Exploring Opportunities for Job-temporal File Systems with ADA-FS

Sebastian Oeste, Michael Kluge
Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden
Dresden, Germany
Email: {sebastian.oeste, michael.kluge}@tu-dresden.de

Mehmet Soysal, Achim Streit
Steinbuch Centre for Computing (SCC)
Karlsruhe Institute for Technology (KIT)
Karlsruhe, Germany
Email: {mehmet.soysal, achim.streit}@kit.edu

Marc-André Vef, André Brinkmann
Johannes Gutenberg University Mainz
Institute of Computer Science
Efficient Computing and Storage Group
Mainz, Germany
Email: {vef, brinkman}@uni-mainz.de

***Abstract*—In modern HPC systems, parallel (distributed) file systems are used to allow fast access from and to mass storage infrastructure. However, they are bound to static and predefined application demands. Since the amount of data required by scientific applications grows and the methods to access the data change, an additional migration of data between the parallel file system and the compute nodes is unavoidable. As a result, the I/O subsystem becomes progressively more the bottleneck of the whole HPC system. With ADA-FS, we design and develop an infrastructure for intelligent I/O planning and data staging at extreme scales by utilizing idle storage subsystems on compute nodes for ad-hoc job-temporal file systems. In this position paper, we show how various required components and information, such as a system's topology, interact and how the ad-hoc file system works. We present and evaluate early prototypes of these components and explore design properties of modern and future pre-exascale systems.**

1. Introduction

Today's HPC systems utilize parallel file systems that comply with POSIX semantics, such as Lustre [1], GPFS [2], or BeeGFS [3]. Despite separating data and metadata to improve performance, metadata performance cannot scale by adding more hardware to the storage subsystem and, therefore, is much harder to tackle. In the past, research and development on parallel file systems focused on increasing the bandwidth of read and write operations in a parallel manner. Still, this I/O bottleneck is present today. Moreover, parallel file systems (and their I/O subsystem) are often shared by many users and their jobs. Consequently, bad behaving applications can result in poor performance affecting all users.

In February 2014, *The Advanced Scientific Computing Advisory Committee* (ASCAC) released a report about the top 10 challenges towards developing exascale systems [4]. One of them is the creation of a management software which is able to handle the volume at an adequate speed and can cope with the diversity of anticipated data. Hence, exascale systems are going to need storage systems that are able to scale in many directions. Building exascale systems with technologies available today would mean to invest a tremendous amount of hardware.

Many of the recently announced large scale HPC systems (for example AURORA [5], Sierra [6], and Summit [7]) will offer node local storage ranging from SSDs, through NVRAM, to high bandwidth local persistent storage. The ADA-FS project is aiming at a solution where this node local storage is used to create an on-demand private parallel file system, created exclusively for a single job, and removed after it finished. As a result, we expect much more efficient I/O patterns to a shared global file system assuming that the bottleneck is the aggregate link bandwidth between the compute nodes and the global parallel file system or the global file system itself. In this paper, we are going to describe our expectations from exascale machines and how our project will help to reduce the the bottleneck between very fast node local storage and a slower global parallel file system.

First, we depict related work on the previously mentioned topics in Chapter 2. In Chapters 3 and 4, we describe the components of our project and show with small benchmarks how our concept could help with future exascale systems. Chapter 5 discusses how our approach should behave on future exascale systems and which parts needs to be considered compared with today's supercomputers. In Chapter 6 we introduce we plan to collect resource information on heterogeneous systems. We discuss existing solutions for gathering static resource information and briefly present our architecture. Finally, we will conclude our work in Chapter 7.

2. Related work

In this section, we explore design properties of announced pre-exascale and state of the art systems. In addition, we evaluate modern solutions of four critical aspects that we aim to improve.

State of the art. Planned future pre-exascale system are announced with node local storage for caching of files or using it as a extended memory. Due to dropping hardware

prices in the recent years, more and more HPC solutions utilize small SSDs in their compute nodes. Thus, a lot of systems have a break-even point where the combined local I/O capabilities of some nodes exceed those of the global file system. Table 1 shows an expected extrapolation of

TABLE 1. TODAY AND FUTURE HPC SYSTEMS

HPC system	# compute nodes	Total I/O in GiB/s (S)	Caching bandwidth in GiB/s (C)	Break-even point $P^* = S/C$
HRSK2 at TUD	2,014	80	0.5	160
ForHLR II at SCC	1,172	50	0.5	100
TITAN ar ORNL	18,688	240	0.5	480
Aurora at ANL	>50,000	1,000	1.6	640
Summit at ORNL	>3,400	1,000	1.6	640

S: Total global parallel storage bandwidth
C: Node local storage conservative speed assumption
P^*: Break-even point: ad-hoc storage is as fast as the total I/O

the break-even point for the I/O bandwidth. *HRSK2* at TU Dresden and *ForHLR II* [8] are mid-ranged HPC systems, both using the Lustre parallel file system, are equipped with SSDs in their compute nodes. The upcoming *Summit HPC System* [5] was announced with NVM-e like SSDs and the *Aurora System* [7] will use some kind of local high bandwidth (persistent) memory. Both systems are planned with a global parallel file system bandwidth of 1 TiB/s. On the other hand, the interconnect bisection bandwidth was announced with 500 TiB/s in Aurora, offering 500 times more bandwidth within the compute nodes compared to the global file system. Hence, we can assume that the interconnect is not a bottleneck in our application. Considering the break-even point $P^* = S/C$, where the average bandwidth is equal to the interconnect bandwidth within a job, all jobs with $P > P^*$ may achieve better performance in our approach. If a job is asking for less nodes, it would not reach the break-even point but it would also not have to share the private parallel file system with other jobs. While we assume that the hardware, regarding to compute node local SSDs and the interconnect, is going to be fast enough for our approach, we have to design high-performing and easy to use software to make the local storage usable. For this, ADA-FS evaluates aspects from the research areas of data-aware scheduling, parallel file systems, and monitoring.

Data-aware Scheduling and data-staging. In the past, Monti et al. [9] have shown that HPC systems will benefit drastically if the data staging process is included in the scheduler's decision, i.e., moving computing resources to the corresponding data or vice versa. *The Parallel Runtime Scheduling and Execution Controller* (PaRSEC) is an architecture aware scheduling framework developed at the University of Tennessee at Knoxville [10]. *The Hobbes Project* at Sandia National Labs has a similar component [11]. Data staging also reached commercial products, such as the *MOAB workload manager*. However, the above mentioned solutions are only covering single aspects of data scheduling.

Parallel and ad-hoc file systems. In recent years, several parallel file systems have evolved. Unfortunately, the capabilities of these systems struggle at extreme scales, resulting in poor performances for accessing applications. One common bottleneck is the available bandwidth and the

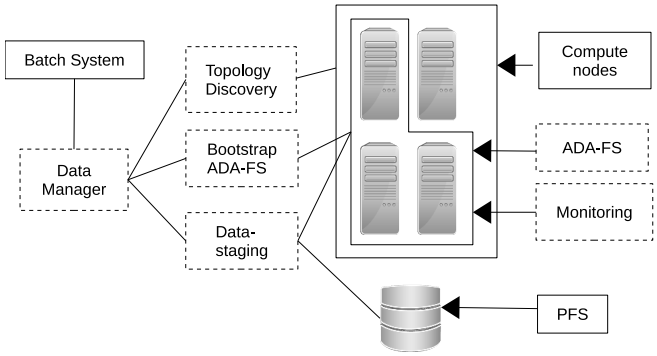

Figure 1. Overview of ADA-FS

interface of storage and compute nodes in which the bisection bandwidth grows at a faster pace than the bandwidth to the file system. Another observable obstacle in production systems are single, I/O intensive jobs, able to severely disrupt other applications that access the same file system. Some parallel file systems, such as Spectrum Scale with its *File Placement Optimizer* (FPO) try to mitigate these issues by using local compute node SSDs as a buffer for the global file system. However, neither of these solutions offers a way to stage-in data to a buffer or on-demand file system before a job start.

Data management. Existing data staging solutions, e.g. MOAB, require users to manually specify the amount of stage-in and stage-out data to receive an estimation for the scheduling time. This is problematic because of two reasons: First, it is hard (and often impossible) to know how much data is needed and produced from a particular job. Second, giving a precise estimation of scheduling time is not only depended on the amount of data but also on the number of files. Metadata intensive workloads are known to cause significantly higher load on the storage subsystem with less data compared to data intensive workloads.

Monitoring. An important aspect for our approach is monitoring and resource discovery. The *DARSHAN project* [12] has established methods for analyzing I/O requests in a holistic way in an HPC storage environment. *Hwloc* is a set of tools to collect local resource information [13]. The *LIKWID* performance monitoring tool suite is also able to gather topological information of the system [14] by using hwloc. Global network information can be collected by directly using tools tailored for a certain kind of network, for example `ibroute` and `ibnetdiscover` tools from the *OFED distribution* [15] for InfiniBand topologies.

3. Setup of the ADA-FS Project

ADA-FS is sponsored by the *German Research Foundation* within the priority program *"Software for Exascale Computing"*. The goal is to develop an ad-hoc file system with an integrated data scheduler that uses the local I/O capabilities of those compute nodes that a job allocates but

still integrates with the globally shared parallel file system. Thus, each job gets its own parallel file system without the need to share resources. We plan to to offer an on-demand private parallel file system, exclusively created and tailored to a single job, by utilizing available local compute node storage. Additionally, we develop efficient data-aware scheduling techniques to stage-in data before they are required for computation. Figure 1 shows an overview of the components (dashed rectangles) of ADA-FS. All of our components are intended to work together with existing HPC system components, such as resource managers or schedulers.

Data manager. The data manager will be responsible for data scheduling. When data is moved through an HPC system, methods are needed to track this movement. We are going to introduce the concept of *workpools*, which are similar to the workspaces in *workspace++* [16]. This will virtualize the actual data location on the file system. The data can be staged anywhere and the user locates the workpool with a command line interface. Existing solutions neither offer information about the amount of data per workspace nor have staging capabilities.

Data Staging. The process of data staging will be transparent for a end-user. Based on the changes a job made in its local temporary file system, the data is staged-in and staged-out by this component whenever the data manager assigns a time slot for this action.

Bootstrap ADA-FS. This component spawns and manages the requested ad-hoc file system. In our approach, an ad-hoc file system has to be available and accessible only for the job duration. The ad-hoc file system does not need the same fault tolerance as the global file system, but should be resilient against single node failures. In this early stage of the project, the bootstrap is launched by a prologue program before a job starts and is going to be as generic as possible to work with any batch system. It is supplied with all information for the needed ad-hoc file system which is created by utilizing either BeeGFS or IBM Spectrum Scale.

ADA-FS. Parallel file systems, such as IBM Spectrum Scale [2] and Lustre [1], are POSIX compliant to support a wide range of scenarios. However, it is known that providing full POSIX compliance hurts parallel I/O performance. As a result, some parallel file systems, e.g. Ceph-FS [17], are able to relax POSIX semantics for specific operations to improve performance. We plan to extend this idea and remove particular POSIX rules where feasible. For example, we assume that each file/object is only accessed by a single application (in parallel from multiple nodes) at a time. Consequently, consistency has only to be ensured within the jobs' nodes. Further, we expect that known problematic operations, for example `ls -a`, are not performed during an application run.

Monitoring and topology discovery. In order to allocate the required resources for an ad-hoc file system, it is essential to provide vital information, such as the static composition and topology of the system as well as the dynamic resource usage. As an input for the data staging component, the resource and topology information are necessary. We are going to build these on top of existing frameworks. Moreover, monitoring of the ad-hoc overlay file system and the I/O behavior of parallel applications is done to optimize I/O planning and data placement.

4. Initial Benchmarks

To evaluate boundary conditions and to assess the assomptions from Table 1 we performed initial benchmarks for a few aspects of ADA-FS. These were done on 256 nodes of the ForHLR II system. Each node has 64 GiB memory, 2 Intel Haswell Xeon E5-2660v3 (10 cores), and a local SSD with approximately 400 MiB/s write performance. As the experiment was executed on a production system, we created a file on each node and associated a loop device to the file. The loop device was given to the tested file systems (BeeGFS and IBM Spectrum Scale) directly. We will only show a few results here. In the case of BeeGFS, we created and mounted a file system on the loop device. First, we analyzed how fast file systems can be deployed on the compute nodes and evaluated the bandwidth with the iozone benchmark. Table 2 indicates that the startup time increases with the number of nodes. While the startup times might be acceptable on smaller HPC systems, they are not feasible in massive systems and at extreme scales. An integration into the resource manager prologue would shorten this time. Additional metadata experiments were performed with the

TABLE 2. BEEGFS STARTUP AND THROUGHPUT

Nodes	8	16	32	64	128	256
Startup (s)	10.21	16.75	29.36	56.55	152.19	222.43
Shutdown (s)	11.90	12.13	9.40	15.96	36.13	81.06
Throughput (GiB/sec)	2.79	6.74	10.83	28.37	54.06	129.95

Throughput test: iozone (3.46) with one process/node

mdtest benchmark [18], showing that IBM Spectrum Scale performs better for metadata intensive operations. However, IBM Spectrum Scale was harder to integrate into such an dynamic environment as BeeGFS, which contains a tool that supports an easy deployment.

Another relevant component, whose performance characteristics are of interest, is *fuse* (*file system in user space*) which comes with its own kernel module and a corresponding user space library. Fuse allows an lightweight integration of new storage concepts or file systems without the need to write a kernel module. Thus, it provides a natural starting point for the development of the ad-hoc file system. However, working with fuse seems to increases the number of context switches significantly. For instance, an I/O system call, such as the `open()` operation, will initially always call the *Virtual File System* (VFS) in the kernel. Afterwards, the VFS passes the request to the fuse kernel module and sends it back to the libfuse library and to the application implementing the fuse file system.

To quantify this impact, we employed a pass-through file system with fuse[1] which prepends the root of the fuse file

1. https://github.com/ada-fs/ptfs

system to the path and calls the corresponding underlying system call. A self-written microbenchmark mounts the ramdisk as root of our fuse file system and writes data using `dd` to it. Table 3 shows the average bandwidth, total time and context switches, for an execution of the micorbenchmark with fuse and without fuse. The values in Table 3 are the accumulated average of multiple runs. We can see that the bandwidth and the total execution time of the benchmark is 5 times higher when running without fuse compared to runs with a fuse file system, increasing the number of context switches by a factor of 2000.

TABLE 3. FUSE MICROBENCHMARK

Benchmark	Bandwidth in MiB/s	Total time in s	# context switches
Fuse on RAMDisk	474	22.64	4,385,395
Plain on RAMDisk	2,400	4.49	1,697

Accumulated average bandwidth, total execution time, and number of context switches for the `dd` microbenchmark on a plain *tmpfs* and through a fuse pass-through file system.

This indicates that reusing a native parallel file system has severe performance advantages but also drawbacks in terms of development and management. Nonetheless, we argue that the advantages of using a fuse file system outweigh the performance issues. As part of our project, we will further analyze fuse's behavior in more detail and optimize our setup to mitigate the performance impact of fuse.

5. Projections towards exascale

In the future, the performance gap between the global parallel file system, the network between the compute nodes, and node local I/O devices is going to increase. For example, the Aurora HPC system will have 500 TiB/s bisection bandwidth and only 1 TiB/s bandwidth to the file system. Also, new technologies, such as NVRAM or persistent local memory, are will be used in HPC systems. Thus, we expect ad-hoc file systems to perform well on exascale systems. By design, the file system will be independent of the number of nodes that it runs on since the distributed meta data and deployment process can be parallelized.

The huge increase in the size and and complex topology hierarchy of supercomputers makes it very difficult to predict a scheduling algorithm and its performance. As those existing algorithms have to evolve or completely has to be developed from scratch, our approach will have a small impact on the scheduling. Data staging is only adding another factor onto the scheduling.

In terms of collecting resource information, we do not expect issues in future exascale systems. Local resources can be collected as it is done today in parallel on all nodes. Global connection information are most probably available within the batch system, network routing, or management software. The latter has many ways to adapt to changes in the system configuration. Our approach will help to close the increasing bandwidth gaps by intelligently using all available resources in modern as well as future HPC systems.

6. Collection of Resource Information

As mentioned before in Chapter 3, detailed information of hardware resources and the network topology of a system is important in our approach. Only with appropriate information about the used underlying storage technology and corresponding bandwidths of the communication and storage networks, the data manager will be able to make the right decisions. In this section we describe what kind of resource information is of interest and depict possible ways to gather this information. Furthermore, we will discuss the architecture of a tool that we build to gather the relevant resource information for ADA-FS.

First, we distinguish between static and dynamic resource information. At its core, static resource information describes components that do not change frequently and are often similar between nodes. This includes the number of CPU cores, the amount of main memory, and cache sizes etc. of a single node. Because it is unlikely that this kind of hardware is replaced frequently, it is not necessary to update this information before each job. However, dynamic resource information describes the utilization of the static resources, e.g. the average load on the storage or the communication network. The difference between static and dynamic resources gives an estimate of the available resources that can be used by ADA-FS.

For static resource information, we started to implement a system-map tool which provides a map of parts or the whole HPC system. This information can be queried by the data manager directly. As described in Section 2, there are already several tools which gather information about the topology and hardware resources of a system. *Hwloc* provides many output formats and supports a wide range of architectures. `lstopo` collects node local information about NUMA domains, memory, and cache sizes. In addition, PCI-Express sockets and devices are collected as well. Additional information, e.g. file systems mounted on these block devices or available partitions, are not discoverable with this approach.

Therefore, we are extracting information from the *procfs* and *sysfs* pseudo-filesystems to learn about the number of partitions, appropriate sizes, mountpoints, and the used file systems, for example. Moreover, we plan to gather network topology information for InfiniBand networks by utilizing the well known `ibroute` and `ibnetdiscover` tools from the OFED distribution [15]. Hence, we have designed an extensible architecture for our system-map tool. Each resource of interest will be utilized by a so called *extractor*. Figure 2 shows a schematic UML-diagram of two *extractors* from our system-map tool. An *extractor* module has an abstract part, which defines the structure of the data that will be gathered and a specialized part which implements the logic to read the data from a certain source, by overriding the abstract interface. In Figure 2, the `Filesystem_Extractor` and the `Disk_Extractor` are examples of the abstract parts. The `Linux::Filesystem_Extractor` and the `AIX::Filesystem_Extractor` are the specialized

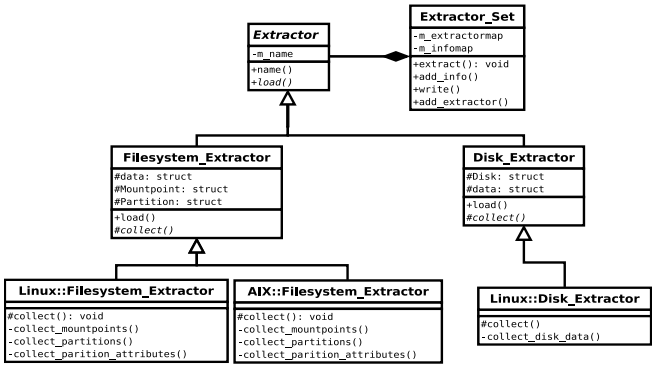

Figure 2. Simple UML-Diagram for two example *extractor* modules of our system-map tool.

parts for extracting information of mountpoints and partitions of a specific system. If a specific source is available on a system the appropriate *extractor* will be used to extract the information. This is useful because the same information may be available on different systems through different sources. With this architecture, we are able to implement specialized *extractor* modules for different sources resulting in an equivalent representation of the data for our tool. All kinds of post-processing after gathering the information can rely on a defined representation of the data. That gives us the opportunity to support different output formats or query interfaces for other components of ADA-FS at a higher level of the tool. Since hwloc is good for getting CPU and cache topology information, we add an *extractor* which uses hwloc particularly for this kind of information. The ADA-FS sysmap topology discovery is work in progress while we are also working on integrating network topology discovery to make bisection bandwidth calculations possible.

7. Conclusion

Within the ADA-FS project, we will create job local file systems that exploit the I/O capabilities of the compute nodes in current and exascale HPC systems. This allows users to run I/O intensive jobs in parallel, but isolated. With our approach idle times of resources for applications involving massive amounts of data will be reduced. We have already developed some prototypes of the needed components: an ad-hoc file system, a data manager, a data staging component, and a monitoring service. As this project is work in progress, some components, such as the algorithms for the data scheduler, still need to be researched. In the future, we are going to tightly couple and integrate all parts of our system, test and mature the components.

8. Acknowledgement

The project ADA-FS is funded by the DFG Priority Program “Software for Exascale Computing” (SPPEXA, SPP 1648), which is gratefully acknowledged.

References

[1] Qihai Zhou and Yuan Wang and Yongquan Lu and Chu Qiu and Pengdong Gao and Jintao Wang. 2011 2nd International Conference on Challenges in Environmental Science and Computer Engineering (CESCE 2011) Performance Evaluation of A Infiniband-based Lustre Parallel File System. *Procedia Environmental Sciences*, 11:316 – 321, 2011.

[2] Frank Schmuck and Roger Haskin. Gpfs: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST ’02, Berkeley, CA, USA, 2002. USENIX Association.

[3] Jan Heichler. An introduction to BeeGFS. http://www.beegfs.com/docs/Introduction_to_BeeGFS_by_ThinkParQ.pdf, 2014. Accessed: September 6, 2016.

[4] ASCAC Subcommittee for the Top Ten Exascale Research Challenges. The top ten exascale research challenges. Technical report, US Department of Energy, Office of Science., February 2014.

[5] OAK RIDGE National Laboratory. Summit fact sheet. https://www.olcf.ornl.gov/wp-content/uploads/2014/11/Summit_FactSheet.pdf, 2014. Accessed: August 23, 2016.

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

[7] INTEL Cooperation. Argonne National Laboratory’s Aurora SystemUshering in a New Era. http://www.intel.com/content/www/us/en/high-performance-computing/intel-argonne-aurora-announcement-presentation.html. Accessed: August 23, 2016.

[8] Steinbuch Center for Computing. Forschungshochleistungsrechner ForHLR 2. http://www.scc.kit.edu/dienste/forhlr2.php, 2016. Accessed: August 16, 2016.

[9] H. M. Monti, A. R. Butt, and S. S. Vazhkudai. On timely staging of hpc job input data. *IEEE Transactions on Parallel and Distributed Systems*, 24(9):1841–1851, Sept 2013.

[10] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J. J. Dongarra. Parsec: Exploiting heterogeneity to enhance scalability. *Computing in Science Engineering*, 15(6):36–45, Nov 2013.

[11] Ron Brightwell. Hobbes - an operating system for extreme-scale systems. http://xstack.sandia.gov/hobbes/, 2014. Accessed: September 05 2016.

[12] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)*, 7(3):8, 2011.

[13] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc: A generic framework for managing hardware affinities in hpc applications. In *PDP 2010-The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, 2010.

[14] Jan Treibig, Georg Hager, and Gerhard Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *2010 39th International Conference on Parallel Processing Workshops*, pages 207–216. IEEE, 2010.

[15] OpenFabrics Alliance. Open fabric enterprise distribution. https://www.openfabrics.org/, 2016. Accessed: August 17 2016.

[16] Holger Berger, Beisel Thomas, and Martin Hecht. HPC Workspace. https://github.com/holgerBerger/hpc-workspace, 2016. Accessed: August 16, 2016.

[17] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.

[18] Alfred Torrez, Brett Kettering, William Loewe, and Ruth Klundt. mdtest metadata benchmark. https://sourceforge.net/projects/mdtest/, 2015. Accessed: September 4, 2016.