accuracy drops and jobs may not run for a lot longer than stated. Here we see that the consistency of the user's experience is variable and does not improve their understanding and ability to learn about the organization and operation of a queue.

In a different vein, another scientist faced challenges learning how to set up the archiving of their data and model setups using a queue dedicated to such a task. This scientist explained how there was a *"xfer queue specifically designed for backing up your files to HPSS"*, the archival data storage system. Information about this queue's configuration were opaque (black boxed) and only surfaced in use, rather than being more clearly designed in a user friendly, learnable way. This individual came to understand that seemingly the xfer queue was setup to only allow runs for twelve hours whereas the scientist's archives required more time. The scientist also wasn't sure how many processors this queue let tasks access and how this might be impacting the performance of their work. Their software was breaking down, and not failing gracefully, so this scientist had to quit using the queue and shift to using one of the HPC system's login nodes. This is a use case that the login nodes are not meant to be used for, but this user manages to do so to accomplish their goals.

Facility challenges designing consistent, transparent queues. Constructing queues for running jobs is a challenging process. Facilities have to balance overall usability and utility of their different HPC systems with flexibility to handle the different types of work scientists have, as well as different project's economic considerations around how to expend their allocations of computing time. Conveying the details of these decisions in an informative way without obscuring the functionality of the system is also difficult and make producing systems users can learn easily a challenge.

During our study we learned from NERSC staff how the facility had experimented with different quantities and types of queues on HPC systems over time to try and provide a more usable user experience. One facility staff member explained that at one point there were seven or so queues that provided users access to machine resources with different priorities for processing and amounts of memory and CPU resources available. This allowed NERSC to charge compute allocations at varying rates depending on the size and priority of work being executed. Finding that users had a hard time determining which queue to utilize, when the facility brought a new machine online they reduced the number to three — a debug queue for rapid testing of changes before later running a large job, then regular and premium queues for standard jobs with different priorities. This new design was partially possible because the software underlying the queues could now support users providing more information about their requests (e.g., more details about the amount of CPU and memory needed), and the system could then automatically adjust usage more efficiently. This would overall improve access to the machine and was intended to improve the usability for domain scientists. The challenge with adopting this queuing setup is that users visibility into this software's decisions and operations were obscured. If users end up without the ability to see why their work is or is not being run at a given time, thanks to the queue software's decision making, then they may end up frustrated with the HPC system. A key part of the design challenge here is coherently illustrating to end users how these automated queues operate and why the policies for them are designed in a given manner so that these scientists have visibility into the decisions, and can expect consistent behavior that enables them to learn how their work is likely to be run on a machine.

4.3.3 Takeaways

- *HPC facilities could clarify their mechanisms for warning projects of impending time allocation reductions. Augmenting automated reductions with clearer or more visible messages to the associated PI, allocation managers, and/or entire team with the action and reason for it would help clarify what did or did not take place.*
- *HPC facilities should more clearly define a process for teams to engage in discussions about their use of computing allocations so that timelines can be discussed and support provided where needed.*
- *Projects and teams using HPC resources should try to collectively track their use of time allocations and as much as possible forecast whether they will be in a situation where large amounts might expire. If they find themselves in such a situation they could start a conversation with the facility to try and come to an agreement that meets their timelines.*
- *Commands should present reasonable estimates for jobs and enough information to help a user understand the factors influencing the value.*
- *Configure queues with clear parameters so that users can develop reasonable expectations of how long their jobs will take to be run in everyday circumstances.*
- *Minimizing the number of queues users chose from while requiring more parameters about their jobs simplifies their engagement with the system but may make it difficult to grasp when their job will be executed.*

4.4 Challenges Deploying & Adopting Unique System Features: Experiences with the Cori Burst Buffer

Users face challenges with I/O that change over time as technological solutions are developed and deployed while working with continuously increasing volumes of data. During the initial phase of our study, NERSC was preparing to bring the Cori HPC online. Distinct from prior generations of machines the Cori system's computing fabric included elements that would upend the conventions of practice users had established, including manycore Knights Landing (KNL) processors and the Burst Buffer on-node flash storage. The Burst Buffer emerged as a point of anticipation in our early interviews since it was specifically designed to address I/O issues by incorporating high speed flash memory in the storage hierarchy. We subsequently interviewed users about their thoughts and experiences with the Burst Buffer after the Cori HPC had been in production for a few years.

Overall the Burst Buffer (BB) was introduced to provide higher performance I/O operations on Cori while balancing the cost of flash memory technologies. Scientists could employ the Burst Buffer during computing jobs by taking advantage of this higher speed solid state memory hardware that was included on a subset of Cori's nodes. Incorporating a Burst Buffer was a way to include this technology while making the overall Cori HPC economically feasible for NERSC. Our interviews identified varied visions and anticipations of how this feature would work and later reflections on the experience and usefulness of this element of Cori.

4.4.1 Anticipations and Visions

Facilities try to incorporate new technologies in HPC systems to understand how they can be used by scientists while generating insights for future systems. This practice and vision resulted in the Burst Buffer being woven into the computing fabric of Cori — as a resource to address current demands and as a way to look towards the future. Stakeholders we engaged with underscored varying aspects to this vision, both from the facility's view and that of the user community. Stakeholders from the facility as well as the user community undertake anticipation work [22] to develop a collective vision for how a new feature like the Burst Buffer can work and be useful. These visions and anticipations are shaped by past, and other, computing experiences. The usability, or lack of, systems in these experiences influences users expectations as well. We see that the initial descriptions of this feature by the facility, then explorations by users shape an initial collective vision of the Burst Buffer and its utility. Subsequent efforts trying out the delivered feature reshape and refine collective visions as projects and collaborations attempt to leverage the feature in their everyday work.

Facility staff visions and anticipations. The Burst Buffer was communicated to the NERSC user community as a way to ameliorate existing I/O struggles when running large computing tasks. Interviews with facility staff highlighted a vision that the Burst Buffer was an innovation that could improve user's compute jobs, include newer technology which would enable research on changing technological resources, and influence the design of future machines. A paper published by NERSC after the initial deployment and testing of the Burst Buffer described this feature as *"one path forward"* by serving as a *"fast storage layer, close to the compute"* to address the growing I/O challenge in HPC realms [1].

Stakeholders from the facility noted there was excitement about the Burst Buffer, but at the same time they wanted to manage and temper the expectations of user communities for the built solution. One subject explained that the Burst Buffer was expected to have variable success improving scientist's work depending on the structure of their software analyses. The anticipation was that the Burst Buffer would probably overall be faster for most users than established parallel file systems at the facility, but as one staff member described the feature *"it's not a silver bullet"* that will readily solve all issues. Realistically as a new technology the Burst Buffer is *"not gonna make all of the people happy all the time,"* it may help some users jobs run 10, 20, or 30 percent faster while others may see no or perhaps even negative improvements.

Recognizing this expected variable utility of the Burst Buffer, we learned that the system procurement process included a research and development contract specifically to develop usable software abstractions for the feature. These abstractions were deemed necessary so that science users could have a more seamless and hopefully transparent experience with the Burst Buffer. Part of the hope was that users would have minimal learning to do to be able to use this feature. One of the facility staff explained that this element of the contract was motivated by experiences observed at other HPC facilities incorporating cutting edge technology that required significant steps by users to actually use it in their computing work. Over time we learned that experiences with the resulting software stack ended up varying for users, with there being a lack of clarity about the scope of features that were even actually delivered years after the Cori HPC was in production.

Science stakeholder anticipations. We spoke with a variety of science stakeholders who use NERSC resources about their initial understanding of the Burst Buffer, and anticipations for its application to their computing work. These conversations included scientists doing particular research along with computer engineers who develop systems supporting large projects. Across all of these conversations the earliest notions of the Burst Buffer underscored uncertainty and confusion about this new resource. Interviewees professed that they didn't really know what the Burst Buffer was going to be exactly, and offered varying conjectures based on their experience with HPC systems that existed at the time as well as their personal computers. The Burst Buffer, like any change in an HPC ecosystems, raises the issues identified above with having to continually learn how HPC systems function and the need for visibility into I/O systems and their middleware.

HPC users with years of experience became accustomed to file systems globally accessible across an HPC. Consequently, in conversations about the Burst Buffer one domain scientist expected that this storage system would be available globally across all of Cori's nodes. They had spent years working with this model in other HPC systems and didn't anticipate the fundamental conceptualization of the HPC would be changing all that much. We explained the anticipated design to this individual, including that the Burst Buffer would be local to specific nodes. This prompted this scientist to speculate that the feature might be able to help their work by solving data caching issues that were occurring when running simulations. This user speculated that they could employ the Burst Buffer to save intermediate data products before migrating these outputs to NERSC's stable disk storage systems. Doing so would let their models treat this flash memory as extra disk storage during jobs. This type of use case was expressed by at least one other scientist as well.

In contrast, one scientist working with large observation datasets faces different I/O challenges. Rather than simulation workloads that need high bandwidth, their data processing requires many *"little metadata-style operations"* that stress I/O systems that are not optimized for low latency. This individual's expectation was that the burst buffer's attempt to provide storage faster than normal disks, but still not as fast as memory, would probably not really help their research work. Ideally from this user's perspective the Burst Buffer would feel like working on their laptop with a solid state disk where it functions mostly invisibly. Whether that would be the case remained to be seen at the time. In their idealized vision the Burst Buffer would

behave and perform similar to the commercial Dropbox tool at petabyte scale. Data would always feel like they are locally accessible, regardless of where they are actually housed. The system's software abstractions would function seamlessly enough that they do not have to spend time thinking about where there data is or how it works. This desire is similar to our earlier point that not having to think about data's location in a storage hierarchy is nice, but perhaps not realistic in practice. The feasibility of this vision with high volumes of data is probably low, but the influence of this user's everyday computing experiences juxtaposed with the potential for changes to the HPC's design results in a lofty vision, even tempered with this individual's expectations that this is unlikely in practice.

4.4.2 Early experiences evaluating the test bed system.

NERSC deployed a test bed system to provide access to an early version of the Burst Buffer. With the test bed scientists could explore how they potentially could make changes to their software to adopt this I/O feature in their work. Deploying a test bed is a mechanism for helping users to learn about the shift to the computing fabric and realign their ways of doing work, encouraging them to open the black box of the HPC and think about the way it functions when executing their software. Test beds however come with their own challenges and tensions emerged in practice with the early versions of the Burst Buffer that made users express hesitation about relying upon it in their work.

One scientist explained that their goal with using the test bed was to compare to standard Lustre disk storage available on pre-existing NERSC machines with the Burst Buffer. This individual wanted to determine whether reading files would indeed be faster since this new type of hardware should result in performance gains. They explained that during their tests they did not actually see such gains. He admitted that his small experiment's results couldn't necessarily be trusted since he wasn't sure he had actually devised and executed an appropriate test. This tension between having a test bed available and having the clarity to know whether an evaluation is appropriate highlights a challenge for facilities introducing new features and being flexible about their use. As a user this scientist was not sure how this impending change should necessarily be conceptualized and considered when building software analyses. With further inspection, we discovered that the test bed was not optimized for parallel read operations while the existing Lustre disk system was. As this scientist suspected, they were indeed comparing the performance of two different configurations that did not entirely align. This scientific user's inability to determine how the test bed was configured, and the relationship of this configuration to the existing systems they were accustomed to, hindered their ability to make truly meaningful use of the system. The abstractions they used when testing this new way of working didn't provide enough visibility into either data storage system to help them determine what was being evaluated.

4.4.3 Experiences and reflections on the built system

We investigated user's experiences with the Burst Buffer once Cori was in production and available for everyday use. We found that many interviewees and the projects they work on conducted some exploratory tests of the Burst Buffer, but the majority of the subjects we interacted with claimed they did not end up using this feature all that much. Interestingly, stakeholders from NERSC noted that the feature is in fact heavily utilized, based on usage logs and system activity, indicating there is a user base but they were not clearly visible in our study. Future studies could rely upon system logs to identify active users to gather their perspective.

Regardless, even with logs indicating the feature is used, it is not necessarily clear that the Burst Buffer is being utilized as it was envisioned. Users may be leveraging the feature without getting the full range of benefits possible due to issues we see from the interviews we conducted. A recurring concern across many of our interviews ended up being the anticipated amount of human effort to maintain support for the Burst Buffer if adopted. From our conversations certain types of work, like simulations, could benefit from using this feature while other work like bioinformatics could not. As a stepping stone towards a future all flash memory file system the Burst Buffer demonstrated varying degrees of utility for HPC users.

Scientists running simulations benefit with this feature. Among our conversations scientists with traditional HPC workloads running large simulations were able to benefit from the Burst Buffer with fairly

minimal effort. Talking with one scientist their group found that adopting the Burst Buffer was an effective way to remove bottlenecks within individual nodes during their simulation runs. From this individual's perspective the Burst Buffer made their I/O bottlenecks go away. Not only did this feature help with the loading of large datasets at the beginning of a simulation run, it also made it possible for these scientists to write out data during check points without worrying that the I/O system would crash.

Evaluated, but adoption requires too much effort. Many of the scientists we talked to are building complex data analysis software for multi-national projects. These individuals noted that their projects tested the Burst Buffer when it initially came out, but in the end could not align the feature to their longitudinal visions and practices for work. Projects typically had a team member develop some experiments, sometimes in collaboration with NERSC, to benchmark the performance of their project's software with existing storage systems and the new Burst Buffer. They would typically find a small performance increase without doing much work to adapt their software, but not enough of an increase to justify expending the large amounts of effort required to fully incorporate this resource. Participants noted how incorporating this specific system feature would also result in maintenance headaches over time. These include needing to have a person keep code bases up to date and functioning in this unique environment, to more basic issues where the Burst Buffer itself may be unavailable when a job needs to be run.

Varied computing environments and project timelines hinder usage of unique features. Stakeholders working in multi-institutional, often multi-national, scientific collaborations face challenges adopting a facility specific feature beyond just the human time and effort. Large scientific projects can spend many years building an instrument, collecting data, and running analysis software. These projects often leverage computing resources from facilities across the country and/or world. In contrast, the procurement, development, operation, and retirement of an HPC is on the order of half a decade or more. This depends on the efforts of a facility and the agency funding the resource. Consequently, mismatches emerge between the timelines of the long-term work of science projects and the HPC resources provided by any particular facility. The variability in features among HPC systems across, and even within, facilities combines with mismatches in timelines to make it challenging for collaborations to take advantage of unique features on one system.

Multiple participants in our study work as part of multi-institutional and multi-national projects that rely upon computing resources around the world. Their teams build software and data systems that must function consistently across a variety of HPC systems. Domain scientists noted that their use of varying HPC systems for a project results in multiple streams of hardware and software features available across these different ecosystems. Each of these ecosystems shift features over time at different rates. As a result, the scientists and engineers in the projects design for the lowest common denominator of features readily available across the different HPC systems available for their work. These individuals also noted that their project's software stacks try to consistently use data management abstractions that handle any unique system features as much as possible. Multiple scientists managing project's software raised these challenges, noting that adopting one site's unique feature would have to offer a significant benefit to balance out these human time costs. Faced with Cori being one of the only machines with a Burst Buffer, members of these teams indicated they had low expectations about their potential ability to us this feature. One scientist commented *"I had very low expectations. ... the work that I'm doing ... has a global workflow system"* and as a result adapting this system to the Burst Buffer was unlikely to be rational given the potential headaches and increased amount of human labor.

Adapting to use the Burst Buffer also would have introduced a long-term maintenance burden, even though using this feature requires fairly simple adaptations to where data is referenced. Teams working globally with many different systems could not justify expending the human time to adapt their software to this one site's unique feature due to potential maintenance and reliability concerns. This is especially important for a core element of the work that must be reliable and without a fully seamless API participants anticipated too many hindrances to adoption. In spite of hesitance to incorporate the Burst Buffer into key software work, these same participants noted they had pondered hosting parts of their software stack on this hardware or using it for non-critical tasks. Employing this feature to run databases or smaller scale simulation workflows was one envisioned opportunity, letting the project take advantage of the Burst Buffer and its potential speed while not disrupting their essential, primary work.

Doesn't align with software designed to use local machine storage. Another issue that emerged from our conversations was a general mismatch between scientific software designed to use local storage temporarily and the construction of HPC systems which the Burst Buffer could have impacted. Multiple domain science software developers explained that much of the commonly used software in their field expects a computing system to have local disk storage available. This local storage is used by these tools to temporarily write data out before migrating to other storage. HPC systems generally do not have local storage on nodes, relying instead upon the shared file system. Our informants noted that initially they speculated that the Burst Buffer could stand in as a form of local disk since it is associated with particular nodes of Cori. One interviewee explained *"... we were hopeful for burst buffer and, uh...I would say...it's been very spotty successes."*

Adopting the Bust Buffer failed for these domain scientists and engineers. This was the case in part because their testing kept surfacing bugs with the Burst Buffer's software. Some of these bugs were resolved, but others stemming from fundamental design decisions about how this feature is made available to users were never resolved. These scientists also ended up not using the Burst Buffer because it did not improve performance with the various off the shelf software tools they have to support for their community. Third, the short-term nature of storing data on the Burst Buffer was a hindrance to improving the performance of jobs that rely on common, shared bioinformatics datasets. The Burst Buffer was designed so that most data is staged in and out when executing a job, with the ability to have short-term reservations holding data for a week or two at most typically. The work of these scientists may have benefited if some common data could be stored in the Burst Buffer for use during randomly executed jobs. Without a guarantee that the data would actually exist on this hardware when a job begins, if it is not directly staged into this component, then trying to store commonly used datasets was determined to be futile.

Overall, as developers ensuring pre-existing tools function in an HPC environment there is only so much time or money available for science software developers to adapt pre-existing software to the environment. The Burst Buffer was enticing in theory but given issues with its design in relationship to common tools these developers could not align it to their use case.

5 Summary

HPC ecosystems provide vital resources for large-scale scientific exploration with large volumes of data that are distinct from many of the computing systems familiar to most individuals. The facilities procuring and sustaining these resources work with diverse stakeholders to ensure everyday tasks can be accomplished. Ensuring these systems are usable for science stakeholders is essential to the mission of HPC facilities and our study identified various key concerns impacting usable HPC experiences. We interviewed domain scientists, computer engineers, and facility staff who work with HPC systems to explore many facets to user's experiences with these ecosystems.

We examined the easily black boxed nature of these large computing systems and the perpetual work users must do to understand how to effectively work with these resources. Abstraction layers can enable usability, but need to be maintained as system architectures shift and ensure user's can peer inside to identify the sources of errors. Without visibility into abstractions, in particular their error messages, users may end up frustrated and find this software tool not useful in their work. Users working in HPC ecosystems also must work with allocations of computing time to the projects they are a part of and the variable cost for running tasks on a system through a queuing system with this allocation. Similarly, challenges emerge for users who might want to adopt unique features deployed on one HPC system. In this case we examined the concerns science collaborations faced with the vision and deployment of the Burst Buffer in the Cori HPC. These collaborations work with diverse resources and find that adopting unique features is unlikely to be worth the effort of having an individual maintain a unique software stack and the risk of the resource being unavailable when work needs to be accomplished. Abstractions could enable projects to need unique hardware features but a facility would have to maintain this solution and ensure it is aligned with other data abstractions and made user friendly.

The experiences working with HPC systems that we've examined vary and highlight some of the diversity to work in these realms. Facilities must support a range of users while scientists themselves bring varied needs. Creating usable interfaces to these systems to ensure user experiences are positive remains challenging. Continued, systematic qualitative studies with these groups and examination of changes over

time is one way to support the development and maintenance of user friendly HPC ecosystems.

6 Acknowledgements

The authors wish to thank the members of the Usable Data Abstractions team and our anonymous study participants for their insights and feedback. This work is supported by the U.S. Department of Energy, Office of Science and Office of Advanced Scientific Computing Research (ASCR) under Contract No. DE-AC02-05CH11231.

References

[1] Bhimji, W., Bard, D., Romanus, M., Paul, D., Ovsyannikov, A., Friesen, B., Bryson, M., Correa, J., Lockwood, G. K., Tsulaia, V., et al. Accelerating science with the nersc burst buffer early user program. pages 17

[2] Charmaz, K. *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*, 2nd ed. Sage, 2014. pages 7

[3] Chen, N.-C., Poon, S., Ramakrishnan, L., and Aragon, C. R. Considering time in designing large-scale systems for scientific computing. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (New York, NY, USA, 2016), CSCW '16, ACM, pp. 1535–1547. pages 2, 7, 14

[4] Ghoshal, D., and Ramakrishnan, L. Madats: Managing data on tiered storage for scientific workflows. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing* (New York, NY, USA, 2017), HPDC '17, Association for Computing Machinery, p. 41–52. pages 2, 6

[5] Jirotka, M., Lee, C. P., and Olson, G. M. Supporting scientific collaboration: Methods, tools and concepts. *Computer Supported Cooperative Work (CSCW) 22*, 4-6 (2013), 667–715. pages 5

[6] Lawrence, K. A. Walking the tightrope: The balancing acts of a large e-research project. *Computer Supported Cooperative Work (CSCW) 15* (2006), 385–411. pages 6, 10

[7] Lawrence Livermore National Laboratory. VisIt. `https://wci.llnl.gov/simulation/computer-codes/visit/`. pages 3

[8] Lee, C. P., Dourish, P., and Mark, G. The human infrastructure of cyberinfrastructure. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work* (New York, NY, USA, 2006), CSCW '06, Association for Computing Machinery, p. 483–492. pages 3, 6

[9] National Energy Research Scientific Computing Center. About NERSC. `https://www.nersc.gov/about/`. pages 3

[10] National Energy Research Scientific Computing Center. Cori. `https://docs.nersc.gov/systems/cori/`. pages 5

[11] Nielsen, J. 10 usability heuristics for user interface design. `https://www.nngroup.com/articles/ten-usability-heuristics/`, Apr 1994. pages 6

[12] Paine, D., and Lee, C. P. Coordinative entities: Forms of organizing in data intensive science. *Computer Supported Cooperative Work (CSCW) 29*, 3 (2020), 335–380. pages 3, 5

[13] Paine, D., and Ramakrishnan, L. Understanding interactive and reproducible computing with jupyter tools at facilities. Report LBNL-2001355, Lawrence Berkeley National Laboratory, 2020. pages 5

[14] ParaView. ParaView. `https://www.paraview.org/`. pages 3

[15] Parnas, D. L. Information distribution aspects of design methodology. *Methods 4*, 5 (1971), 6–7. pages 11

[16] Ramakrishnan, L., and Gunter, D. Ten principles for creating usable software for science. In *2017 IEEE 13th International Conference on e-Science (e-Science)* (2017), pp. 210–218. pages 2

[17] Ribes, D., and Finholt, T. A. The long now of technology infrastructure: Articulating tensions in development. *Journal of the Association for Information Systems 10*, 5 (2009), 375–398. pages 6

[18] Ribes, D., and Lee, C. Sociotechnical studies of cyberinfrastructure and e-research: Current themes and future trajectories. *Computer Supported Cooperative Work (CSCW) 19*, 3 (2010), 231–244. pages 5

[19] Segal, J. Software development cultures and cooperation problems: A field study of the early stages of development of software for a scientific community. *Computer Supported Cooperative Work (CSCW) 18*, 5 (2009), 581–606. pages 6

[20] Sharp, H., Rogers, Y., and Preece, J. *Interaction design: beyond human-computer interaction*, 2nd ed. John Wiley and Sons Inc., 2007. pages 6

[21] Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N., and Diakopoulos, N. *Designing the User Interface: Strategies for Effective Human-Computer Interaction (6th Edition)*, 6th ed. Pearson, 2016. pages 6

[22] Steinhardt, S. B., and Jackson, S. J. Anticipation work: Cultivating vision in collective practice. In *Secondary Anticipation Work: Cultivating Vision in Collective Practice*, edition ed., Series Anticipation Work: Cultivating Vision in Collective Practice. ACM, 2675298, 2015, ch. Chapter, pp. 443–453. pages 6, 16

[23] Weiss, R. S. *Learning From Strangers: The Art and Method of Qualitative Interview Studies*. The Free Press, New York, NY, 1995. pages 6