Ad-hoc file systems at extreme scales

Mehmet Soysal, Achim Streit

Abstract This work presents the results of the project with the acronym ADA-FS (Advanced Data Placement via Ad-hoc File Systems at extreme scale). The project which has been approved for the ForHLR II aims to improve I/O performance for highly parallel applications by using distributed on-demand file systems. These temporary file systems are created on the allocated compute nodes, using the node-local disks for the on-demand file system. Through integration into the scheduling system of the supercomputer, it can be requested like any other resource. The research approach contains the design of the file system itself as well as the questions about the right planning strategy for the necessary I/O transfers. In the granted project for the ForHLR II we are investigating the methods on how to integrate the approach into a HPC system. Also, we are evaluating the impact of the on-demand created file systems to running HPC jobs and the applications.

Key words: file systems, on-demand file systems, wall time, data staging

1 Introduction

Today's HPC systems utilize parallel file systems that comply with POSIX semantics, such as Lustre [1], GPFS [2], or BeeGFS [3]. The storage subsystem within HPC systems is increasingly becoming a bottleneck. Furthermore, the performance is limited by the interface between the global file system and the compute nodes. Moreover, parallel file systems (and their I/O subsystem) are often shared by many users and their jobs. When users develop applications for HPC systems, they typically tend to optimize for computing power, sometimes disregarding the I/O behavior of the application. While the computing resources can often be allocated exclusively, the global PFS is shared by all users of a HPC system. This environment makes it difficult for the user to optimize the application concerning I/O. There are many possible factors a user would have to consider. Influences from the back-end

Steinbuch Centre for Computing, Karlsruhe Institute of Technology
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany e-mail: mehmet.soysal@kit.edu, achim.streit@kit.edu

storage device, network interface, storage servers, data distribution, request size, and other applications slowing down the PFS [4].
One of the reasons why PFSs struggle with certain I/O operations is that they have to cover a wide range of applications. In addition, the PFS must be robust with high availability as the HPC system is dependent on the global storage system. But there are applications and scenarios that do not suit the general case. This includes scenarios and cases in which large amounts of data or millions of small files are generated, causing high load on the storage system. Consequently, bad behaving applications can result in poor performance affecting all users. The ADA-FS project aims to improve I/O performance for highly-parallel applications by a distributed ad-hoc overlay file systems. In order to achieve our goals, several challenges need to be addressed.
The first step is to find a way to deploy these on-demand file systems on production systems. It has to be minimal invasive and should not involve any changes to the operating model. This initial step also includes performance measurements with synthetic benchmarks. The results for this first step are represented in Chapter 3. Another point in our approach is the question, if the data can be pre-staged to the allocated compute nodes. For this challenge it is important to know which nodes are going to be allocated to a waiting job. To this end, we investigated how the run times of jobs can be predicted and how good they must be to allow our approach. Methods from the field of machine learning were used here, to predict run times. Also we simulated different workloads of HPC systems and evaluated the impact of improved wall time estimates. In Chapter 4 we present our evaluation regarding this part of the project. The next Chapter includes applications and use cases of our users. Here we present how much performance is achievable with our approach and what impact we have on applications and the HPC system. For this we picked three different applications from our users. We examined the application behavior with the on-demand file system and how our approach can help these use cases. We present the results of the real usage scenarios in Chapter 5. First we start with the related work in the next Chapter 2 and conclude at the end with a summary of the approved project.

2 Related Work

The project covers different scientific domains, e.g., machine learning, file systems, scheduling and a wide range of applications. In this chapter we give a brief introduction in the important parts of the related work.

2.1 I/O

In the recent past there have been many developments and innovations to improve I/O throughput and performance. We cannot cover everything and try to give a brief overview of existing solutions. Many solutions are implemented at multiple levels in the I/O stack, but four basic categories can be formed: file system features, hardware solutions, libraries and dynamic system re-configurations.

File system features

File Systems have received interesting new features to reduce I/O bottlenecks. BeeGFS offers storage pools [5] to group storage targets in different classes, e.g., one pool with very fast solid state drives. GPFS has implemented a Highly Available Write Cache (HAWC)[6]. Node-local *solid-state drives* (SSDs) are used as buffers for the global file system. As a result, random I/O patterns are processed on local storage. Lustre has the Progressive File Layouts (PFL) [7] feature, which adjusts dynamically the stripe pattern and chunk size based on I/O traffic.
However, such solution are only available when using the vendor software solution.

Hardware solutions

Today's wide spread use of SSDs in compute nodes of HPC systems has provided a new way of accelerating storage. SSDs have been considered for file system meta-data [8] [9], as its meta-data performance is a major bottleneck in HPC environments.
A different kind of hardware solutions are burst buffers, which aim to reduce the load on the global file system [10].

Libraries

There is a large number of libraries available for improving I/O behavior of an application. Middleware libaries, such as MPI-IO [16], help to improve usage of parallel storage systems, e.g. data sieving and collective I/O [17]. High-level libraries, such as HDF5 [18], NETCDF [19] or ADIOS [20], are trying help users to express I/O as data structures and not only as bytes and blocks. These libraries are not in contrast to our approach. The advantages of using such libraries also apply to the on-demand file systems.

System reconfiguration

Like our approach, the configuration of the system can be modified to improve I/O. There are several basic methods. A Dynamic Remote Scratch [22] implementation was developed to create an on-demand block device and use it with local SSDs as a LVM [23] device. Another software based solution is the RAMDISK Storage Accelerator [24]. It introduces a additional cache layer into HPC systems. Our approach also fits into this category.

2.2 Job walltime prediction

Batch schedulers are responsible for the resource planing and allocate the nodes to a job [25]. One of the factors of this resource planning is based on wall time estimates, given by the user. It is a well known problem that the user provided estimates are far from optimal. Enhanced predictions of HPC job wall time can be used to improve the scheduling performance [26]. With exact information about the run time of a job, the scheduler can predict more accurately when sufficient resources are available to start queued jobs [27]. However, the user requested wall time is not close to the real used wall time. Gibbons [28, 26], and Downey [27] use historical workloads to predict the wall times of parallel applications. They predict wall times based on templates. These templates are created by analyzing previously collected metadata and grouped according to similarities. However, both approaches are restricted to simple definitions.
In the recent years, the machine learning algorithms are used to predict resource consumption in several studies [31, 32, 33, 34, 35, 36].
However, all of the above mentioned studies do not try to evaluate the accuracy of the node allocation predictions. Most of the publications focus on observing the utilization of the HPC system and the reliability of the scheduler estimated job start times. In our work we focus on the node allocation prediction and how good wall time estimates have to be. This directly affects, whether a cross-node, ad-hoc, independent parallel FS can be deployed and data can be pre-staged, or not.

2.3 Machine learning

Machine learning (ML) is about knowledge retrieval from data. It can also be understood as statistical learning and predictive analytics. In general, machine learning is a method to learn from a set of samples with a target value and use the learned data to predict target values from unknown samples. For our evaluation, we use a supervised machine learning approach [39].
In our evaluation, the AUTOML library auto-sklearn[40] (based on scikit-learn[41, 42]) is used to automate the complex work of machine learning optimization. In a

classical ML process, different models and systems are explored until the best is chosen and auto-sklearn automatizes this process.

3 Deployment on-demand file system

Usually HPC systems use a *batch system*, such as SLURM [43], MOAB [44], or LSF [45]. The batch system manages the resources of the cluster and starts the user jobs on allocated nodes. At the start of the job, a prologue script may be started on one or all allocated nodes and, if necessary, an epilogue script at the end of a job (see Figure 1). These scripts are used to clean, prepare, or test the full functionality of the

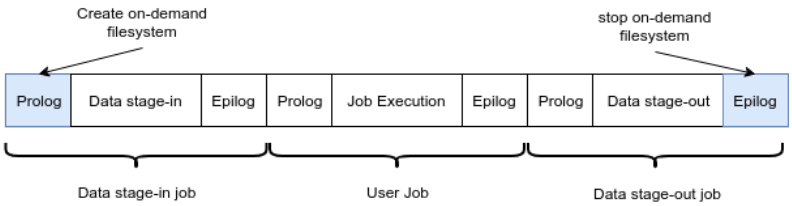


Fig. 1: Job flow for creating an on-demand file system

nodes. We modified these scripts to start the on-demand file system upon request. During job submission a user can request an on-demand file system for the job. This solution has minimal impact on the HPC system operation. Users without the need for an on-demand file system are not affected.

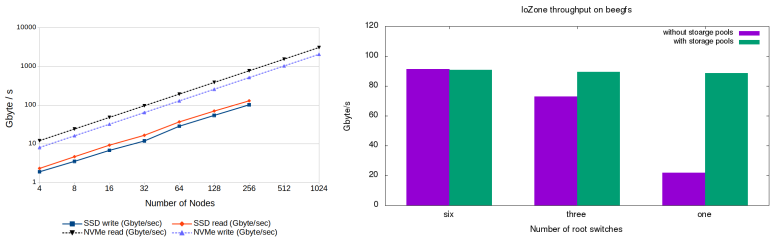
3.1 Benchmarks

As initial benchmarks we tested the startup time of the on-demand file system and used the „iozone“ benchmark for a throughput test. The Startup and shutdown times are shown in Table 1. The delivered tools in the BeeOND package have a serial part

Table 1: 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/s)	2.79	6.74	10.83	28.37	54.06	129.95

during initialization. After optimizing these regions we were able to start BeeGFS within 60 seconds on 512 nodes.
In Fig. 2a we show the IoZone [46] benchmark to measure the read and write throughput of the on-demand file system (solid line). The Figure show that performance increases linearly with the number of used compute nodes. The limiting factor here is the aggregate throughout of the used SSDs. A small throughput variation can be observed due to normal performance scattering of SSDs [47]. The dotted line indicates the theoretical throughput with NVMe devices. At assumed speeds with $[\,3500]MB/s$ read and $[\,2000]MB/s$ write performance for todays common PCIe x4 NVM devices [48].



(a) IoZone Throughput (b) Disabled root switch

Fig. 2

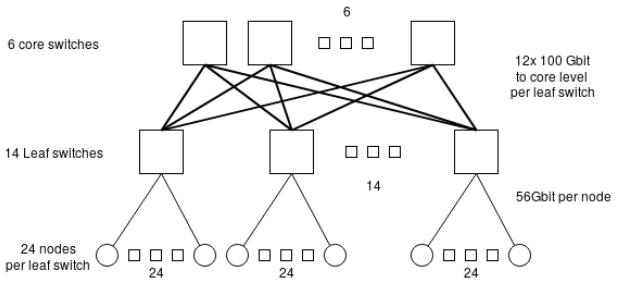


Fig. 3: Scheme of the fabric topology (small island)

In a further test, we evaluated the storage pooling feature of BeeGFS [5]. We created a storage pool for each leaf switch(see Fig. 3). In other words, when writing to a storage pool, the data is distributed via the stripe count and chunk size, but remains within the storage pool and thus on a switch. Only the communication with the meta data server is forwarded across the core switches. Figure 2b shows the write throughput for three scenarios. Each scenario uses a different number of core switches with six being the full expansion level. In the first experiment, with all six

core switches, there is only a minimal performance loss, which indicates a small overhead when using storage pools. In the second case we turned three switches off, and in the last case we turned off five switches. With reduced number of core switches the throughput drops due to the reduced network capacity. If the topology is taken into account, and storage pools are created accordingly, it is possible to achieve the same performance as with the full expansion.

3.2 Conclusion

Adding the on-demand file system functionality to an HPC system is easy. There is no need to change the operating model. An on-demand fs is only started for jobs when it is actually requested. While the startup times might be acceptable on smaller HPC systems, they are not feasible at large scales. What exactly is acceptable, however, depends on several factors. A few minutes start-up time might be acceptable if the jobs runs for a day, but waiting a hour for the on-demand fs when the job is running not much longer might be not.
Various observations show that with this approach the network is no longer the bottleneck. Since the fabric of an HPC system has a high bisection bandwidth, there is enough bandwidth left for an on-demand fs. However, if the network is designed somewhat weaker, enormous bandwidths can still be achieved with taking the topology into consideration.

4 Walltime prediction

An investigation whether data can be pre-staged also belongs to the tasks of this project. One of the challenges is to know which nodes are going to be allocated to a queued job. The HPC scheduler predicts these nodes based on the user given wall times. Therefore, we have decided to evaluate whether there is an easy way to automatically predict such wall time. Our proposed approach for wall time prediction is to train an individual model for every user with methods from the machine learning domain. As historical data, we used workloads from two of the HPC-systems at the Karlsruhe Institute for Technology / Steinbuch Centre for Computing [49], the ForHLR I + II [50] [51] clusters. We used Automatic machine learning (AUTOML) to pre-process the input data and selecting the correct model including the optimization of hyperparameters. In this work, the auto ML library auto-sklearn [40] is used. It is based on scikit-learn [41], [42].
Figure 4 shows the R^2 score for models of the users on ForHLR I+II with 30 min AUTOML. A concentration of the points in the upper right corner indicates a higher number of good models for the training and test data. A more descriptive illustration of the results are given in Figure 5 for the ForHLR II. Here the median absolute error is compared between the AUTOML, the default linear regression, and the user given

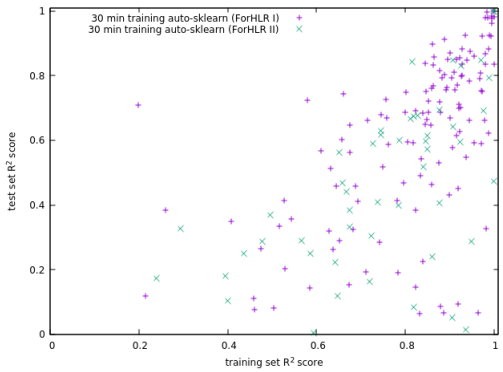


Fig. 4: X-Axis R^2 score on training samples, Y-Axis R^2 score on test samples for ForHLR I+II with 30 min AUTOML

wall time prediction. On the ForHLR II cluster 50% of the prediction have a smaller

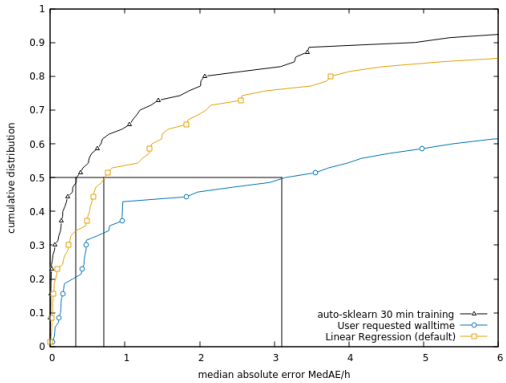


Fig. 5: Y-Axis Cumulative distribution, X-Axis Median absolute error ForHLR II.

median absolute error of around 21 min, 43 min, 186 min for the AUTOML model, the linear regression model, and the user prediction, respectively.

4.1 Node Prediction

Predicting the run times of jobs is only aspect of the challenge. However, the decisive factor is the prediction of which nodes will be allocated to a job. In this following investigation, we have determined the influence of the wall-time on the node

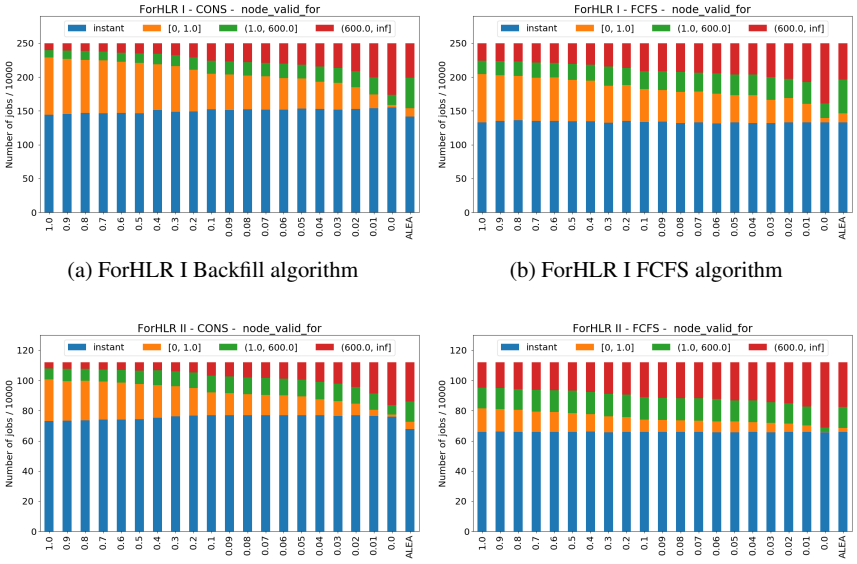
prediction. Therefore the ALEA Simulator [52] has been extended to simulate the time of the node list.
The main goal of our experiments was to identify how good or bad are node allocation predictions subject to variously accurate job wall time estimates. Therefore, we have conducted several experiments where job run time estimates were subsequently improved, starting with very imprecise wall time estimates as provided by real users of the system and continuing to fully accurate job run time estimates. For this purpose, we introduce $\tilde{T}_{\text{Req}}$, the "refined" requested wall time,

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

where T_{Req} is the user requested wall time and T_{Run} is the run time of the job. Each job in the workload is then adjusted by the same λ, effectively simulating different precision of provided wall time estimates.
The simulation of the workloads is shown in Figure 6, each bar represents a simulation with a different λ value. The last bar, labeled "Alea", is the simulation with Alea's built-in wall time prediction. Each row shows the simulation of a specific workload with two different scheduling algorithms. A conservative back-filling algorithm (left column) and a simple FCFS algorithm (right column). The bars are categorized into four groups based on the T_{NAP}. The blue part represent the jobs that are started immediately after the job is submitted(instant). The orange part represent queued jobs with a T_{NAP} between 0 and 1. Jobs with a T_{NAP} from one second up to 10 minutes are represented by the green part. Red indicates a T_{NAP} value for more than 10 minutes (long term prediction). The class of jobs with long-term predictions (red) is in our focus. This long-term predictions increases significantly only at very small $\lambda \leq 0.1$ which already proves that very good run time estimates are needed.
The results in Figure 6i-j show a high rate of jobs that are started immediately after the job submission. There are several reasons for this. First, the workload time-frame is almost two years and various maintenance slots have not been simulated. During the simulation, the queue is processed normally during maintenance time. Second, a part of the nodes has been separated for a limited time for various campaigns. The consumption of the campaigns is not included in the workloads.

4.2 Conclusion

Two different investigations were carried out here. At the first one we showed how you can achieve good walltime predictions with very simple means. We only used general meta-data and trained an individual model for each user. The results are very remarkable, considering that hardly any manual optimizations were performed on the models.
The second part of the work examined how good the predictions should be for our approach. The results are very sobering. It turned out that even with almost perfect



(a) ForHLR I Backfill algorithm

(b) ForHLR I FCFS algorithm

(c) ForHLR II Backfill algorithm

(d) ForHLR II FCFS algorithm

Fig. 6

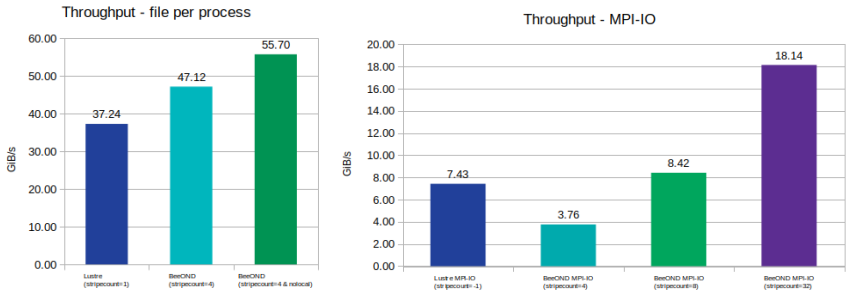
predictions, there is still a lot of uncertainty. If an FCFS algorithm is used, the situation looks a bit better, but this would have a negative effect on the load.
It has therefore been concluded that further work is needed here. In this case, a modification must be made to the operational processes of the scheduler. We have achieved this by developing a plug-in (On-demand burst buffer plugin) for the SLURM scheduler. If required, this plugin starts an on-demand plugin and transfers the data on the temporary fs. The challenge with the unknown node list is solved with reservations by this plugin.

5 Scientific applications

We have evaluated several applications regarding to on-demand file systems. We have selected application which either generated a very high load on our system or the I/O part is identified as a bottleneck.

5.1 super_sph

We evaluated the application super_sph (”Simulation for Smoothed Particle Hydrodynamics”) [54] which is developed at „Institut für Strömungsmaschinen” @KIT. The software scales up to 15000 Cores and 10^9 particles. The first implementation of the software created a file per process and required data-gathering as post-processing. A new implementation is now writing directly to time steps using MPI-IO which makes the data-gathering process unnecessary. From our observation – file per process method is causing heavy load on the PFS. Using MPI-IO is slower but has less impact on global PFS. Figure 7 show the results of super_sph when writing directly to the global filesystem (Lustre) and to an on-demand created filesystem (BeeOND/BeeGFS). For the benchmark we used 256 Nodes. While using the simple



(a) Write troughput with file-per-process method

(b) Write troughput with MPI-IO method

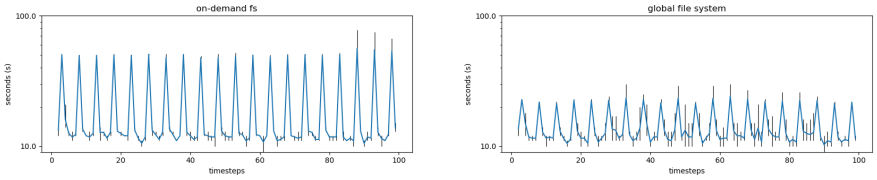
Fig. 7: Write benchmark with super_sph.

file-per-process method we gain a small performance increase. When using MPI-IO is important to choose the right parameters for the file-system. If the chunksize is not adequate the performance loss is tremendous.

5.2 OpenFOAM

Another application we investigated is OpenFOAM [55]. OpenFOAM is a toolkit for computational fluid dynamics (CFD) written in C++. It is a widely used open-source fluid dynamics code [56] for engineering applications. OpenFOAM offers tools for many areas of research, like heat transfer, reacting flows, or multi-phase flows. Investigation of I/O performance has been performed with a OpenFOAM case, using OpenFOAM v1712 and a custom solver developed for detailed combustion simulations [57, 58]:

Use case: A production run simulation of an experimentally investigated burner of laboratory scale [59]. The focus of this setup lies on simulating the burner flame in great detail, including all intermediate chemical species that are formed during combustion. Due to the physical complexity of the flame, the computational domain consists of 150 million cells. The simulation is typically run on 5 000–28 000 CPU cores [60]. In this work, the case has been run on 240 nodes or 4800 processes, leading to the creation of 95 files per process or half a million files in total, with about 0.2 MB per file or 25 MB per process or 120 GB in total. For previous runs with 28 800 CPU cores, the number of files which are written at the same time increases to 2.7 million.



(a) execution time per timestep using on-demand fs

(b) execution time per timestep using global fs

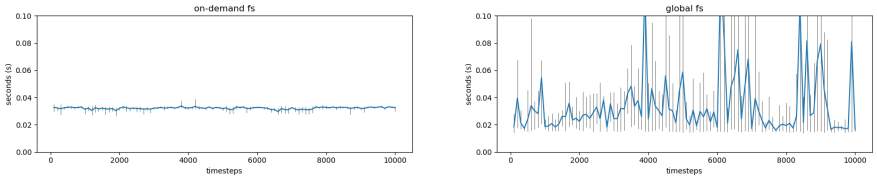
Fig. 8

Figure 8 show results for our scenario. We run the application five times on 240 Nodes. The Blue line is the average execution time of the time steps. The black bars represent the min/max values for the time steps. The high spikes in both cases occur when the application is writing his intermediate results. Fig. 8a show the results when OpenFOAM is writing to the on-demand fs and 8b when using the global file system. The spikes are higher when using the on-demand, this is explained by the fact that the data is stored on the nodes. The On-demand fs has to share the resources with the application on the compute nodes. But what also can be seen, is the deviation(black bars) is much higher when using the global file system. Here the advantage is the dedicated bandwidth and meta data performance which is offered by a private on-demand file system.

5.3 NASTjA

The NAStJA framework [61] is another application we examined with the use of on-demand file systems. NAStJA is a massively parallel stencil code solver. It is designed for simulations in several scientific domains. The framework provides several modules for writing out the results. For our purpose, we configured a single file per

block per time step. A basic phase-field model of crystal solidification was chosen for the simulation.
Use case: A run with 4800 parallel NAStJA processes is distributed to 4800 working nodes with 20 processes each. A particular node is reserved for the BeeGFS processes. We choose 4 800 blocks, i.e., one block per core, of the size of [1]MB each and write out every 20th time-step of a total of 100 000 time-steps.



(a) execution time per timestep using on-demand fs

(b) execution time per timestep using global fs

Fig. 9

Figure 9 show results for the NAStJA scenario. We run the application five times on 240 Nodes. The Blue line is the average execution time of the time steps. The black bars represent the min/max values for the time-steps. Here we see another result in contrast to OpenFOAM use case. Fig. 9a show the results when NAStJA is writing to the on-demand fs and 9b when using the global file system. When using the on-demand fs there is almost no deviation, while using the global fs, there are high spikes during the whole simulation.

5.4 Data staging

We also considered the case of copying data back to the PFS while the application is running. For this purpose, we used different NAStJA simulations on 23 nodes with 16, 19, and 20 cores per node for the application. The remaining resources on the compute nodes are then available for the on- demand file system and data staging. Of course, when using all 20 cores for the application there are no physical cores left, but we wanted to evaluate how much the interference is. For reference, each simulation is performed without data staging. To stage the data, during the NAStJA execution, we used the parallel copy tool dcp [62]. As the staging workflow, we considered two cases: a single node with four dcp processes, and a case with one dcp process per compute node. In the case of a single node the MDS server node of the on-demand file system was used. Figure 10 show the average execution time per time-step of five runs in our different scenarios. With 16 cores for the application, from available 20 cores, the run times are similar whether the run was executed with

or without data staging. When using 19 or 20 cores, the application is slowed down when the copy is executed with one process per node. At the beginning, the slowdown is significant (orange line) due to the high amount of metadata operations. In this case, a portion of the data is indexed on every node and this is causing interference with the application. When using only the MDS-server to copy the generated data (green line) the indexing is done only on the node with the MDS-server. If there are enough free resources on the compute nodes, the data can be staged-out without slowing down the application. Staging the data back afterwards with dcp needs approximately 30 seconds, and only 6 seconds for the pure data transfer. This raises the question of whether it makes sense to copy the data back during the simulation.

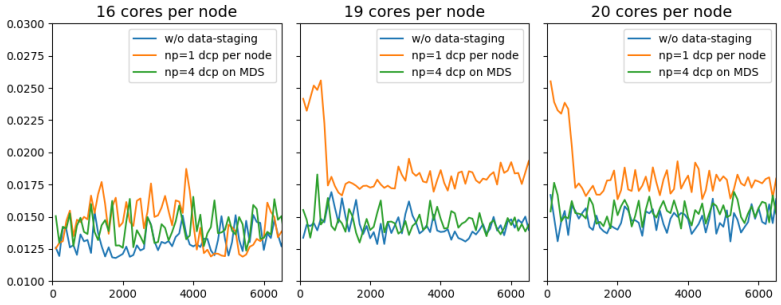


Fig. 10: Execution time per time-step. Different scenarios w/o data staging.

5.5 Conclusion

The results with real applications and use cases are already very good in the early phase. The use of on-demand file system immediately reduces the load on the global file system. This is of great importance for the shared HPC system and means a much more stable operation with less interference between the jobs. The impact of an on-demand fs to the application is minimal. However, we have only tested a handful of applications and use cases to see if an on-demand fs becomes a disadvantage. Also the results for data-staging are promising, depending on whether you have much or little time to move the data, there are ways to choose the right method. Also the influence on the application is controllable.

6 Summary

On-demand file systems is easy adaptable into a HPC System. It immediately reduces the load on the global file systems. Startup and shutdown times are acceptable only for long running jobs. For very short running jobs it might be use senseless, but experience shows that large scale jobs usually request longer wall times. However, many more factors have to be taken into account here to enable a reasonable and fast use in a wide range. There are also many factors to consider during deployment so that a user is not overwhelmed, e.g., setting the right strip-count und chunk-size parameters.
It turned out that pre-staging data to the compute nodes is not possible with the unreliable allocation prediction of the scheduler. Here a modification is needed to cope with the issue of the unknown node list. A Plugin has been developed which solves this issue, by using reservations. The plugin extends the use of the built-in burst buffer concept and creates aa on-demand fs and moves the required data to the temporary fs.
The trend in the HPC environment clearly shows that faster solid state disks keep coming into the compute nodes. With these, the advantages of on-demand file systems on the compute nodes should be even more significant.

Acknowledgments

The project ADA-FS is funded by the DFG Priority Program ,,Software for exascale computing" (SPPEXA, SPP 1648), which is gratefully acknowledged. This work was supported by the Helmholtz Association of German Research Centres (HGF) and the Karlsruhe Institue of Technology. This work was performed on the computational resource ForHLR II with the acronym ADA-FS funded by the Ministry of Science, Research and the Arts Baden-Württemberg and DFG ("Deutsche Forschungsgemeinschaft"). We would like to thank the operation team of the ForHLR II cluster, which allowed us to adapt operational areas of the system to our needs.

References

1. S. Microsystems, "LUSTRE^{TM} FILE SYSTEM High-Performance Storage Architecture and Scalable Cluster File System," http://www.csee.ogi.edu/~zak/cs506-pslc/lustrefilesystem.pdf, 2007, accessed: September 05 2016.
2. F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02. Berkeley, CA, USA: USENIX Association, 2002.
3. J. Heichler, "An introduction to BeeGFS," http://www.beegfs.com/docs/Introduction_to_BeeGFS_by_ThinkParQ.pdf, 2014, accessed: September 6, 2016.

4. O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu, "On the root causes of cross-application I/O interference in HPC storage systems," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 750–759.
5. BeeGFS, "BeeGFS Storage Pool," https://www.beegfs.io/wiki/StoragePools, 2018, accessed: August 18 2018.
6. IBM, "GPFS - highly available write cache (hawc)," 2018. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/STXKQY_5.0.0/com.ibm.spectrum.scale.v5r00.doc/bl1adv_hawc.htm
7. R. Mohr, M. J. Brim, S. Oral, and A. Dilger, "Evaluating progressive file layouts for lustre."
8. J. Xing, J. Xiong, N. Sun, and J. Ma, "Adaptive and scalable metadata management to support a trillion files," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 26:1–26:11. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654086
9. S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/O performance challenges at leadership scale," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Nov 2009, pp. 1–12.
10. N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. IEEE, 2012, pp. 1–11.
11. T. Wang, K. Mohror, A. Moody, K. Sato, and W. Yu, "An ephemeral burst-buffer file system for scientific applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 69.
12. DataDirect Networks, "IME - Flash native cache," https://www.ddn.com/products/ime-flash-native-data-cache/, 2018.
13. CRAY, "Cray® DataWarp™ Applications I/O Accelerator," https://www.cray.com/datawarp, 2018.
14. J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate, "Plfs: a checkpoint filesystem for parallel applications," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, p. 21.
15. BeeGFS, "BeeOND™: BeeGFS On Demand," http://www.beegfs.io/wiki/BeeOND, 2018, accessed: August 18 2018.
16. R. Thakur, W. Gropp, and E. Lusk, "On implementing MPI-IO portably and with high performance," in *Proceedings of the sixth workshop on I/O in parallel and distributed systems*. ACM, 1999, pp. 23–32.
17. ——, "Data sieving and collective I/O in ROMIO," in *Frontiers of Massively Parallel Computation, 1999. Frontiers' 99. The Seventh Symposium on the*. IEEE, 1999, pp. 182–189.
18. M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the HDF5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. ACM, 2011, pp. 36–47.
19. R. Rew and G. Davis, "Netcdf: an interface for scientific data access," *IEEE computer graphics and applications*, vol. 10, no. 4, pp. 76–82, 1990.
20. J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*. ACM, 2008, pp. 15–24.
21. W. Frings, "SIONlib," http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SIONlib/_node.html, 2009.
22. M. Neuer, J. Salk, H. Berger, E. Focht, C. Mosch, K. Siegmund, V. Kushnarenko, S. Kombrink, and S. Wesner, "Motivation and implementation of a dynamic remote storage system for I/O demanding HPC applications," in *International Conference on High Performance Computing*. Springer, 2016, pp. 616–626.
23. D. Teigland and H. Mauelshagen, "Volume managers in linux." in *USENIX Annual Technical Conference, FREENIX Track*, 2001, pp. 185–197.

24. T. Wickberg and C. Carothers, "The RAMDISK storage accelerator: A method of accelerating I/O performance on HPC systems using RAMDISKs," in *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, ser. ROSS '12. New York, NY, USA: ACM, 2012, pp. 5:1–5:8. [Online]. Available: http://doi.acm.org/10.1145/2318916.2318922
25. M. Hovestadt, O. Kao, A. Keller, and A. Streit, "Scheduling in hpc resource management systems: Queuing vs. planning," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1–20.
26. R. Gibbons, "A historical application profiler for use by parallel schedulers," in *Job scheduling strategies for parallel processing*. Springer, 1997, pp. 58–77.
27. A. B. Downey, "Predicting queue times on space-sharing parallel computers," in *Parallel Processing Symposium, 1997. Proceedings., 11th International*. IEEE, 1997, pp. 209–218.
28. R. Gibbons, "A historical profiler for use by parallel schedulers," *Master's thesis, University of Toronto*, 1997.
29. W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," in *Job Scheduling Strategies for Parallel Processing*. Springer, 1998, pp. 122–142.
30. W. Smith, V. Taylor, and I. Foster, "Using run-time predictions to estimate queue wait times and improve scheduler performance," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1999, pp. 202–219.
31. A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 495–504.
32. N. H. Kapadia and J. A. Fortes, "On the design of a demand-based network-computing system: The purdue university network-computing hubs," in *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*. IEEE, 1998, pp. 71–80.
33. A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.
34. F. Nadeem and T. Fahringer, "Using templates to predict execution time of scientific workflow applications in the grid," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2009, pp. 316–323.
35. W. Smith, "Prediction services for distributed computing," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–10.
36. D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, 2007.
37. Xsede. https://www.xsede.org/.
38. Karnak start/wait time predictions. http://karnak.xsede.org/karnak/index.html.
39. M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2012.
40. M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf
41. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
42. L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project,"

in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.
43. "Slurm - schedmd," http://www.schedmd.com.
44. "Adaptive Computing," http://www.adaptivecomputing.com.
45. "IBM - platform computing," http://www.ibm.com/systems/platformcomputing/products/lsf/.
46. D. Capps and W. Norcott, "Iozone filesystem benchmark," 2008. [Online]. Available: http://iozone.org/
47. E. Kim, "SSD performance-a primer: An introduction to solid state drive perfromance, evaluation and test," Tech. rep., Storage Networking Industry Association, Tech. Rep., 2013.
48. J. J. Hung, K. Bu, Z. L. Sun, J. T. Diao, and J. B. Liu, "PCI express-based NVMe solid state disk," in *Applied Mechanics and Materials*, vol. 464. Trans Tech Publ, 2014, pp. 365–368.
49. Steinbuch Center for Computing, "Scc," http://www.scc.kit.edu, 2016, accessed: August 16, 2016.
50. "Forschungshochleistungsrechner ForHLR 1," www.scc.kit.edu/dienste/forhlr1.php, 2018.
51. "Forschungshochleistungsrechner ForHLR 2," www.scc.kit.edu/dienste/forhlr2.php, 2018.
52. "Alea 4: Job scheduling simulator," February 2019, https://github.com/aleasimulator.
53. M. Soysal, M. Berghoff, and A. Streit, "Analysis of job metadata for enhanced wall time prediction," in *Job Scheduling Strategies for Parallel Processing*, 2018.
54. S. Braun, R. Koch, and H.-J. Bauer, "Smoothed particle hydrodynamics for numerical predictions of primary atomization," vol. 15, no. 1, pp. 56–60, 2017.
55. OpenCFD, *OpenFOAM: The Open Source CFD Toolbox. User Guide Version 1.4, OpenCFD Limited*. Reading UK, Apr. 2007.
56. "The OpenFOAM foundation," https://openfoam.org/, 2018.
57. T. Zirwes, F. Zhang, J. Denev, P. Habisreuther, and H. Bockhorn, "Improved vectorization for efficient chemistry computations in openfoam for large scale combustion simulations," in *High Performance Computing in Science and Engineering '17*, W. Nagel, D. Kröner, and M. Resch, Eds. Springer, 2018.
58. T. Zirwes, F. Zhang, T. Häber, and H. Bockhorn, "Ignition of combustible mixtures by hot particles at varying relative speeds," *Combustion Science and Technology*, vol. 0, no. 0, pp. 1–18, 2018. [Online]. Available: https://doi.org/10.1080/00102202.2018.1435530
59. R. Barlow, S. Meares, G. Magnotti, H. Cutcher, and A. Masri, "Local extinction and near-field structure in piloted turbulent CH4/air jet flames with inhomogeneous inlets," *Combust. Flame*, vol. 162, no. 10, pp. 3516–3540, 2015.
60. T. Zirwes, F. Zhang, J. Denev, P. Habisreuther, and H. Bockhorn, "Automated code generation for maximizing performance of detailed chemistry calculations in OpenFOAM," in *High Performance Computing in Science and Engineering '17*, W. Nagel, D. Kröner, and M. Resch, Eds. Springer, 2017, pp. 189–204.
61. M. Berghoff, I. Kondov, and J. Hötzer, "Massively parallel stencil code solver with autonomous adaptive block distribution," *IEEE Transactions on Parallel and Distributed Systems*, 2018. [Online]. Available: http://doi.acm.org/10.1109/TPDS.2018.2819672
62. D. Sikich, G. Di Natale, M. LeGendre, and A. Moody, "mpifileutils: A parallel and distributed toolset for managing large datasets," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2017.