Report from Dagstuhl Seminar 17202

Challenges and Opportunities of User-Level File Systems for HPC

Edited by

Andrë Brinkmann[1], Kathryn Mohror[2], and Weikuan Yu[3]

1 **Johannes Gutenberg University Mainz, DE,** brinkman@uni-mainz.de
2 **LLNL – Livermore, US,** mohror1@llnl.gov
3 **Florida State University – Tallahassee, US,** yuw@cs.fsu.edu

Abstract

The performance gap between magnetic disks and data processing on HPC systems has become that huge that an efficient data processing can only be achieved by introducing non-volatile memory (NVRAM) as a new storage tier. Although the benefits of hierarchical storage have been adequately demonstrated to the point that the newest leadership class HPC systems will employ burst buffers, critical questions remain for supporting hierarchical storage systems, including: How should we present hierarchical storage systems to user applications, such that they are easy to use and that application code is portable across systems? How should we manage data movement through a storage hierarchy for best performance and resilience of data? How do the particular I/O use cases mandate the way we manage data? There have been many efforts to explore this space in the form of file systems, with increasingly more implemented at the user level. This is because it is relatively easy to swap in new, specialized user-level file systems for use by applications on a case-by-case basis, as opposed to the current mainstream approach of using general-purpose, system-level file systems which may not be optimized for HPC workloads and must be installed by administrators. In contrast, file systems at the user level can be tailored for specific HPC workloads for high performance and can be used by applications without administrator intervention.

Many user-level file system developers have found themselves "having to reinvent the wheel" to implement various optimizations in their file systems. Thus, a main goal of this meeting was to bring together experts in I/O performance, file systems, and storage, and collectively explore the space of current and future problems and solutions for I/O on hierarchical storage systems in order to begin a community effort in enabling user-level file system support for HPC systems. We had a lively week of learning about each other's approaches as well as unique I/O use cases that can influence the design of a community-driven file and storage system standards.

The agenda for this meeting contained talks from participants on the following high level topics: HPC storage and I/O support today; what do HPC users need for I/O; existing user-level file system efforts; object stores and other alternative storage systems; and components for building user-level file systems. The talks were short and intended to be conversation starters for more in-depth discussions with the whole group. The participants engaged in lengthy discussions on various questions that arose from the talks including: Are we ready to program to a memory hierarchy versus block devices? Are the needs of HPC users reflected in our existing file systems and storage systems? Should we drop or keep POSIX moving forward? What do we mean when we say "user-level file system"? Do we all mean the same thing? How should the IO 500 benchmark be defined so it is fair and useful? and How are stage-in and stage-out actually going to work?

The report for this seminar contains a record of the talks from the participants as well as the resulting discussions. Our hope is that the effort initiated during this seminar will result in long-term collaborations that will benefit the HPC community as a whole.

Challenges and Opportunities of User-Level File Systems for HPC, *Dagstuhl Reports*, Vol. 7, Issue 05, pp. 97–139
Editors: Andrë Brinkmann, Kathryn Mohror, and Weikuan Yu

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Seminar May 14–19, 2017 – http://www.dagstuhl.de/17202
1998 ACM Subject Classification B.4.3 Interconnections, E.2 Data Storage Representations, E.5 Files, H.3.4 Systems and Software
Keywords and phrases High Performance Computing, I/O and Storage, Object Stores, User-level Storage Systems
Digital Object Identifier 10.4230/DagRep.7.5.97

1 Executive Summary

Kathryn Mohror
Andrë Brinkmann
Weikuan Yu

Seminar Overview

The primary goal of this Dagstuhl Seminar was to bring together experts in I/O performance, file systems, and storage, and collectively explore the space of current and future problems and solutions for I/O on hierarchical storage systems. We had a lively week of learning about each other's approaches as well as unique I/O use cases that can influence the design of a community-driven file and storage system standards. We also engaged in several informal, in-depth discussions on questions surrounding how we should best move forward in the I/O and storage community.

A portion of agenda for this meeting was partitioned into sessions containing short talks. The short talk sessions were grouped into high level topic areas: high performance computing and storage systems today; user needs for I/O; user level file system implementations; object stores and alternatives; and file systems building blocks. The intention behind the short talks was to acquaint the attendees with each other's work and to inspire further discussions. Following each talk topic, we had panel-style discussions with the talk speakers serving as the panel. In these panel-style discussions, the audience had the opportunity to ask questions about the speakers' talks as well as note and discuss commonalities and differences across the presentations.

The remainder of the agenda for the meeting was reserved for open discussions with the entire group. The participants engaged in lengthy discussions on various questions that arose from the talks. Additionally, participants were encouraged to propose and vote for discussion topics on a white board. The proposed topics with the most votes were included in the agenda. The in-depth discussion topics included:

- How are stage-in and stage-out operations actually going to work?
- How can we fairly judge the performance of storage systems – IO 500?
- What is a user-level file system? What do we mean when we say that?
- How can we characterize what users need from storage systems?
- Are we ready to program to a memory hierarchy versus block devices?
- and What should we do about POSIX?

The combination of short talks and open discussions resulted in a fruitful meeting. Since the work of the participants was not necessarily familiar to all, the short talks provided a foundation for getting everyone oriented with each other's efforts. Once that was achieved, we

were able to productively dive into the informal topic discussions. Overall, several common themes emerged from the talks and discussions. The participants agreed that these themes were important to address to meet the needs of HPC applications on next-generation storage systems. We include these themes in this report in Section 9 to serve as suggestions for further investigations.

Report Organization

Here we present an overview of the topics in this report to guide the reader. Our goal in this report is to capture as much information as possible from the seminar so that those who could not attend can benefit from the talks and discussions.

We detail the short talk sessions in Sections 3-7. First, we provide a summary of the notes from the session note taker and other comments from the talks and panel discussions. Following this summary we provide a listing of each talk in the session and its abstract.

In Sections 8.1-8.6 we give summaries of the informal discussion sessions. The summaries in this case are in outline format in order to capture the conversational and informal nature of the sessions. In many cases, the discussions drew out many interesting questions instead of clear paths forward, so the outline format captured this well.

Following the summaries of the sessions, in Section 9 we conclude with a discussion of recurring themes, including issues for future discussion and work, that occurred during the meeting. We feel that these themes are the true product of this meeting and can serve as a foundation for future meetings or other community efforts.

2 Table of Contents

3 Talks Session: High Performance Computing and Storage Systems Today

HPC is undergoing several transformations at the same time, where many of them include storage systems and technologies as a driving component:

- The growing gap between the performance of magnetic disks and processors drove the introduction of Solid State Drives (SSDs) as an additional layer in the storage hierarchy. SSDs are more expensive for the same storage capacity than traditional magnetic disk drives, but they are more cost efficient with respect to small I/O performance. Back end parallel file systems of HPC clusters with capacities in the range of tens to hundreds of PBytes are therefore still mostly based on magnetic disks, while SSDs are used as burst buffers to cache data, which can be implemented as either node-local storage or as shared storage external to compute nodes.
- The workload on many HPC systems is changing from pure number crunching to also include data intensive computing, which uses data parallelism to process data sets with a size of many PBytes per application run. Principles of data locality and data caching borrowed from big data approaches help to improve overall application performance, making burst buffers with low access latencies especially attractive.

Therefore, the first session of this Dagstuhl seminar focused on the big picture of HPC architectures and the current (and possible future) usage of HPC storage systems.

Thomas Böhnisch started with a presentation of the storage system at the High-Performance Computing Center HLRS in Stuttgart. The center focuses on engineering and industry applications and 80% of the applications periodically output data, while this data is not necessarily intended as defensive checkpoint data. In his talk, **Some Thoughts on Todays and Future I/O** Thomas described use cases, where applications spend more than 1.5 days of 2 days runtime in I/O even after code optimization, which had been able to increase file system bandwidth usage from 1 GByte/s to 7.5 GByte/s. The bandwidth usage nevertheless is only 1/10-th of the backend storage bandwidth available in Stuttgart, where the storage systems delivers up to 75 GByte/s. The main reason seems to be that many applications developers only optimize on one aspect of the code, which rarely is I/O.

Applications seem to become, according to Thomas, a black box for the scientific users, who have no knowledge of the I/O performed by the application and even much less of optimal file system settings. Thomas presented burst buffers as a possible way to overcome challenging and bursty I/O patterns, but also mentioned that they have to be seamlessly integrated into the HPC environment, as users want to perform their science and as they are not interested in I/O. Furthermore, he identified storage class memory as a possible game changer, which will impact the working methods in HPC, while there is quite some research and development necessary to efficiently use it.

Mark Parson from the Edinburgh Parallel Computing Centre (EPCC) started his talk **File Systems for HPC** with the observation that the complexity of storage systems has greatly increased in the past decade. Most HPC systems now have multiple storage systems and multiple file systems per storage system, while GPFS and Lustre dominate as backend parallel file systems. Resiliency stays an important aspect of storage management and most HPC centers have at least one storage crisis a year.

The dominant usage scenario seen in Edinburgh is that most users use the POSIX file system interface and do not include parallel IO libraries. Layered I/O (NetCDF over HDF5 over MPI-IO) approaches on the other side fight with each other by providing optimizations at each layer.

Buying storage is, according to Mark, still terribly complicated and confusing. The bandwidth requirements are mostly dominated by the start and end-phases of jobs as well as by checkpointing, which are very read-write-intensive, and which therefore waste investment. Furthermore, performance degrades over time (e.g., due to file system aging effects) and scientific users often *miss-use* parallel file systems. Examples from genome processing include that one user moved around 400 TB through storage every week, while others created more than 240 million files in a directory within one year, stressing both Lustre and GPFS. It would be therefore beneficial to have parallel I/O benchmarks, which can be used for procurement purposes, including different HPC usage scenarios.

Mark's talk agrees with Thomas in that a new hierarchy of next generation NVRAM technologies will profoundly change memory and storage hierarchies, while HPC systems and data intensive computing systems will merge. Profound changes will therefore come to all data centers, while HPC centers still need to develop software – OS and application – to support their use. This change of HPC storage has to be seen in the context that parallelism beyond 100 million threads demands new I/O approaches, where reading and writing data to parallel file systems still seems a major bottleneck and data movement and processing on capability systems has to be re-thought. Mark sees a demand for truly parallel file systems with reproducible performance, while current technologies simply will not scale, leading to large jobs spending hours reading data and writing results.

The talk **HPC Filesystems Today – What's Working and Opportunities to Improve** from Ned Bass is based on the experiences gained running seven Lustre file systems being in production at Lawrence Livermore National Laboratory, hosting more than 50 PByte on the open side. Ned started with a summary of well-working aspects, including the Open Source development of Lustre and ZFS (even while pushing changes up-stream is difficult). Stability has furthermore improved according to him a lot compared with the past and it is difficult for him to remember the last time when LLNL has lost user data. Furthermore, it is in many cases not the fault of the file system if something goes wrong, but even in such cases users often perceive the file system as a root cause for a problem. Another advantage of today's storage systems is that the POSIX programming is well understood and is portable from a laptop up to a supercomputer. Relaxing POSIX would therefore also mean that it becomes necessary to build a compatibility layer so that people can still perform application development on their laptops.

A further observation from Ned is that well-formed I/O also performs well. Nevertheless, the definition of *well-formed* depends on the file system in use and is not even portable between Lustre and GPFS. Additional challenges are, according to Ned, that the storage hierarchy is not transparent to the end users and that it is a heavy burden for them to understand where data lives and where it *should* live. Ned sees inflexible semantics, where the system decides where the data stays, as one of the obstacles and wants a clean interface for moving data between the storage hierarchies.

He agrees with the former speakers that users do not care about I/O and assume the technical staff to make it work and fast. Users spend very little time on I/O optimization, as long as it stays below 10% of the runtime they do not care. Managing data on the other side (including data visibility and debugging) requires a high degree of expertise from developers and system administrators, who could benefit from an API covering the complete storage hierarchy. Deciding on such an API is on the other side a big investment, as it can incur high technical debts if it is not well thought through.

The influence of SSDs and burst buffers on the storage architecture of HPC centers has been the core of John Bent's talk **Three Musings from Bent at Dagstuhl**. The talk describes the change of the storage architecture, starting with the two-layer architecture

consisting of parallel file systems and tape archives, which has been dominant from the early 2000's until 2015. Both, aligned and unaligned storage accesses hit the parallel file system, while I/O to tape has been typically well-formed (based on any definition).

The price advantage of SSDs concerning I/O rate and bandwidth led to the introduction of burst buffers, which are able to absorb the unaligned IO traffic, while changing this traffic into well-aligned accesses to the parallel file system. The next layer, which is already becoming part of many HPC installations, is an object storage layer for data which is neither hot (and therefore placed on the parallel file system or the burst buffer) nor cold (and therefore being stored on tape). This lukewarm storage occurs in the context of simulation campaigns, where huge data volumes are analyzed over time spans of many months.

Object storage has become prominent in the context of Cloud computing and has the advantage of nearly unlimited scalability, while offering the relatively low latencies and sequential access bandwidth of (highly-parallel) magnetic disks. A major additional advantage of object storage is furthermore its manageability, which, according to John, helps to decrease overall cost even below tape costs.

These four layers are (from most attendees' perspectives) too many layers for users. John's proposal is therefore to radically change the storage architecture and to drop the backend parallel file system and the tape archive and to keep hot data in burst buffers, while storing all cold to lukewarm data in object storage. John closes his presentation with a comparison of different possible locations of burst buffers, including node local storage as private burst buffers (e.g., in Cray/Intel Aurora at Argonne), as node-external or shared burst buffers (e.g., deployed in Cray Trinity at LANL) or embedded burst buffers (e.g., being part of Seagate Nytro NXD).

The talk **Data Movement Requirements for HPC and Data-Intensive Burst Buffers** from Dean Hildebrand focuses on the requirements to integrate burst buffers into the HPC environment. He argues similar to the previous speakers that burst buffers (and therefore SSDs) have not mainly been integrated based on their superior performance, but mostly based on their lower costs per (unaligned) IOPS. Dean therefore discusses workloads and memory access models, which are suited for burst buffers and asks for the necessary namespace integration approach, the durability, capacity, and availability requirements and the best API to use burst buffers, whether it might be POSIX, key value or object.

Transferring data in and out of burst buffers is seen as an unsolved problem by Dean. Example applications from the domain of deep learning, like Caffe, Attila, or TensorFlow are putting high pressure on the storage backend, where input data sets range from ten GByte up to tens of TByte. The entire input data sets are re-analyzed dozens of times and even intermediate writes can easily grow to tens of TBytes. Furthermore, it is often not clear which data is required where, so that all data is copied to all nodes.

The talk therefore identifies several requirements to efficiently integrate burst buffers into HPC designs, especially in the context of new applications. It is obviously important to reduce the bandwidth between burst buffers and the durable tier, while it can be very beneficial to include application specific direct data placement. Burst buffers can furthermore only be efficiently used if the network bandwidth scales with the capacity of the burst buffer. Unsolved challenges also include efficient scheduling strategies (and to couple them with HPC batch environments), which help to identify for each data set when it has to be transferred where. Especially node-external data sets also impose new security challenges and demand for multi-tenancy features.

In summary, Dean sees burst buffers as caches, not primary data stores, which are useless without data, while efficiently transferring data in and out of burst buffers seems to be still an unsolved challenge.

The talks discussed so far focused on technology transitions, which are already visible in today's HPC environments. Jay Lofstead's talk **Evolving Machine Architectures Are Shifting Our Research Agenda – We Need To Keep Up!** discusses the impact of new memory and storage technologies based on byte addressable storage class memory (SCM), which builds another layer into the memory/storage hierarchy. SCM is able to not only blur the dividing line between memory and storage, but to obliterate it.

A challenge seen by Jay is that the HPC community has not yet adequately solved the problems inherent to the architectures being deployed today, not to mention those of the future, where memory becomes persistent and networking becomes part of the memory hierarchy instead of just the storage hierarchy.

Fundamental questions during the transition to SCM include the interface to SCM. Can we keep files as an abstraction if the POSIX semantics cannot be kept without inducing major performance losses? How do we identify a collection of bytes in the absence of files? Do we still call it storage if we use CPU-level get/put operations? Additional challenges during the transition include keeping consistency, coherence, and security, which are currently provided by file systems.

Jay describes four transition phases, where phase one already started in the late 1990's, when extra compute nodes have been used for their memory as data staging areas, building the origin of *burst buffers*. Flash as burst buffers in or near the IO path has been the starting point for phase 2, including rudimentary job scheduler support for data pre-staging, and data draining.

Phase 2 is still on-going and explores additional use cases for Flash, e.g. as part of the compute nodes. This new architecture might lead to interferences, as accesses impact the network, memory or disk bus of the local node. Phase 3 will provide additional node local storage, e.g. as persistent 3D Xpoint, while nodes also gain on package high-bandwidth memory (HBM). Challenges in phase 2 and 3 are the interconnect speed, erase cycles in Flash and 3D Xpoint, maintaining coherency and consistency for multi-user support and globally shared spaces.

Phase 4 of the architectural transition will then provide a memory-centric design with network storage (on switches), DRAM (potentially in the same address space). The line between memory and storage will therefore be all but gone. Typical use cases will be coherent virtual fat nodes operating on 10s of TBytes, the introduction of persistent storage near/fast enough to "swap" to, while hopefully providing new and easier programming models to access data.

A possible dividing line between memory and storage might be (according to Jay) that (today) things placed in memory have external metadata, which is generally expressed in the program code. Memory therefore has a compact representation, which is optimized for the interaction with processors. Things placed in storage are (today) wrapped in metadata to make them easily usable by other applications. File formats make it possible to read in simulation output into visualization tool, even in case of a different endianness.

Franz-Josef Pfreundt tries to tackle the new challenges from two different perspectives. His talk **Thoughts about the future of IO** presents BeeOND (a sibling of BeeGFS) as a burst buffer file system to efficiently use node internal SSDs. BeeOND can build a temporary file system across nodes (on demand per user), where every compute node can become a metadata server. Possible approaches, e.g., to deep learning (DL) I/O challenges are to combine all files in one large binary with fixed offsets.

Data management is seen by Franz-Josef as one key problem in parallel computing, as it is too complicated for application developers. He therefore also proposes new communication interfaces (in his case the GPI-2 Global Address Space Communication Interface) with explicit one-sided communication with notification.

GPI provides a standardized API (GASPI) and hides latencies by asynchronous one sided RDMA communication. Memory virtualization is provided by GPI Spaces, which can keep data without a running application being responsible for the data. Data exchange between applications and the virtual memory is performed through shared memory segments, while data transfer between nodes is provided by GPI. GPI Space is currently becoming a complete distributed runtime system, which provides failure tolerance, JIT compilation and execution of the underlying Petri nets, co-scheduling of multi-node tasks and preemptive scheduling of data transfers.

The presentation included two use-cases, where the combination of GPI-Space and domain knowledge helped to overcome application problems. SPLOTCH, a MPI visualization program from the astrophysics domain, has been rewritten in GPI-Space in three weeks, while being able to improve the application runtime by a factor of more than three in the worst case. A Deep Learning on Demand Architecture has been presented as a second use case, which uses Amazon's spot market to provide automatic meta-parameter search and which offers automatic data-management, while still supporting the original DL model descriptors, e.g. from Caffe or Tensor Flow. Future work includes an architecture that offers virtual memory directories and caches based on a client-server architecture, while automating the transfer between storage and (persistent) memory.

3.1 Some Thoughts on Todays and Future I/O

Thomas Bönisch (HLRS – Stuttgart, DE)

The presentation shows the three main issues of user applications with a current file system implementation at the Hazel Hen HPC system in Stuttgart. The first issue is that applications very often cannot make use of the file system performance. We discuss the reasons for that depending on the user group. The second issue is that I/O is in bursts and third, several applications tend to generate millions of files. In addition to these issues, we discuss a new approach for data management and what this could mean for user applications.

3.2 File Systems for HPC

Mark Parsons (University of Edinburgh, GB)

From a data centre point of view the quantity and complexity of data we are being asked to manage is growing all the time. Coupled to this are complex requirements around security – particularly related to authorisation, accounting and auditing – who uses what data, when, and why. In the High Performance Computing and Data Analytics world, new memory technologies are on the immediate horizon that will markedly change how use and manage data in both the research and commercial worlds. We need to think carefully about what these competing demands mean for file systems and how we can provide rich environments for data manipulation and its transformation into knowledge in whatever sector it is being used.

3.3 HPC File systems Today – What's Working and Opportunities to Improve

Ned Bass (LLNL – Livermore, US)

In this talk I provide an overview of the HPC storage environment at LLNL, and I discuss successes and challenges with the current state of the art. Aspects that are working well include stability, data integrity, security, scalability for petascale class systems, a well understood portable programming model, and thriving open source collaborations. Some major challenges also exist in HPC file systems today. Users are exposed to internal implementation details of the storage hierarchy which would be better hidden behind intuitive abstractions. Options to tailor API semantics and consistency models to the needs of the application are limited. Users bear a heavy burden to curate their data and manage its movement throughout the storage hierarchy. A well-designed interface would allow users to provide hints about the expected life span and access time requirements of their data and the system would do the right thing on their behalf. Finally, due to their complexity and aggressive pace of development, current HPC file systems carry high total cost of ownership and significant technical debt. Now is the time to lay the groundwork for the maintainability of future systems through careful design and thorough documentation. As we move toward new storage paradigms to address current challenges we should take care not to lose ground in the areas that are working well today.

3.4 Three Musings from Bent at Dagstuhl

John Bent (Seagate Government Solutions – Herndon, US)

One, the grind-crunch model of computational simulation and what would you do with one more dollar. Two, from 2 to 4 and back again; the evolution of HPC storage architectures. Three, to share or not to share; a comparison of burst buffer architectures.

3.5 Data Movement Requirements for HPC and Data-Intensive Burst Buffers

Dean Hildebrand (IBM Almaden Center – San Jose, US)

Over the next several years, all applications—regardless of industry—will begin to expect the bandwidth and latency of fast storage devices such as NVMe and 3D XPoint. A discrepancy therefore exists between the high-performance expectations of users vs. what their budgets can contain. This means that dataset movement between slower durable storage and fast storage will become an integral part of the storage hierarchy on-premise and in the cloud. This is further complicated by the diverse set of storage software and their APIs that exist at

both the fast and durable tiers (e.g., NAS, object, block, parallel file systems, tape, optical), most of which struggle to communicate and share data. In my talk, I will discuss the growing relevance of "bust-buffer'-like data movement requirements across a variety of industries and their emerging requirements.

3.6 Evolving Machine Architectures Are Shifting Our Research Agenda – We Need To Keep Up!

Jay Lofstead (Sandia National Labs – Albuquerque, US)

While IO/storage research has slowed in recent years, the bits that still come out tend to be clustered in incremental improvements to two-phase collective IO, isolated traditional storage aspects such as metadata improvements, or trying to understand the chaotic IO patterns in an attempt to predict and/or schedule IO more efficiently. While there is some value in these research areas, they are missing the real problems we will face soon – changing underlying storage/memory technologies and machine architectures. With SSDs creeping into HPC platforms and NVM over fabric being a near term product, we in the IO community need to focus more efforts to address these challenges or risk losing relevance to the programming models community. This talk will explore why the current topics outlined above are less useful or even harmful to continue to investigate while laying out a case for why we need a serious shift in focus to stay relevant.

3.7 A future I/O concept

Franz Josef Pfreundt (Fraunhofer ITWM – Kaiserslautern, DE)

With the appearance of large memory machines and the future dominance of fast storage class memory we have to rethink parallel I/O and parallel programming. At Fraunhofer we work on two software systems that are intertwined with respect to this topic. GPI-Space is a system to automate parallel programming in using a global virtual memory space and petri nets to express parallelism. Today we can map already the virtual memory space into our parallel file system to extend the size and add failure tolerance. BeeGFS, the parallel file system that we develop, is a state of the art POSIX compliant parallel meta file system. How we can merge the two systems is our topic for the future. At the end, the goal must be to manage data transport from disks to HBM fully automatic.

4 Talks Session: User Needs for I/O

Technological progress can only benefit us if it is in line with our needs. New device technologies and faster supercomputers are therefore only useful if they enable users (here in the form of scientists) to perform better research. This includes several different dimensions.

First of all, science has to demand for faster supercomputers to justify growing investments. The widening HPC user community in research as well as the broad application of HPC in industry seems to be a good indication for this demand. Second, it is important that the new capabilities are provided to the scientist in a way that they can be easily integrated into their scientific workflows, so that scientists do not have to become HPC experts, which would distract them from doing the research. Third, applied HPC research should focus on the current pain points of researchers, optimizing scalability challenges.

The presentation **What do we and the users know about I/O?** from Michael Kluge started with an overview about I/O usage data which is typically collected and analyzed in HPC data centers. The reason for this data collection is that I/O is a shared resource, where individual users can abuse the file system in a way that interferes with concurrently running jobs of other users. This interference can become in practice a denial of service attack, as HPC users have often not been taught to program in an I/O friendly manner, while there mental model of I/O is not compatible with the real HPC (storage) hardware.

The positive news is that there are today plenty of data collection tools available, which help to get an overview about, e.g., how much data has been read or written from which user at a specific bandwidth. There are also many performance counters integrated, helping to also get detailed information about additional metadata usage. This profile data helps system administrators to get a good overview about quantity and quality of storage usage, but it does not immediately help us to learn about which new interfaces are needed and how we can modify current applications to become more I/O-friendly.

A hypothesis within Michael's talk was that we do not only need new file system interfaces, but that many new storage interfaces with different semantic variations of the POSIX interface are required. New semantics can help to find a sweet-spot between avoiding performance bottlenecks (which are induced by today's file system implementations) and changing as little as possible for users. The main questions seem to be how much the file system interface has to be changed for specific applications and whether application experts, users, and system administrators are able to give convincing answers to this question.

It seems obvious that the answers to these questions are not easy, as changing the API requires knowledge about what semantics an application relies on when using a storage interface. This does not only includes the file system itself, but also I/O libraries and abstraction layers like burst buffers, I/O forwarding nodes, or automated workflow engines. It is very likely in this context that the knowledge about I/O usage decreases with an increasing depth of the I/O stack.

Michael proposes two possible approaches (as well as combinations). The formal approach requires the developer to describe the kind of features needed, while I/O libraries and file systems describe the features which they provide. Both lists can be matched with each other, offering to the application the storage interface with the minimal set of features required (and therefore also hopefully with high performance). The technical approach tackles the challenge per use case by system wide I/O tracing and analysis. This tracing can then hopefully reveal all dependencies, but would be limited to this specific execution.

Dave Montaya highlights in his talk **User Workflow Use Cases** a trend seen in many HPC data centers: physical insights are not gained by a single application run, but by an ensemble of applications working together within a workflow. Running large suites of simulations is according to Dave already routinely done at the Los Alamos National Laboratory (LANL) to study the effects of (usually small) changes to problem geometry and/or materials. Furthermore, workflows help to continually assess the viability of new physics models, meshing strategies or code settings that are not related to physical changes to a problem while they enable rapid simulation turnaround, repeatability, and consistency.

The data infrastructure is according to Dave one of the pain points, as it is continuously evolving. The well-understood hierarchy of memory, parallel file systems, and archive is already including additional storage layers, like burst buffers and campaign storage, while the boundaries between these layers will blur with the introduction of byte-addressable storage class memory. These changes in the infrastructure have also to be addressed by changing the workflows underlying the simulation runs.

Dave also sees challenges concerning the large sets of input data with complex dependencies. This input data includes the syntax of the physics code, the common model data, as well as suite specific data, which overrides the common model and gives explicit meshing instructions and provides execution strategies, while the shot specific data includes additional overrides for the specific workflow run.

These challenges have to be seen in the context of the size of large scale capability workflows. Dave explains, based on the example of the Hydra UQ workflow to support ignition experiments at the National Ignition Facility (NIF) in Livermore that each study run can include up to 60,000 simulation runs within each single allocation, generating over one billion synthetic x-ray images over a runtime of 8 weeks. Each of these simulation runs generates new files every few minutes, leading to more than 5 PBytes of intermediate files. Even if the average bandwidth of 40 GByte/s does not saturate the backend Lustre file system, it would have already been very useful to be able to integrate burst buffers running at higher peak bandwidth.

The talk **The SIONlib Multi-file Container Approach for Massively Parallel Task-local I/O** by Wolfgang Frings moves from the problem definition to providing solutions for a specific challenge. Many applications running on JUQUEEN in Jülich (and on many other HPC centers) generate task local checkpoint files, while I/O is handled by separate I/O nodes in each rack. Generating a checkpoint from applications running on up to 458,752 cores can therefore easily flood the I/O nodes as well as the underlying parallel file system. Wolfgang presented graphs showing that generating one file from each JUQUEEN core can already take more than four minutes, while assuming 64 tasks on each node easily leads to file `create()`-times of more than 10 minutes.

The main approach of SIONlib is to provide an interposing layer, which transparently translates the $1:1$ relationships between clients and files into a $n:1$ relationship, so that n clients share 1 file. The number of new files can therefore be reduced to a few, significantly decreasing the `create()` time. SIONlib shows on the other hand a tradeoff concerning the number of files, which are shared by many clients, and the write bandwidth accessing these files. Wolfgang showed that the `create()`-times can be reduced to a few milliseconds if all threads share a single file, while at the same time the bandwidth to this shared file significantly decreases with the number of accessing tasks.

SIONlib therefore introduced a multi-file approach, which maps clients to I/O nodes, so that no I/O node gets overwhelmed by accesses of too many clients. Applications with an 1:1 access pattern like checkpointing still work by simply linking the SIONlib library achieving up to 67% of the peak bandwidth for 1.8 million tasks. SIONlib is already integrated into many HPC applications like multigrid solvers (DUNE-ISTL), fusion simulation (ITM), or applications to simulate the human brain (NEST).

The new Buddy CP approach also integrates local storage within cluster nodes to further increase possible bandwidth utilization. Data is first written to the local storage of the node, where a process is running, while it can be later (transparently) redistributed to a buddy node within the cluster or to the global file system.

Nathan Hjelm's talk **libhio: Optimizing IO on Cray XC Systems With DataWarp** tackles a similar challenge from a different perspective: How do we use new technology if it is provided to us? Cray's DataWarp technology is a burst buffer implementation, which has been initially developed for the leadership computers Trinity at Los Alamos National Laboratory and Cori at NERSC. The DataWarp implementation at LANL consists of service nodes including one Xeon E5 processor and two 3.2 PCIe SSDs each, which are directly connected to the Cray Aries network. The software stack includes an API with functions to initiate stage in / stage out and to query stage state. DataWarp can be configured in multiple modes using the workload manger, e.g., to stripe data over multiple nodes, to use it as a cache or in shared / private mode.

The Hierarchical IO Library (HIO) has been developed at LANL to facilitate the use of burst buffer technologies and specially to decouple application development from today's storage implementations, so that they can still efficiently run on next generation hardware. The development has been user-focused, so that it can be easily incorporated into existing applications and help to improve checkpoint performance for $n : 1$, $n : n$, and $n : m$ IO patterns.

HIO is centered around an abstract name space and each HIO *file* is a *dataset*. HIO datasets support named binary elements and a shared or unique-per-rank element offset space, where an offset space is defined per named element. The internal on-disk structure is not yet specified to preserve flexibility for future optimization. The currently supported "File Per Node"-mode has been originally targeted for Lustre and is intended to reduce the metadata load compared to traditional $n : n$ approaches, while leading to less file contention than the $n : 1$ approach, leading to similar optimizations like presented for SIONlib. Cross-process data lookup is supported via MPI-3 Remote Memory Access (RMA).

Scott Klasky's talk on **Self Describing Data** abstracted from today's user requirements and provided an outlook on how data can be better (re-)used. The talk started (after a short outlook on ADIOS 2.0) with the observation that raw data is of little use if it is not accompanied with an appropriate description. The costs for recovering information from such datasets typically increases linearly with the size of the dataset. Self-describing datasets should simplify the access to information by annotating the data with easily and fast accessible metadata.

An important challenge is to find the extra annotations to make data more valuable without adding so much information that it greatly affects performance. Self-describing data is thereby nothing completely new, examples of self-describing files include MS Word, JPEG, HDF5, or ADIOS. ADIOS, e.g., includes a self-describing BP file format, which enables to read arbitrary subarrays of variables, so that variables being written out from n processor can be read in by an arbitrary number of processors.

Self-describing data is especially useful for queries, as searching through the data can be largely simplified by its self-descriptive nature. Data analysis for decision-making can therefore be performed on a "need to know" information basis. Metadata can include information on min/max-values of temperatures, power consumptions, or humidity for certain data blocks, so that blocks with important data can be easily derived and further analyzed. It is also in principle possible to put time constraints on queries, e.g., to get the best possible information which can be collected in one hour.

Potential performance problems using self-describing files include that it can become expensive to capture global metadata when the problem size grows. ADIOS partially solves the problem by maintaining redundant metadata. The redundancy mechanisms allow ADIOS

to regenerate metadata in case of failures and in case of performance issues when writing. Metadata can furthermore kept locally, while a (virtual) global metadata file can be generated for online operations.

(Virtually) Global metadata makes it possible to first read through the metadata and then to only move data needed based on metadata operations, which is important for scientific workflows including multiple sites. The proposed metadata analysis is in this case cheap compared to the costs incurred by blindly moving huge datasets. Using self-describing data for staging can enhance data services and communication among applications providing an intermediate common area (staging) that reduces file system access costs, while it can include plugins to perform analytics and visualization, data reduction, or data transport.

Coupling tools like ADIOS or HDF5 enables application developers to present their data based on a common schema, so that tools in a workflow can cooperate with each other without providing data transformations for every new coupling between applications. Refactoring data to fit into schemas also enables applications not only to read data which is important, but also to perform on the fly transformations, e.g., by reducing the accuracy of certain variables, as this does not only decreases transfer, but also later processing costs.

4.1 What do we and the users know about I/O?

Michael Kluge (TU Dresden, DE)

Research file systems relax POSIX semantics in different ways. The questions is, whether there can be a common description about what semantics a certain file system provides. My talk has an proposal about what kind of information is needed to make proper decisions about whether or not a research file system fits to a given use case.

4.2 User Workflow Use Cases

David Montoya (Los Alamos National Lab., US)

Understanding simulation study workflows has become more important as HPC architectures and computational scales increase. There is a need to communicate application patterns and behaviors to vendors and performance insight to application developers. We have started a methodology and tools to describe and collect metrics to support this need. Additionally we describe use cases that show how workflows, which include pipelines of application runs are evolving and changing to adapt to architecture changes. The data management and provenance of these workflows are creating the need for a distributed view of data interdependencies.

4.3 The SIONlib Multi-file Container Approach for Massively Parallel Task-local I/O

Wolfgang Frings (Jülich Supercomputing Centre, DE)

The talk will focus on the multi-file container approach of SIONlib, a parallel I/O library, which addresses the problem of large number of task-local files by transparently mapping the task-local files onto a small number of physical files. Among other optimizations SIONlib uses a virtual shared container which is divided into a small number of physical files to represent a user based file system for parallel task-local I/O on application layer. Using multiple shared files instead of one big file allows to benefit from hierarchical storage systems, for example, by managing these files on node-local storage. In addition, recent developments within the EU-project DEEP-ER will be shown, where a combination of SCR and SIONlib is used to implement efficient multi-level checkpointing based on shared file containers. Within this project SIONlib was extended with a checkpointing option, based on the multi-file approach, which allows to store additional copies on buddy-nodes or to store parity information on network attached memory.

4.4 libhio: Optimizing IO on Cray XC Systems With DataWarp

Nathan Hjelm (Los Alamos National Laboratory, US)

High performance systems are rapidly increasing in size and complexity. To keep up with the Input/Output (IO) demands of High Performance Computing (HPC) applications and to provide improved functionality, performance and cost, IO subsystems are also increasing in complexity. To help applications to utilize and exploit increased functionality and improved performance in this more complex environment, we developed a new Hierarchical IO (HIO) library: libhio. In this paper we present the motivation behind the development, the design, and features of libhio. We detail optimizations made in libhio to support checkpoint/restart IO workloads on the Trinity supercomputer, a Cray XC-40 at Los Alamos National Lab. We compare the large scale whole file read/write performance or IOR when using libhio against using POSIX when writing to Cray DataWarp.

4.5 Self-Describing Data

Scott Klasky (Oak Ridge National Laboratory, US)

There is a critical national need to enhance economic development using high-performance computing. The exascale challenge will transition HPC science to a "predictive" science which means we must be able to extract and preserve sufficiently accurate output from simulations. Exascale systems are projected to provide a far greater increase in computational speed

than in I/O bandwidth, compared with petascale systems. This creates a time-critical need to optimize I/O to make sure that essential data generated by science applications can be written and read from the storage system. The data generated by these applications are extremely valuable, since the cost of running at scale on the average system on DOE cost over $200K USD per day (today), and is expected to become even more expense. Therefore, we argue that the data should be annotated in a self-describing storage format, such as ADIOS, HDF5, netcdf, etc. There are many challenges to do this at scale, and make this ubiquitous among HPC applications, but generally these lead us to understand how to make this easy to use and easy to achieve extreme portable performance for "most" of the I/O patterns from large scale scientific applications. Furthermore, since many of the newer exascale requirements are pushing us into the land of on-line analysis, we argue that the data formats used in these streaming/staging techniques should also use this self-describing data streams to allow a new data software-eco-system to develop and be nurtured through our efforts.

5 Talks Session: User Level File System Implementations

The third session of the Dagstuhl seminar focused on the development of user-level file systems to meet various needs of HPC applications and systems. Yue Zhu a second-year doctoral graduate student from Florida State University started this session with her talk on **Direct-FUSE: A User-level File System with Multiple Backends**. Her talk begins with a comparison of user-level and kernel-level file systems on their development complexity, reliability and portability. Yue enumerated a number of file systems that are implemented as user-level libraries to avoid the involvement of FUSE kernel module, but argued that an application may face problems when having to work with multiple backend file systems and their distinct file paths and descriptors. She proposes Direct-FUSE to provide multiple backend services for one application without going through the FUSE kernel. While FUSE alleviates the difficulty of developing a kernel-based file system, she recognizes the need of addressing the significant overheads for FUSE I/O function calls. Through a detailed analysis on the costs from context switches, data movements and metadata operations, Yue elaborated her approaches to alleviating the costs and achieving an efficient implementation of Direct-FUSE. Her results demonstrate an effective prototype of Direct-FUSE that supports multiple backend file systems.

Alberto Miranda from the Barcelona Supercomputing Center of Spain then presented his talk on **echoFS: Enabling Transparent Access to Node-local NVM Burst Buffers for Legacy Applications**. He reiterated the I/O challenge faced by petascale and future exascale systems and compared the two options of introducing burst buffers into HPC systems for data-intensive simulation and analysis. Alberto considers that node-local NVM-based storage to compute nodes provides denser burst buffers and offers an opportunity to construct temporary file systems with ad-hoc I/O optimizations for specific batch jobs. Through a research project NextGenIO that is funded by the European Union, Alberto and his team introduce a new user-level file system called echoFS that aims to aggregate the NVM burst buffers available to compute nodes into a collaborative burst buffer and then coordinate with the job scheduler to preload a job's data dependencies for first run-time I/O and simulation performance. While echoFS can benefit from explicit user inputs, echoFS can glean implicit hints from the SLURM batch submission scripts without placing a burden on the application users. In addition, echoFS a a pseudo-randomized striping strategy that aims at distributing

data across nodes for elastic load balance, and scalable aggregated I/O, within the same job or across different jobs. Furthermore, a POSIX interface provided from echoFS allows easy access to burst buffers available from any compute node (e.g. SSDs or NVM) for legacy applications. While the echoFS project is still ongoing research, it has demonstrated a number of attractive features that a user-level file system can achieve for ad-hoc application users.

This session continues with more talks on the implementation of user-level file systems. Tianqi Xu a doctoral student from Tokyo Institute of Technology in Japan presented his talk on **Exploring User Level Burst Buffer on Public Cloud and HPC**. To exploit burst buffers for fast I/O required by large-scale applications, Tianqi explored the concept of user-level file systems in two distinct computing environments: public clouds and HPC centers. For the first environment, Tianqi and his colleagues introduce CloudBB as a user-level and on-demand file system to leverage burst buffers in the cloud, tapping virtually unlimited resources therein. Tianqi positioned the burst buffers managed by CloudBB as a new storage tier shared by compute nodes in the cloud. These burst buffers, in combination with the shared cloud storage, form an on-demand two-level hierarchical storage system managed by CloudBB. Together, this new user-level file system enables scalable I/O with multiple metadata servers, support fault resilience through file replication and recovery techniques. Besides its support to burst buffers in the cloud, Tianqi extends CloudBB with additional capabilities to exploit burst buffers for applications in the HPC centers. His work is then manifested into a new implementation called HuronFS. By building dedicated burst buffer on-demand, HuronFS can accelerate applications and avoid I/O contention by leveraging low-latency and high-bandwidth interconnects (InfiniBand) commonly available from HPC centers. It has been shown to work very efficiently on Tokyo Tech's supercomputer, TSUBAME2.5.

The potential of user-level file systems is further demonstrated with two file system implementations MarFS and DeltaFS developed by a team from the Los Alamos National Laboratory in the U.S. Further in this session, Brad Settlemyer from that team covered these two file systems with his talk on **MarFS and DeltaFS at LANL: User-Level File Systems Opportunities and Challenges**. Aiming for great simplicity, MarFS leverages a tool called PFTool that provides parallel file transfer agents (FTAs) to distribute storage objects across a scalable long-term storage system. All I/O operations have to go through the FTA front engine, where the small files will be packed and large files will be sharded. The resulting MarFS is capable of storing extremely large scientific data sets with really scalable metadata performance. Brad argues that MarFS is able to combine a simple POSIX consistency model while enable simple capacity addition over time. Furthermore, Brad's team has designed another user-level file system called DeltaFS that decouples application metadata load from the underlying HPC platform storage system, thereby enabling an extremely scalable metadata plane for VPIC (Vector Particle in Cell) applications. Brad believes that both systems allow LANL to procure storage systems with great flexibility while supporting ultrascale scientific simulation workloads.

Besides enabling new file systems for burst buffers, the power of user-level file system has also been employed for existing file systems to leverage burst buffers. Osamu Tatebe from the University of Tsukuba in Japan presents a case study along this line through his talk on **Gfarm file system meets burst buffers**. Like Google FS and HDFS, Osamu and his team have developed their own high-performance file system called Gfarm to exploit local access performance and avoid network data transfer. His team sees a new opportunity to leverage node local NVMe SSD and NVRAM. They contend that the importance of

durability constraints can be downgraded while striving for storage performance, especially for temporary data. To this end, Osamu has modified the conventional I/O calls such as pread and pwrite to leverage fast RDMA operations and dynamic memory registrations, while ditching the durability for temporary data. The experimental results show that this extension can enable effective integration of burst buffers in Gfarm.

Finally, Prof. Hermann Härtig from TU Dresden in Germany presented his talk on **HPC File System Opportunities with Microkernels (LWK), especially L4-based**. Hermann argued that micro kernels have a lot to offer for the construction of user-level file systems. With a review of HPC operating system research, he pointed out that a variant of systems has become popular again after a period of hibernation, e.g., by combining a micro kernel with monolithic OS while splitting the application functions across the two. He then contends that a number of important but interesting questions ought to be addressed for the resulting software ecosystem to function efficiently. A lot of these questions rest with how the system calls can be implemented across the split between the micro kernel and OS. There are ample opportunities for OS experts as well as researchers on user-level file system implementations.

5.1 Direct-FUSE: A User-level File System with Multiple Backends

Yue Zhu (Florida State University – Tallahassee, US)

Although the FUSE file system alleviates the difficulty of developing a kernel-based file system, it introduced significant amount of additional overheads for I/O function calls. Some file systems are leveraged as libraries to avoid the involvement of FUSE kernel module. However, they may meet problems when dealing with multiple backends and distinct file paths and file descriptors for different backends. We propose Direct-FUSE to provide multiple backend services for one application without going through the FUSE kernel.

5.2 echofs: Enabling Transparent Access to Node-local NVM Burst Buffers for Legacy Applications

Alberto Miranda (Barcelona Supercomputing Center, ES)

The current growth in data-intensive scientific applications poses strong demands on the HPC storage subsystem, since data needs to be typically copied from compute nodes to I/O nodes and vice versa when calculations start and stop. In this scenario, the emerging trend of adding denser, NVM-based storage to compute nodes as burst buffers, offers the possibility of using these resources to construct temporary filesystems that perform ad-hoc I/O optimizations for specific batch jobs. Thus, we introduce the concept of a temporary filesystem called echofs that coordinates with the job scheduler to preload a job's data dependencies into a collaborative burst buffer, which is created by virtually aggregating the NVM burst buffers available to compute nodes. The filesystem distributes data across nodes with a pseudo-randomized striping strategy to favor load balance, offer aggregated I/O and

allow for efficient data redistributions when the number of compute nodes change between jobs. In addition, echofs offers a POSIX interface and internally manages the access to the different burst buffers available to a compute node (e.g. SSDs or NVM), so that researchers and legacy applications can interact with the filesystem without significant issues.

5.3 Exploring User Level Burst Buffer on Public Cloud and HPC

Tianqi Xu (Tokyo Institute of Technology, JP)

As the big data grows, more and more computational resources are required to run large scale applications, However, the strict access in HPC centers prevents public users from running their applications at large scale. In this talk, we introduce CloudBB, a user-level and on-demand filesystem to exploit burst buffer techniques for clouds. With CloudBB, large scale applications can be executed on public cloud instead of HPC for easier access, and virtually unlimited resource. Unlike conventional filesystems, CloudBB creates an on-demand two-level hierarchical storage system and caches popular files to accelerate I/O performance. CloudBB enables scalable I/O with multiple metadata servers, and also is resilient to failures by using file replication, failure detection and recovery techniques. On the other hand, in recent HPC systems, the computational performance has been rapidly increasing. However, the performance of parallel file systems in HPC has not been able to keep up with computing power. This limits the performance of scientific data-intensive applications in HPC. In addition, parallel file systems are shared by all the users, which exacerbates the situation due to I/O contention. We also introduce HuronFS, an extension of CloudBB to solve the problems in HPC. By building dedicated burst buffer on-demand, we can accelerate applications and avoid I/O contention. HuronFS exploits low-latency and high-bandwidth interconnects (InfiniBand) for HPC systems. Our evaluation shows HuronFS can achieve up to 3.5 GB/sec per a single I/O node in I/O throughput, which is comparable to the average throughput of the parallel file systems in Tokyo Tech's supercomputer, TSUBAME2.5.

5.4 MarFS and DeltaFS at LANL: User-Level File Systems Opportunities and Challenges

Brad Settlemyer (Los Alamos National Laboratory, US)

In this talk I discuss MarFS and DeltaFS, two user-level file systems efforts at Los Alamos National Laboratory that provide greater flexibility in how larg high performance computing (HPC) data centers procure storage systems. MarFS leverages a POSIX namespace and distributed object storage technology to construct a scalable long-term storage system that is capable of storing extremely large scientific data sets. The architecture enables a subset of the POSIX consistency model relevant to long term storage use cases while also supporting simple capacity addition over time. DeltaFS provides an extremely scalable metadata plane that decouples application metadata load from the underlying HPC platform storage system. Both systems allow LANL to procure storage systems with great flexibility while supporting ultrascale scientific simulation workloads.

5.5 Gfarm file system meets burst buffer

Osamu Tatebe (University of Tsukuba, JP)

File system for data processing like Google FS, HDFS and Gfarm, exploits local access performance and avoids network data transfer. This is especially relevant now with node local NVMe SSD and NVRAM. But there are some changes too, more performance required, durabilitly not as important in some modes, and data may be temporary. This talk designs Gfarm for burst buffers to implement RDMA access and no durability option.

5.6 HPC File System Opportunities with Microkernels (LWK), especially L4-based

Hermann Härtig (TU Dresden, DE)

I have tried to point out to the file system experts present what micro kernels can do for the construction of user-level file systems. In HPC-operating systems research, a variant of systems has become popular again after a period of hibernation: a micro kernel(like L4) and a monolithic OS (like Linux) are combined and applications are split in two parts one running on the micro kernel and the other on Linux. A number of interesting questions then arises such as: who is the boss (L4 or Linux), what is the programming model for the micro kernel, how to simplify the programming of the split application. I described how in a combination of L4 and Linux, some of the questions are answered: 1) in contrast to competitors L4 has a message-passing/memory-management oriented interface, much better suited for user-level memory-management than the common a Posix-like interface, 2) L4 is in control over Linux, which help isolate critical applications from irregularities of Linux, 3) Linux applications can simply leave the sphere of control of Linux, freely float on the available cores, and be pushed back if Linux SysCalls are needed. I left it the FS experts present to decide how to use that opportunities.

6 Talks Session: Object Stores and Alternatives

The fourth session of the Dagstuhl seminar focused on research and development of object stores as an alternative self-describing storage solution to popular but wieldy parallel file systems. This session features four separated talks that covered the employment of object stores for distinctly different use cases.

Suren Byna from Lawrence Berkeley National Laboratory in the U.S. started the session with his talk on **Proactive Data Containers (PDC) for next generation HPC storage**. This work is from an ASCR research project funded by the U.S. Department of Energy to explore next generation storage systems and interfaces. In view of the hefty impedance imposed by the deep storage hierarchy to vertical data movements, Suren and his colleagues introduced PDC as the new storage paradigm that offers an object-centric data abstraction for storage management on next-generation exascale systems. PDC is particularly keen on

addressing the fundamental performance challenges caused by the consistency requirements and stateful semantics of the POSIX I/O calls. To this end, the PDC team has introduced a number of novel features as part of the PDC storage architecture, including a simple client API for light-weight object manipulation, SoMeta for scalable metadata management and a data elevator for automatic and transparent data movement across storage hierarchies. All these features are implemented as part of the HDF5 I/O model for scientific datasets.

Andreas Dilger from the Intel High Performance Data Division presented the second talk on **DAOS: A Scale-Out Object Store for NVRAM and NVMe**. The Distributed Asynchronous Object Storage (DAOS) project is sponsored by the U.S. Department of Energy. Different from the PDC effort, the DAOS project is targeted as a general storage stack to exploit low-latency byte-granular NVRAM and NVMe storage for any upper-level storage paradigms such as MPI-IO, ADIOS and even PDC. DAOS leverages proven technologies for its communication protocol and fine-grained lockless client I/O. Similarly to PDC, it also adopts the container abstraction as an integral part of its five-tier storage model for a novel object store. These tiers simplify the formation of automatic storage workflows for applications and enable transparent encapsulation of a wide variety of different storage interfaces. As Andreas indicated, the overarching goal of the project is to overcome POSIX limitations and offer a unified storage model over which domain-specific data models can be developed, such as HDF5, ADIOS, HDFS and Spark, in the meantime, increasing data velocity by several orders of magnitude. The project is in active community development. The current code has been released open-source on GitHub (https://github.com/daosstack/daos).

Maria S. Perez from the Universidad Politëcnica de Madrid and Luc Bougë from INRIA of France jointly gave the third talk on **Are objects the right level of abstraction to enable the convergence between HPC and Big Data at storage level?**. Given the dual demands from HPC and data analytics applications, Maria and her team have developed an object based storage system called Tyr that aims to provide a converged storage architecture. Maria indicated that Tyr is designed with scalable concurrency and synchronization management. Using BLOBs (Binary Large OBjects) as the basic building blocks, Tyr enables the store and retrieval of unstructured data through simple object-based binary methods. Tyr internally maps all writes to transactions. Its transactions are not limited to a chunk for a file, but may span multiple chunks and blocks. The WARP algorithm is employed for high-performance, sequentially-consistent transaction management. Maria and her team have evaluated Tyr using the MonALISA data collection and comparing with other object-based storage systems including Ceph, Azure and BlobSeer. Her results showed that Tyr delivers good performance advantages for transactional writes and reads. Furthermore, Maria believes that Tyr is well suited for both HPC and Big Data Analytics applications, for which a number of nice figures with comparative results are also included in her talk.

Finally, Lance Evans and Raghunath Raja Chandrasekar from Cray Inc jointly gave a talk on **An Exploration into Object Storage**. In this talk, they presented their exploration of a converged storage architecture for both HPC simulation and analytics platforms. Cray's position is that a converged storage platform shall emerge through hardware and software co-design of exascale-class supercomputers and analytics frameworks. A solution they are currently exploring is a library called SAROJA (Scalable And Resilient ObJect storAge). In its software architecture, SAROJA consists of three main components. The first is its user-space API library designed a host of intelligent mechanisms including algorithmic selection of data nodes, automatic translation of posix commands to KV/NoSQL operations. The other two components are the meatadata service and data path services. Lance and Raja designed

these two critical pieces in a decoupled manner to allow flexible evolution. Particularly, Lance and Raja explored the use of Casandra NoSQL operations for the metadata services, and a number of Ceph components for data path services. They claim to have seen promising results from the prototype, which warrants future development and plenty of opportunity for advancement.

6.1 Proactive Data Containers (PDC) for next generation HPC storage

Suren Byna (Lawrence Berkeley National Laboratory, US)

Emerging high performance computing (HPC) systems are expected to be deployed with an unprecedented level of complexity, due to a multi-layer memory/storage hierarchy. This hierarchy is expected to range from CPU cache through several levels of volatile memory to non-volatile memory, traditional hard disks, and tape. Simple and efficient methods of data management and movement through this hierarchy is critical for scientific applications using exascale systems. Existing storage system and I/O technologies face severe challenges in dealing with these requirements. POSIX and MPI I/O standards that are the basis for existing I/O libraries and parallel file systems present fundamental challenges in the areas of scalable metadata operations, semantics-based data movement performance tuning, asynchronous operation, and support for scalable consistency of distributed operations.

Moving toward new paradigms for next-generation storage in the extreme-scale era, we have started investigating novel object-centric data abstractions and storage mechanisms that take advantage of the deep storage hierarchy and enable proactive automated performance tuning. We are developing a fundamentally new data abstraction, called Proactive Data Containers (PDC). A PDC is a container within a level of storage (memory, NVRAM, disk, etc.) that stores science data in an object-centric manner. In this talk, I will present an evolution of parallel I/O paradigms, high-level libraries such as HDF5, and our recent progress in PDC system development, PDC application programming interface (API), scalable metadata management and data movement optimizations aimed for object-centric storage.

6.2 DAOS: A Scale-Out Object Store for NVRAM and NVMe

Andreas Dilger

The Distributed Asynchronous Object Storage (DAOS) is an open-source (https://github.com/daos-stack/daos) storage stack designed from the ground up to exploit low-latency byte-granular NVRAM and NVMe storage. DAOS provides true byte-granular data and metadata I/O using persistent memory, combined with NVMe storage for bulk data. This, combined with end-to-end OS bypass, results in ultra-low I/O latency. The DAOS stack aims at increasing data velocity by several orders of magnitude over conventional storage stacks, while providing extreme scalability and resilience.

The DAOS API abstracts the underlying multi-tier storage architecture and offers a unified storage model over which domain-specific data models can be developed, such as HDF5, ADIOS, HDFS and Spark. Legacy POSIX access is similarly built on top of the DAOS

API. The core data model is a byte-granular key-value store which allows I/O middleware to overcome POSIX limitations and to have access to advanced capabilities like non-blocking I/O, ad-hoc concurrency control, distributed snapshots, native producer-consumer pipeline, end-to-end data integrity, index/query and in-situ analysis. DAOS also provides scalable, distributed transactions to I/O middleware with improved data consistency and automated recovery.

6.3 Are objects the right level of abstraction to enable the convergence between HPC and Big Data at storage level?

Maria S. Perez (Universidad Politëcnica de Madrid) and Luc Bougë (INRIA – Rennes, FR)

One of the most challenging aspects of storage in HPC is addressing concurrent write sharing. This scenario appears in applications requiring ACID transaction support or having situations in which concurrent updates may lead to incoherent results. Many HPC storage systems rely on locking to coordinate the access when concurrent writes are needed. This locking mechanism introduces a non negligible performance overhead.

On the other hand, storage systems are converging towards the use of a distributed object storage model, as a way of providing scalability and more flexibility. BLOBs (Binary Large OBjects) appear in this arena as a very good alternative. They allow storing unstructured data accessed through low-level binary methods.

However, the scalability and high performance of BLOBS are achieved at the expense of weak consistency. Tyr is a new BLOB storage architecture, which provides efficient multichunk and multiblob transactions under heavy access concurrency, but guaranteeing sequential consistency. A significant advantage of this approach is that it enables the building of lightweight file systems on top of this layer. Moreover, due to its features, Tyr can be used as underlying storage system for both HPC and Big Data Analytics applications, contributing to the desired convergence between these two worlds.

6.4 An Exploration into Object Storage

Lance Evans (Cray Inc. – Seattle, US) and Raghunath Raja Chandrasekar (Cray Inc. – Seattle, US)

The need for scalable, resilient, high performance storage is greater now than ever, in high performance computing. This talk presents exploratory research at Cray that studies aspects of emerging storage hardware and software design for exascale-class supercomputers, analytics frameworks, and commodity clusters. Our outlook toward object storage and scalable database technologies is improving as trends, opportunities, and challenges of transitioning to them also evolve. Cray's prototype SAROJA (Scalable And Resilient ObJect storAge) library is presented as one example of our exploration, highlighting design principles guided by the I/O semantics of HPC codes and the characteristics of up-and-coming storage media. SAROJA is extensible I/O middleware that has been designed ground-up with object

semantics exposed via APIs to applications, while supporting a variety of pluggable file and object back-ends. It decouples the metadata and data paths, allowing for independent implementation, management, and scaling of each. Initial functional and performance evaluations indicate there is both promise and plenty of opportunity for advancement.

7 Talks Session: File Systems Building Blocks

The development process of a (parallel) file system typically takes ten years, before a file system matures. User-space file systems should, according to the previous sessions, be able to address specific application demands, which requires a short design cycle.

Shorter times to mature can partly be achieved by moving the file system functionality from kernel space to user space and therefore enable better debugging capabilities as well as access to libraries with predefined functionalities. One aspect of this session is therefore the necessary library support, which can be used in many user-level file system implementations. A second aspect to reduce the development time is to make file systems *simpler* by removing functionality, which is not required for *all* applications. The simplifications can include waiving POSIX guarantees, but also to generally relax security via capabilities or ACLs (if it can be guaranteed by system architecture).

The talk **Security Issues in User-Level File Systems** from Stergios Anastasiadis focused on the current state of the art on how to provide security guarantees within our user-space file system scenario. He started with a general overview on datacenter multitenancy, where virtual machines access disk images rather than files in an Infrastructure as a Service. The traditional ways of federated access control in IaaS include centralized, Peer-to-peer, or Mapping-based (e.g., HekaFS) approaches. Sharing of co-located data resources is typically inflexible due to the lack of appropriate sharing architecture (with Amazon Elastic File System being a notable exception, while still relying on NFS that limits scalability and security).

The Dike-approach, developed in Stergio's group, provides mechanisms for hierarchical identification and authentication, where the provider manages tenants and the tenants manage users. The native multi-tenant authorization separate ACLs per tenant and provider and therefore allows for an efficient permission management, including shared common permissions and inheritance. The Dike prototype has been implemented over Ceph.

Security for user-space files systems has to include, according to Stergios, standard aspects like confidentiality, integrity, and availability, flexibility concerning the policies, efficiency, as well as support for local and remote devices as well as for byte-addressable *and* block-based access. Efficient access to SCM includes that the policy mechanisms cannot rely on kernel mediation, as the switch to the kernel space already removes the latency advantages of SCM. Proposed approaches in literature to include security in untrusted environments include Secure containers (SCONE, OSDI 2016), limited kernel mediation via Arrakis (OSDI 2014), file systems for SCM (Aerie, EuroSys 2014), and remote direct storage access (DiDAFS, HotOS 2015).

SCONE secure containers are based on Intel's Software Guard Extension SGX that provides secure regions of user-mode address spaces as enclaves, which are associated with an enclave page cache, where data from the enclave page cache is encrypted and integrity-protected when sent to memory. SCONE provides secure containers by protecting the confidentiality and integrity of application data even against an adversary with super-user access to the operating system and hardware. All internal data is protected through SGX,

while SCONE provides transparent encryption and authentication of data via shields, e.g., to send system calls to the host OS through shared memory or for writing to the file system. SCONE distinguishes between unprotected, authenticated, encrypted and authenticated files, while an ephemeral file system maintains state in non-enclave memory.

The main idea of Arrakis is to split the operating system tasks in two different entities. The kernel operates in the control plane and has to configure the hardware to limit application misbehavior. Applications in Arrakis can then independently and directly access virtual I/O devices and allows most I/O operations to be performed without prior kernel access. Arrakis implements a hardware-independent layer for virtualized I/O, including a virtual interface controller, doorbells, filters, and rate specifier.

Aerie proposes a flexible file system, where user-mode programs can directly access storage-class memory without kernel interaction. Aerie is based on a decentralized architecture. lifFS is an untrusted library providing a file-system interface including naming and data access via persistent memory primitives, which is linked to the user space application. A centralized trusted file-system service enforces metadata integrity and synchronization by providing distributed lock service and the SCM Manager is a kernel component for SCM allocation, mapping and protection through hardware privileges by providing memory permissions and virtual address mapping.

DiDAFS provides remote allocation mechanisms of main memory or storage device areas to support remote direct memory (and storage) access. The control plane of the device-owning node therefore exposes possible allocations to the network cards, which then can initiate RDMA accesses to remote user virtual memory. A user-level file system is used to manage metadata, while the metadata negotiation can still be handled through standard RPC mechanisms. Data, which is not cache-resident at the data accessing node, can therefore be mapped and efficiently fetched.

Stergios's overview has shown that security for user-space file systems still includes open issues when coupling the proposed ideas concerning protection (limited mediation, virtualization), consistency (processor cache, local/remote caches), interfaces (POSIX, memory-mapped, key-value, high-level data structures), and communication (library, RPC, RDMA, software fault isolation).

The motivation for Federico Padua's talk **An effort to systematically understand UFS design requirements** has been to understand the important design points when starting the development of a user-space file system and which file system components need to be implemented and which components can be thrown away. Furthermore he wants to understand whether tools can help to characterize application and link their characteristics to suggestions for FS component design.

Key file system components are data and metadata management, file system caches, fault tolerance, concurrency control and consistency as well as permissions and related security mechanisms. Especially concurrency control, consistency, and security seem (according to Federico) to be barely touched in file system research as long as a file system is not intended to be practically used.

Federico's talk is then mostly focused on file locking and further concurrency control mechanisms, as today's parallel file systems Lustre and GPFS (as POSIX files systems) implement quite complex distributed byte-range locking protocols to safely access data. Lustre and GPFS employ a pessimistic approach to concurrency control where a conflict is always expected, so that the file systems lock all accesses. OrangeFS on the contrary uses an optimistic approach, where a conflict is not expected and is only dealt with when it (rarely)

occurs. POSIX.1-2008 on the other side does not specify the behavior of concurrent writes to a file from multiple processes, as applications should use some form of internal concurrency control.

The POSIX approach poses the question, whether file locking is that important in HPC at the file system level and whether concurrent accesses should better be correctly handled in applications or libraries (and whether they do it). Serialization (and therefore locking) might become necessary in case of overlapping writes to regions of a file or even to the same byte, in cases of clients reading after data has been written before, so that the client reads the last update. Furthermore, locking can happen in case of false sharing, where two processes from different nodes try to write to different bytes in a file mapping to the same block, or in case of writing to internal file system data structures.

Federico then presented his current work to measure file locking activity to determine the concurrency control mechanism needed in a file system. Starting from a pool of 12 applications from bioinformatics, chemistry, soft matter physics, high-energy physics, and weather modeling he traces in a first step the locking activity (using GPFS tracing) before characterizing applications to understand whether locking has been necessary.

Michael Kuhn's talk **The Case for a Flexible HPC Storage Framework** is motivated by the observation that it is hard to investigate new file system approaches, as file systems are typically monolithic in design and researchers have to change many different components of a file system to investigate a new idea. This leads, according to Michael, to two major problems. First, many specialized solutions do exist for a particular problem, which are seldom contributed back and second, it is necessary to have a complete understanding of a file system before a researcher can contribute to file system research, which is an unnecessary hurdle for young researchers and students.

An additional trend is that many applications rely on high-level I/O-libraries like NetCDF or HDF5, which support the exchange of data based on their self-describing properties. Tightly coupling these interfaces with current file systems is hard to achieve and is most often being performed in the context of big projects like DAOS or ESiWACE. Similar trends can be seen in the context of HPC and big data convergence or of alternative file system interfaces.

The main focus of Michael's talk has been on the design of JULEA, which is a framework to support rapid prototyping of new ideas via plugins for interfaces, storage backends, or new semantics. JULEA should help to overcome the burden that many projects have to re-implement basic functionality from scratch before being able to focus in core innovations. JULEA runs completely in user space and supports plugins that are configurable at runtime. High-level libraries and applications can use it directly, so that it provides a convenient framework for research and teaching.

JULEA therefore makes it possible to offer arbitrary interfaces to applications, both traditional file system interfaces and completely new ones. The possibility to change client implementations or the storage backend should foster experimentation, while it is even possible to dynamically adapt the semantics, e.g., between POSIX and MPI-IO on a per-operation basis. JULEA therefore does not put any restrictions on the client interface and its user-space implementation removes many restrictions of the standard kernel-space VFS architecture. This is, according to Michael, useful for applications and I/O libraries, as libraries like HDF5 can be placed directly on top of JULEA.

The server backends are separated into data and metadata backends. Additionally, client and server backends. Data backends manage objects and their design is influenced by file systems (Lustre and OrangeFS), object stores (Ceph's RADOS), and I/O interfaces (MPI-IO).

Metadata backends manage key-value pairs and are influenced by databases (SQLite and MongoDB) and key-value solutions (LevelDB and LMDB).

An aim of JULEA is it to make file system adapt to applications instead of the other way around. The semantics concerning atomicity, concurrency, consistency, ordering, persistency, and safety of operations can be changed at runtime and it is possible to mix the settings for each of these categories, even if not all combinations might produce reasonable results.

JULEA has been implemented in C11 as an open source project, which can be accessed at https://github.com/wr-hamburg/julea. It integrated support for tracing and unit tests. Future improvements will include an HDF5 VOL plugin, a data backend for Ceph's RADOS and a metadata backend for LMDB.

Phil Carns talk **Why should we still talk about data service building blocks?** is also discussing ways to simplify file system design, but takes a slightly different approach than Michael's recommendations. He started with a reflection about the origins of conventional storage service architectures, which have been based on the key technologies of block devices, sockets, pthreads, and kernel drivers. There have only been few cores per node and the concurrency has been further reduced by introducing I/O forwarding nodes. Data has been served via dedicated, remote service nodes, which have been either connected through TCP/IP sockets or in HPC over dedicated high-speed networks. The costs to tie the different components have not mattered too much, as the underlying magnetic disks had service times in the order of milliseconds.

An updated version of the same storage service architecture shows some key changes. New technologies like NVRAM, RDMA, dynamic services, and a much higher concurrency have been putting much higher pressure on keeping latencies and jitter low. Technology transitions include the shift from block devices to byte-addressable memory, moving from sockets to small messages and RDMA as well as the introduction of dynamic service groups and cluster-local services. Phil's main message is to encourage researchers to not implement everything again and again but to share engineering components and to free up time for research.

Phil therefore presented an example data path which is based on open source components like Argobots, Margo, Mercury, CGI, and Libpmem. The resulting architectural diagram looks like including many layers, but Phil argues that the number of components is not a major challenge as long as the interactions between the layers are efficient. The keys to optimize interaction between layers is to avoid privileged mode transitions, context switches in general, and memory copies.

The formerly mentioned Libpmem is a user space library written by Intel to access persistent memory regions (NVML). It provides well-defined control over persistency and device naming/reference conventions. A family of derived libraries for data structures that understand persistent memory references, transactions, or atomicity have been developed, like Libpmemobj for object storage, Libpmemblk accessing fixed-size blocks, Libvmmalloc, which replaces the standard `malloc()` command, or Pmemfile as a file system in user space with no kernel VFS mediation.

Mercury is an RPC system for use in the development of high performance system services, which has been developed by the HDF Group and ANL. It is portable across systems and network technologies and provides efficient bulk data movement to complement control messages. It has been built on lessons learned from, e.g., IOFSL, Nessie, or lnet (see https://mercury-hpc.github.io). It sits on top of transport libraries using a plugin API for network abstractions and supports OFI, IB, TCP/IP, and SHMEM transport protocols. Mercury provides simplifications for service implementers like remote procedure calls, RDMA

abstraction, protocol encoding, or clearly defined progress and event models without requiring a global fault domain like `MPI_COMM_WORLD`. Phil mentioned that Mercury is minimal and fast, but a challenge to program. Therefore his group has added a layer We added a layer called *Margo* that adds an easy-to-use sequential interface, which relies on user-level thread scheduling for concurrency.

An earlier observation in Phil's talk has been that there is a much higher I/O concurrency than there are cores available on storage service nodes. Each of the requests can be illustrated as a state machine, where each request involves multiple steps: some will block, some will branch and the server needs to keep track of where each request is in state machine. ANL has therefore introduced ARGOBOTS as a user-level threading framework with lightweight context switching among many concurrent threads. Key features for data services are that it enables developers to track the state of many concurrent operations with simple service code paths and low OS resource consumption. ARGOBOTS also allows custom schedulers to implement priorities or limit CPU usage and provides primitives that facilitate linkage to external resources.

Phil closed his talk with an example of how to glue multiple components together when providing a huge set of microservices. Composed libraries can, e.g., exchange RPCs with a composed service, where the composed service delegates RPCs to microservices and propagates RDMA tokens to the service(s) that will drive data transfer. Generally, this approach allows to mix and match remote and non-remote components with the same API conventions, while not changing the service implementation. In a last step, services can be combined in scalable service groups (SSGs) (see https://xgitlab.cels.anl.gov/sds/ssg), which add group membership to Mercury by bootstrapping groups, providing identifiers to concisely reference groups of processes, and providing optional fault detection.

The philosophical question whether **To FS or not to FS...** has been asked by Rob Ross. He experienced that user level approaches are typically employed as a light weight and flexible way to provide functionality that does not have a viable business model driving vendor support. Functionality is therefore limited to just what is needed to accomplish the task at hand. However, the potential for such user level services enables new, fundamental services that might become keystones of future HPC systems. Main questions are how do the particular I/O use cases inform the way we manage data, how we should present hierarchical storage systems to user applications and how should we manage data movement through a storage hierarchy.

The specialization of data services can already be seen analyzing executables and libraries like SPINDLE, checkpoint-restart tools like SCR and FTI, as well as intermediate data products like DataSpaces, Kelpie, or MDHIM, which all are not filesystems. Many of them are provisioned via MPI or the batch system, use local storage, but only provide minimum fault management and security.

Rob argues in favor of Phil's approach when demanding for an ecosystem of data services, where many components can be shared across multiple services. Some of these services will look like file systems, while others take a completely different form. To build such an ecosystem, it is important to tackle hard problems like group membership, authentication and authorization, publish/subscribe **and** performance. These basic services will, according to Rob, enable a broader community to build better, more capable user-level data services than possible today.

The idea behind such an ecosystem is presented based on the example of future applications exploring the use of multi-scale modeling. The application Lulesh, e.g., loosely couples continuum scale models with more realistic constitutive/response properties. Fine scale

model results can be cached and new values interpolated from similar prior model calculations. Goals of a simulation run are to minimize costly fine scale model executions, query/response times to databases and to load balance accesses to the distributed database. A possible approach is to start with a key/value store, to distribute approx. nearest-neighbor queries, to distribute data to co-locate values for interpolation and to import/export results to persistent stores. All these steps do not have to include file systems.

A closing remark from Rob included the observation that many parallel file systems today are veneers on object stores. While users consistently complain about file systems that they cannot create as many files (names) as they want and are not happy with metadata management, these are not the problem of the underlying object stores. Some users have therefore moved to structured APIs like HDF5, ADIOS, or netCDF and many classes of data are used where the file system is not visible to the user at all. Interestingly, object stores themselves are not controversial and probably the best bet for the organization of long(er) term storage.

7.1 Security Issues in User-Level File Systems

Stergios V. Anastasiadis (University of Ioannina, GR)

We provide a brief overview of our recent research activities on reducing the interference from compactions in sorted key-value data stores, improving the performance of object-based distributed filesystems with host-side journaling, and adding native multitenancy support to object-based distributed filesystems. Then we summarize recent design points on secure user-level filesystems and identify possible open issues for further research on isolation, caching, interface and communication.

7.2 An effort to systematically understand UFS design requirements

Federico Padua (Universität Mainz, DE)

File locking is a form of serialization used by parallel filesystems to prevent data corruption or other bugs in case many programs or many processes access the same files at the same time. This is the case for a single MPI program, in which many processes read and write a single shared file or multiple files during the run. If user-level filesystems aim to replace some key functionalities of kernel level filesystems, then it becomes important to verify whether such a filesystem needs to support any form of concurrency control or none. In case a form of concurrency control is needed, it's also important to asses which one would be the best option. Our interest in studying concurrent file accesses and file locking is motivated by one section of the POSIX specification: "This volume of POSIX.1-2008 does not specify behavior of concurrent writes to a file from multiple processes. Applications should use some form of concurrency control.". It then becomes fundamental that filesystem designers take into account this possible issue when developing their filesystems: either some form of

concurrency control is needed or not. This mandates a clear understanding and study of real world applications that run in HPC centers.

We hope that the study we are embarking on will open up a discussion and a rethinking of some assumptions. For instance, previous works tried to explore the benefits of optimistic concurrency control in the HPC context. We plan first to assess to what extent ad-hoc file system developers should care about concurrency control or not, to support one design choice over another: this can justify optimistic or pessimistic approaches.

In the case of more general purpose filesystems in HPC clusters, such as GPFS and Lustre, the default to concurrency control is to employ a pessimistic approach using some kind of distributed locking (with different level of granularity). We plan to use the study to provide a solid experimental foundation to see whether this pessimistic approach is needed even in the case of general purpose filesystems to correctly support HPC applications.

7.3 The Case for a Flexible HPC Storage Framework

Michael Kuhn (Universität Hamburg, DE)

JULEA is a flexible storage framework that allows the offering of arbitrary client interfaces to applications. To be able to rapidly prototype new approaches, it offers data and metadata backends that can either be client-side or server-side; backends for popular storage technologies such as POSIX, LevelDB and MongoDB have already been implemented. Additionally, JULEA allows dynamically adapting the I/O operations' semantics and can thus be adjusted to different use-cases. It runs completely in user space, which eases development and debugging. Its goal is to provide a solid foundation for storage research and teaching.

7.4 Building blocks for user-level HPC storage services

Philip Carns (Argonne National Laboratory, US)

Today's most well-known distributed storage systems were envisioned and developed in an era dominated by Linux clusters and magnetic disks, and the technology building blocks that went into those storage systems were well understood: TCP/IP, local files or block devices, POSIX threads, and OS kernel drivers. New HPC system trends are putting pressure on limitations of those building blocks, however. Storage device latency is dropping, CPU clock speed advances are slowing, and emerging applications demand flexibility in interfaces and provisioning. If our building blocks don't evolve to reflect these trends, then system software overheads will soon become the number one limiting factor in storage system efficiency.

In this talk I will discuss how technology building block assumptions have changed in recent years and the challenges and opportunities that have arisen as a result. In particular I would like to highlight how RDMA-enabled network layers, persistent memory interfaces, lightweight threads, and user-level interfaces are taking the place of TCP/IP, files and blocks, POSIX threads, and kernel modules in HPC storage system design.

7.5 To FS or not to FS...

Robert B. Ross (Argonne National Laboratory, US)

In the HPC community, user level approaches are typically employed as a light weight and flexible way to provide functionality that caters to specific applications or that doesn't have a viable business model driving vendor support. In this scenario, functionality is limited to just what is needed to accomplish the task at hand. However, the potential for user level services goes far beyond the examples that we see today, potentially providing new, fundamental services that are keystones of future HPC systems. In this talk I will discuss some aspects of distributed systems that HPC user level service developers often avoid, highlighting some areas for collaboration on (or leveraging of) components and micro services that would open up new opportunities for user level services in HPC environments.

8 Discussions

8.1 Discussion: How are stage in and stage out operations actually going to work? *Lead: Ned Bass*

On upcoming systems, we assume we will be able to move data to and from the tier of storage on or near the compute nodes before and after a job in stage-in and stage-out operations. We assume that the operations will improve performance, because jobs will not have to wait for the potentially expensive data transfer operations before they begin or after they end. Given all that, it is unclear how these operations will work in realistic situations. In this session we dove into questions surrounding the practical implementation and implications of stage-in and stage-out operations.

Are these stage-in and -out operations part of the job or do they belong to the resource manager?

- We will have to pay the cost somehow. How do we save what we need and minimize the impact of data movement?
- We need a way for users to define, specify what they want for particular data files. What is the information that the resource manager needs? What is the API?
- Are these operations performed during the run? Or pre-/post-run? What happens if pre-stage is supposed to happen before the run but the operation is not complete and the nodes are ready for the job? Should the job be launched anyway?
- What are the security concerns when someone else's data is on the storage but someone else is running?
- Are you consuming network bandwidth that could impact other jobs with stage-in/out? Is the impact on other jobs big enough to offset the wasted time due to waiting for files to transfer (idle compute nodes)

What does this mean from a scheduler perspective?

- It is complex from a resource scheduling perspective.
- Not only is it scheduling resources for data movement (network, capacity), but it is also trying to schedule nodes taking into account additional resources.

- Scheduling jobs is much harder with node-local storage case. Multi-constraint scheduling problem
- Trade-off between the time to wait for jobs to run while data is fetched and the time to read data from the parallel file system. Which is worse?

Are the questions about this different for shared burst buffers versus local burst buffers?

- Does stage-in/out affect jobs with local burst buffers more or less than with shared burst buffers?
- It's still a problem of shared network resources, storage resources
- Perturbation is unavoidable in either case
- Does this mean we need to schedule I/O? E.g. Delay checkpoint transfer to prestage data for the next job, then trickle down checkpoint? Is scheduling too big of a hammer here? What about prioritization instead?
- Could intermix jobs that need I/O with those that don't when possible.

How can a center figure out its priorities and policies for stage-in and out?

- A multi-objective optimization problem between utilization, job interference, performance
- Could we have a dedicated storage network? That would be awesome but expensive

8.2 Discussion: How can we fairly judge the performance of storage systems – IO 500? *Lead: John Bent*

This discussion centered around how to define a set of tests for evaluating storage systems fairly. A good number of participants had already been working towards this with the IO 500, so the discussion was in the context of the IO 500. Evaluating storage systems is more challenging now that we have hierarchical storage systems and a wider variety of important I/O workloads. The IO 500 working group has already been working to define a set of tests; this discussion session introduced the tests and solicited feedback from the seminar participants.

What kinds of tests should we use?

- Easy test: What is the best you can do with your system? Danger of this is that people will just submit their results for the fastest tier of storage, e.g. burst buffers.
- Hard tests – comparisons across systems. Which one has better characteristics for different workloads?
- What is the right way to collect the measurements for a single comparable metric?
- Do we account for performance changes as systems age?
- Bandwidth tests, metadata tests
- A variety of tests for different I/O workloads including well-behaved and poorly-behaved applications. Real applications have complex behavior. We need a good set of proxy applications to capture this.
- What about a real application mix for evaluation? That would be informative for purchasing. However, real applications are very cumbersome and can take a long time. Emulators would be sufficient if they stress the I/O system in the same way.

How do we future-proof the tests?

- We don't want the defined tests to be out of date with the next generation of systems
- Want to make it independent on node size, size of operations
- Should we use relative metrics then?

What metric(s) should we use to evaluate the systems?

- Is a single metric adequate? You really want the details, number of clients, etc in some cases. Could the metric be a list of values for different layers of storage? We will want to know the performance of the layers individually as well as in aggregate
- The metric definition that will be most valuable differs given the situation at hand. A conflict between vendors and procurers
- A single metric is appealing though due to human psychology. People will want to know how they are ranked. Once they look for that, they will look at the details
- Would be good if centers could contribute their acceptance testing criteria for storage systems. Perhaps we could derive a metric from those. They could tell us what is hard/important for their workloads.

8.3 Discussion: What is a user level file system? What do we mean when we say that? *Lead: Philip Carns*

In this discussion, we set out to define what is meant when we say 'user level file system' as it became evident that we all had different definitions in our minds.

What do we mean by user level? Is it just bypassing the OS? And what does that mean?

- Do we mean no system calls to do the I/O or networking?
- No data copies necessary between user and kernel space?
- Is there a difference between user-level and user-space file system? No.
- Can a user deploy the file system or does it take an administrator?

Should we be talking about storage systems instead of file systems?

- Probably yes so we can include object stores
- Well, if we're storing data persistently then perhaps it is sufficient to call it a file system. What do we mean by persistent? Nothing lasts forever, e.g. Lustre purges. That is due to center policies, not Lustre.
- Let's settle on ULSS – user level storage systems since it seems to include everything

Why do we want user level storage systems?

- Easily deployed by user, deployed quickly
- Does not require root/administrator
- Limited lifetime file system, quick to spin up/tear down. Does not consume resources when not needed.
- Ownership of file system is user
- Developed and run at the user level, without kernel hacking
- Failure isolation
- Tailored to specific workloads, customizable. Can use different consistency models. Every metric by which you evaluate storage can be tailored: performance, durability, etc
- OS bypass: is this a good thing? A requirement? Is the only reason we want OS bypass because privilege escalation and de-escalation is high latency?
- With OS bypass we can access the raw device

What are the trade-offs for performance with user-level versus traditional file systems?

- Can add transformative, transparent improvements in performance due to customizability
- Can essentially give the user a "consistency" button, tuner. They can determine their own durability, persistence needs
- Users can choose the implementation that is best tuned to their use case, possibly violating POSIX but it is fine for their needs

8.4 Discussion: How can we characterize what users need from storage systems? *Lead: Scott Klasky*

The key question for this discussion was how can we as a community understand what users need from storage systems? If we do not understand what they need, then our efforts towards optimization will be useless. We need to have a way to characterize the I/O behavior in enough detail to get the information we need.

Can we define a set of patterns or "motifs" that describe application I/O behavior?

- If we can decide what the motifs are, then can we generate benchmarks that are reasonably representative?
- How do we decide if the motifs/patterns are accurate?
- What if they change over time? For example, we have data analytics workloads emerging now.
- What characterization tools can we use to learn the motifs? A single application may contain multiple motifs. For example, checkpoint/restart and output
- What if we define motif differently so that instead of being an I/O type it is more of an application type? The question here is do all applications of the same "type" do the same kind of I/O?

Can we build up a cache of mini apps for I/O behaviors?

- How would we generate the mini apps? From data or from an emulator like IOR or MACSio?
- What do we do about too many mini apps? Different across organizations?
- How do we capture the differences in behavior with scaling, input, architecture?
- We need to track all the I/O types from an application, the behavior of which will be use case and system dependent. For example, the frequency of checkpointing will change depending on (perceived) failure rate
- How will we validate these mini apps?
- We need to be sure to state the purpose of the mini apps. A mini app can't do everything. Need to clearly define what the mini app is intended to mimic.

8.5 Discussion: Are we ready to program to a memory hierarchy versus block devices? *Lead: Jay Lofstead*

This section documents the discussion led by Jay Lofstead surrounding the question of whether the storage and I/O community is ready for changes in hardware expected on upcoming systems, namely working with extended memory devices as opposed to block devices. In general, because this represents a potential paradigm shift for not only the storage

and I/O community but also for the programming models community, this discussion brought up more questions than it did answers.

What is memory and what is storage?

- Memory that is slow may need to be treated by programmer as storage. However, people are already doing out of core computation and treating storage as slow memory.
- If it's temporary data, then it's memory. If it's meant to persist, then it's storage.
- Conclusion seems to be that the difference is in how you use the device more than the speed of the device.

What do applications people need?

- What is familiar to applications people? Is it dealing with memory?
- Upcoming machines have burst buffers, but we will have machines that only have devices that can be addressed as memory very soon.
- Legacy software could continue to do traditional I/O, but perhaps new applications or smaller applications will change to a new programming model. Need to support both models.
- Does this question belong to the programming models folks? Not sure.
- How do we help users figure out what to store and what to recompute?

How long do we store data at each tier?

- Depends on capacity, devices near compute do not have much capacity
- Energy cost of moving data, too high compared to compute. It's much cheaper to move data across compute nodes than to move it to the parallel file system and back
- Do we want to provide hierarchical storage management software/model that users can specify high level needs of data? Can users describe the high level needs for their data in a workflow (where to move it, how long to keep it)?
- Storage hierarchy levels are not independent – data can exist in several locations, with newer versions on compute node storage, older versions on parallel file system. The metadata becomes much more complicated than it is today.
- How can users find their data in such a complex system?

How do we characterize user workloads?

- There are very many types of user workloads from experimental data analysis, but at the 50k foot view they are very similar. HPC workloads tend to write some data during a time step.

  There are different workloads than that though, e.g. ensemble applications, data analysis applications. In general we can think of them as bulk output and asynchronous I/O patterns.
- What will be the requirements of data analytics workloads? Random I/O? No workload characterizations of these yet.

  Random I/O is hard to support – either it fits in the storage tier you are using or it doesn't.
- We really need to have good characterizations of the applications to understand what they are doing

Will we be able to get away from application specific frameworks that solve a specific problem?

- ADIOS, HDF5 provide application-specific interfaces that have proven very useful in some domains

- If we have higher level application interfaces that map to some common API (e.g., POSIX) is that the answer?
- The cloud world offers many interfaces/services for different workloads, seems to be a better idea than trying to shoehorn in an application into a particular I/O model
- In a layered model (application interface over lower-level common interface) they key will be to maintain interoperability
- Can we renegotiate the contract with users on what the interface is? Google never asks. Well, in HPC users are the boss.

Can we make changes to POSIX or define a POSIX-like interface with relaxed constraints?

- Do users understand POSIX? Do we even understand all of POSIX? No.
- Applications developers will not change their I/O approach unless we demonstrate clear, strong benefit. The benefit of POSIX is that it is stable, well benefit and hindrance.
- Relaxing the constraints of POSIX could be a powerful approach

8.6 Discussion: What should we do about POSIX? *Leads: Rob Ross and Andreas Dilger*

The group discussed what to do about the POSIX interface going forward. It has drawbacks and can prevent I/O performance due to its strict semantics. However, it is widely used and has been stable for a very long time.

Do we have a hope of getting away from POSIX?

- There are a mass of tools and applications that already use POSIX, legacy codes. Can we expect them to change?
- Are we stuck with POSIX as long as we have parallel file systems?

How successful are middleware libraries at getting around the problems of POSIX?

- The difficulties of implementing ROMIO on top of POSIX really exposed the problems of POSIX and that implementations of POSIX do not uphold the contract of the API
- Log structured optimizations preclude being able to read the file from POSIX
- Can we write object stores on top of POSIX? There are not products that do this now, but there could be. However it is a semantic mismatch.

Can we keep POSIX and relax the semantics in a defined way? A new contract?

- APIs are a contract, but can we define how to relax the semantics? Eg. relax the semantics on the burst buffer but uphold them on the parallel file system?
- Perhaps upholding POSIX semantics is easier in the burst buffer because of fast compute network
- POSIX is not good for coupled applications communicating through the file system
- Maybe POSIX could be the contract between the I/O middleware and the backend store. We could give applications something else to program to without the strict semantics of POSIX
- POSIX is going to be around a long time. We need a new way to express our storage requirements with or without POSIX
- Perhaps as a community we could come to a consensus on key ways to extend and relax POSIX, e.g. NFSv4 semantics

9 Recurring Themes and Future Work from Seminar

Overall, the week of the seminar was used productively. The group enjoyed the balance of short talks and discussion sessions. Since this was the first time that this group had come together, it was beneficial to have the short talks to give everyone an overview of people's backgrounds and expertise. The primary results of the seminar were identification of needs that the HPC I/O community should address in the approach to exascale computing. These needs arose as recurring themes throughout the seminar week.

9.1 Recurring Themes

Throughout the talks and in-depth discussions in the seminar, the participants identified several recurring themes that represent items that need to be addressed by the HPC storage and I/O community in the coming years as we come to terms with hierarchical storage systems. The needs that were identified during the seminar were:

- The need to characterize I/O behavior of applications
- The need to re-evaluate the POSIX I/O interface
- The need to advance resource management and scheduling policies for I/O management
- The need to define users' data management requirements.

The need to characterize the I/O behavior of applications. Seminar participants identified the need to characterize the I/O behavior of applications in several sessions. In general, the participants felt that as a community we don't have a good grasp on the aspects of application I/O behavior that are needed to design high-performance storage systems. For example, are HPC applications well-behaved in that if they write shared files, does each file offset only get updated by one process? Or do the applications assume that the file system implements locking and thus the writes of different processes to the same offset will have some ordering enforced? The answers to these questions will inform storage system design. In the case of this example, if applications are well-behaved, then a storage system will not need to implement expensive locking and thus I/O operations can have better performance. However, if the application depends on locks for coherent file data, then the storage systems will have to support that.

Quite a bit of discussion in the seminar centered on how to generate a set of mini-apps that mimic true I/O behavior. Much of the work in mini-app design is centered around other aspects of application behavior, e.g., computation or communication patterns, and hardly any attention is given to I/O behavior. Of the few I/O mini-apps that do exist, for the most part, the I/O patterns that are emulated are too abstracted to be of use for next-generation storage system design and simply focus on bulk write and read phases. We need to find or develop mini-apps that mimic different application I/O patterns and are true to varying behavior based on different inputs and scaling of runs. We need to ensure that we collect mini-apps that are representative of all major classes of application I/O behavior, from classic bulk-synchronous checkpoint/restart, to newer I/O models for in-situ analysis or machine-learning workloads.

As part of the I/O characterization effort, we also need to focus on measurement of real applications running on systems. This measurement effort can aid in the generation of useful I/O mini-apps, but also provide additional information for designing storage systems and scheduling jobs and I/O operations on future systems. While several quality I/O measurement tools exist today, I/O measurement is usually done as an ad hoc activity, measuring the I/O operations of a particular application to understand its behavior and performance. While

this is helpful for that particular application on the system it was run on, it does not give a picture of how that application's performance was affected by concurrently running jobs that may have been using network or storage resources, and it does not capture workflow-level I/O interactions where a series of application runs over a long-running, months-long experiment may share file data. Thus, it will be useful to have detailed I/O measurement of applications on a center-wide basis and to couple that with storage system and resource management information. Several centers are making headway in this regard currently, but given that workloads tend to vary across HPC centers, it will be beneficial to have more centers invest in in gathering broad-spectrum I/O information from jobs.

The need to re-evaluate the POSIX I/O interface. The topic of POSIX came up quite often during the week of the seminar. Participant opinions spanned from full rejection and replacement of the ubiquitous standard to acceptance and possible redefinition of semantics thought to be too restrictive for today's and next-generation HPC storage systems. In general, most participants agreed that the POSIX I/O interface is too widely used and relied upon by legacy HPC applications to imagine its disappearance any time soon. Additionally, as a community, we do not have an alternative to POSIX that is as stable and portable to offer application developers.

In general, participants did not envision a near-future end to POSIX, and many felt that there was an opportunity to possibly alter the semantics of POSIX to be less restrictive in order to better support HPC workloads. This effort would require understanding the needs of HPC I/O workloads with respect to file integrity, which further emphasizes the need for comprehensive I/O characterization and measurement as discussed above. However, even before full understanding of application I/O behavior is known, as a community we have a sense of the limitations of POSIX semantics and can make an effort to systematically identify these limitations and define structured relaxations to the POSIX I/O semantics. For example, well-behaved applications that write shared files may be able to specify that locking is not needed by providing a defined flag to the interface. Participants of the seminar plan to begin a community effort for this need.

The need to advance resource management and scheduling policies for I/O management. With hierarchical storage systems becoming a reality for many centers, the consensus of the seminar participants was that resource manager and scheduling policies have not been updated to account for them. Current production resource managers and schedulers only account for the computational requirements of jobs, the node count and run time. However, in order to achieve better system utilization and throughput, resource managers and schedulers should account for new storage system models. For example, node-local storage devices like burst buffers may need to be allocated and shared across jobs (the currently executing job, the previous job that may be draining data, and the next job to run that may have data pre-staging into the node). All of these will require resources including storage capacity, possibly some CPU time depending on the storage device, and network resources. If resource managers and schedulers are able to allocate resources and schedule jobs to minimize the interference of competing I/O activities, then the result will be higher overall system performance.

There will be significant additional complexity in accounting for the resources needed for I/O in hierarchical storage systems. As a community, we plan to work with researchers in the area of resource management to provide input on how storage and network resources might be used on future systems in order to prepare.

The need to define users' data management requirements. User applications are reading and writing data at volumes greater than ever and the trend promises to continue in the

future. This trend towards large data volumes was the impetus behind hierarchical storage systems, where storage devices closer to compute nodes could act as write-through caches or buffers for the I/O operations between stable storage (e.g., the parallel file system) and compute nodes, or alternatively to use the cache tier as temporary storage for data that only has a temporary life span and does not need to be transferred to the parallel file system. The canonical example of the temporary use case is checkpoint/restart, where checkpoint files can quickly be written to the cache tier only, and upon writing a new checkpoint file, an older checkpoint can be deleted since it will not be needed anymore. Another example of using the cache tier as a temporary storage layer is of an in-situ analysis component coupled with a simulation application. In this case, the simulation could output large data files on the cache tier, which could be analyzed in-situ by the analysis component. Following analysis, the files could either be deleted if not needed further, or perhaps only "interesting" output files (as determined by the analysis component) could be transferred to the parallel file system.

In general, hierarchical storage systems offer new options for I/O operations of HPC applications, but there is no general way for applications to specify how a particular data set should be managed on the storage tiers. Examples of information that could be specified include whether the data is temporary or if it should be persisted on stable storage; the requirements on resilience of the data, where perhaps the user can afford to lose some data files with a specified probability; and time before data must be persisted on stable storage, where a user might optionally allow a very slow drain of particular data if it is not needed immediately.

As a community, we agree that we should come together with use cases and requirements in order to develop a specification for users to provide the needs of their data sets. The specification will enable storage system developers to provide higher performance implementations because they will not need to implement the most rigorous case of persisting all output data on stable storage.

9.2 Future Work

The participants of the seminar tended to agree that our community should engage in more meetings of the style we experienced in this Dagstuhl Seminar. The discussion-oriented meeting was fruitful in teasing out overarching themes that represent future work items that need to be addressed by our community. The participants indicated that future seminars could be centered around designing community-supported solutions to one or more of the themes that we identified during this seminar. The group also indicated that they would like to participate in a future joint publication effort, e.g., a special issue journal containing papers from our collective efforts related to this seminar to serve as a basis for future research.

Participants

- Stergios V. Anastasiadis
University of Ioannina, GR
- Ned Bass
LLNL – Livermore, US
- John Bent
Seagate Government Solutions – Herndon, US
- Thomas Bönisch
HLRS – Stuttgart, DE
- Luc Bougë
INRIA – Rennes, FR
- Andrë Brinkmann
Universität Mainz, DE
- Suren Byna
Lawrence Berkeley National Laboratory, US
- Philip Carns
Argonne National Laboratory, US
- Lance Evans
Cray Inc. – Seattle, US
- Wolfgang Frings
Jülich Supercomputing Centre, DE
- Hermann Härtig
TU Dresden, DE
- Dean Hildebrand
IBM Almaden Center – San Jose, US
- Nathan Hjelm
Los Alamos National Laboratory, US
- Scott Klasky
Oak Ridge National Laboratory, US
- Michael Kluge
TU Dresden, DE
- Michael Kuhn
Universität Hamburg, DE
- Jay Lofstead
Sandia National Labs – Albuquerque, US
- Satoshi Matsuoka
Tokyo Institute of Technology, JP
- Alberto Miranda
Barcelona Supercomputing Center, ES
- Kathryn Mohror
LLNL – Livermore, US
- David Montoya
Los Alamos National Lab., US
- Federico Padua
Universität Mainz, DE
- Mark Parsons
University of Edinburgh, GB
- Maria S. Perez
Universidad Politëcnica de Madrid, ES
- Franz Josef Pfreundt
Fraunhofer ITWM – Kaiserslautern, DE
- Raghunath Raja Chandrasekar
Cray Inc. – Seattle, US
- Robert B. Ross
Argonne National Laboratory, US
- Brad Settlemyer
Los Alamos National Laboratory, US
- Osamu Tatebe
University of Tsukuba, JP
- Tianqi Xu
Tokyo Institute of Technology, JP
- Weikuan Yu
Florida State University – Tallahassee, US
- Yue Zhu
Florida State University – Tallahassee, US