Ad-Hoc File Systems At Extreme Scales

Mehmet Soysal and 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.

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

M. Soysal (✉) · A. Streit
Steinbuch Centre for Computing, Karlsruhe Institute of Technology,
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
e-mail: mehmet.soysal@kit.edu

A. Streit
e-mail: achim.streit@kit.edu

W. E. Nagel et al. (eds.), *High Performance Computing in Science and Engineering '19*,
https://doi.org/10.1007/978-3-030-66792-4_36

possible factors a user would have to consider. Influences from the back-end 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. The results have been published [5–7] and we are showing in this paper a overview of our work. 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 Chap. 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 Chap. 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 Chap. 5. First we start with the related work in the next Chap. 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 [8] 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)[9]. 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) [10] 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 [11, 12], 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 [13].

Libraries

There is a large number of libraries available for improving I/O behavior of an application. Middleware libaries, such as MPI-IO [14], help to improve usage of parallel storage systems, e.g. collective I/O [15]. High-level libraries, such as HDF5 [16], NETCDF [17] or ADIOS [18], 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 [19] implementation was developed to create an on-demand block device and use it with local SSDs as a LVM [20] device. Another software based solution is the RAMDISK Storage Accelerator [21]. 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 [22]. 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. 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 [23]. However, the user requested wall time is not close to the real used wall time. Gibbons [24, 25], and Downey [23] 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 [26–31].

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 file system 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 [32].

In our evaluation, the AUTOML library auto-sklearn [33] (based on scikit-learn [34, 35]) 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 [36], MOAB [37], or LSF [38]. 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 Fig. 1). These scripts are used to clean, prepare, or test the full functionality of the

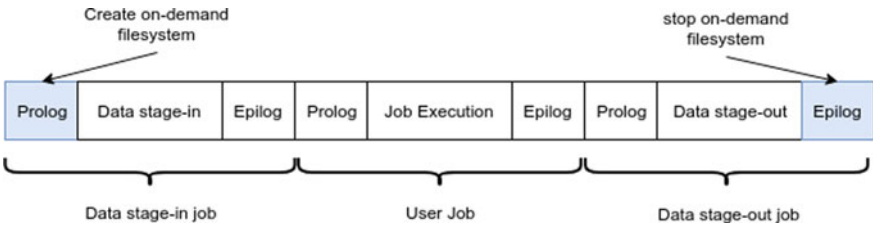


Fig. 1 Job flow for creating an on-demand file system

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

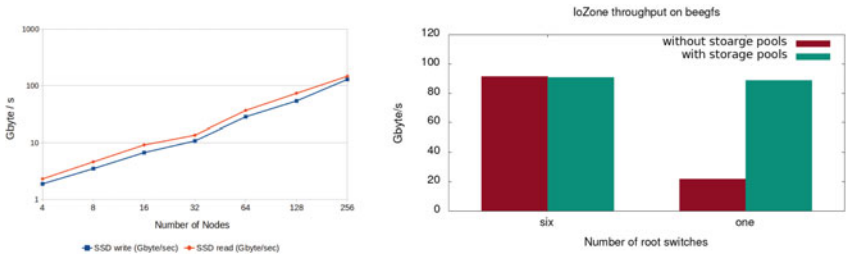
nodes. We modified these scripts to start the on-demand file system upon request. During job submission a user can request an on-demand file system for the job. This solution has minimal impact on the HPC system operation. Users without the need for an on-demand file system are not affected.

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 during initialization. After optimizing these regions we were able to start BeeGFS within 60 s on 512 nodes.

In Fig. 2a we show the IoZone [39] 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 deviation can be observed due to performance scattering of SSDs [40].

In a further test, we evaluated the storage pooling feature of BeeGFS [8]. 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 physical within a switch. Only the communication with the meta data server is forwarded across the core switches. Figure 2b shows the write throughput for the scenarios. 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



(a) IoZone Throughput (b) Disabled root switch

Fig. 2 **a** IoZone Throughput **b** Disabled root switch

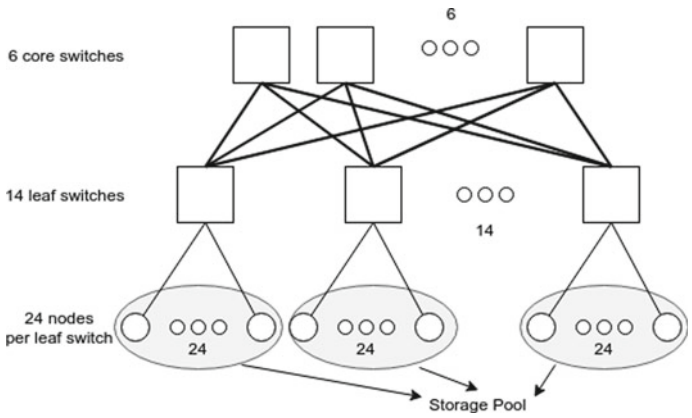


Fig. 3 Scheme of the fabric topology (small island), with a storage pool per leaf switch

second case we turned five switches off. With reduced number of core switches, the throughput drops due to the reduced network capacity. If storage pools are created accordingly to the leaf switches, it is possible to achieve the same performance.

3.2 Conclusion

Adding on-demand file system functionality to an HPC system is easy. There is no need to change the operating model. An on-demand file system is only started if it is actually requested. Startup times might be acceptable on smaller HPC systems but, they are not feasible at large scales. But what is exactly acceptable depends on several factors. For example, a few minutes start-up time may be acceptable if the jobs run for a day. But waiting an hour for the file system to start when the job itself isn't running much longer, doesn't make much sense.

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 throughput 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 serveral workloads from two of the HPC-systems at the Karlsruhe Institute for Technology/Steinbuch Centre for Computing [41], the ForHLR I + II [42, 43] 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 [33] is used. It is based on scikit-learn [34, 35].

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 Fig. 5 for the ForHLR II. Here the median absolute error is compared between the AUTOML, the default linear regression, and the user given wall time prediction. On the ForHLR II cluster 50% of the prediction have a smaller 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 Conclusion

We showed that we can achieve good walltime predictions with very simple methods. 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.

But we have shown in a futher investigation that even with almost perfect job wall time estimates the node allocation can't be predicted in a sufficient manner [7].

It has therefore been decided 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

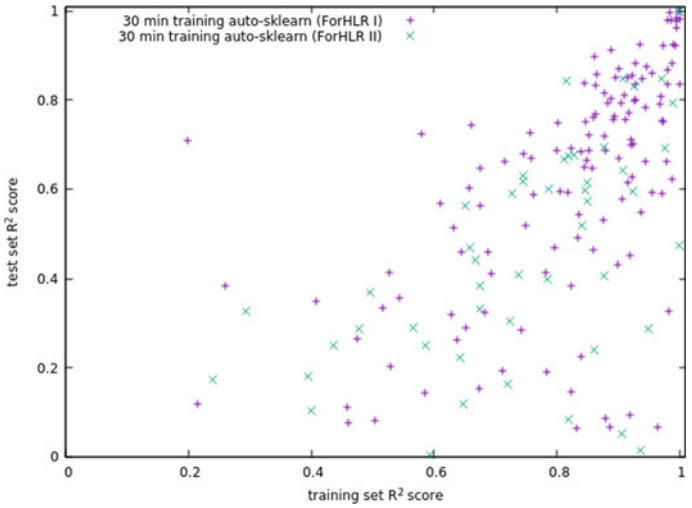


Fig. 4 X-Axis R^2 score on training samples, Y-Axis R^2 score on test samples for ForHLR I+II with 30 min AUTOML

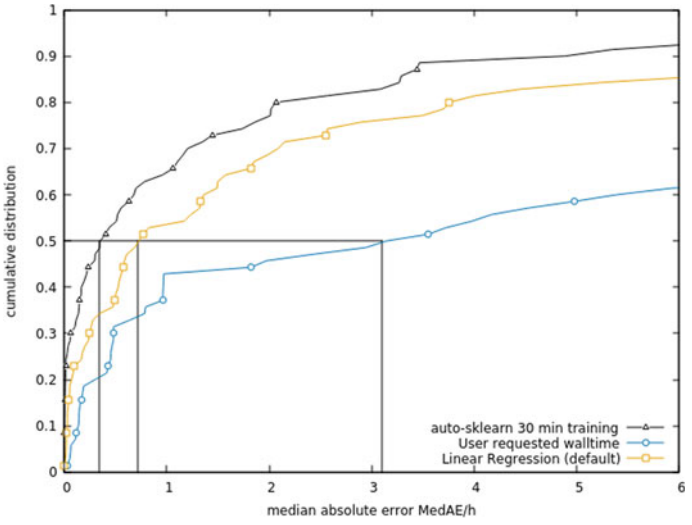
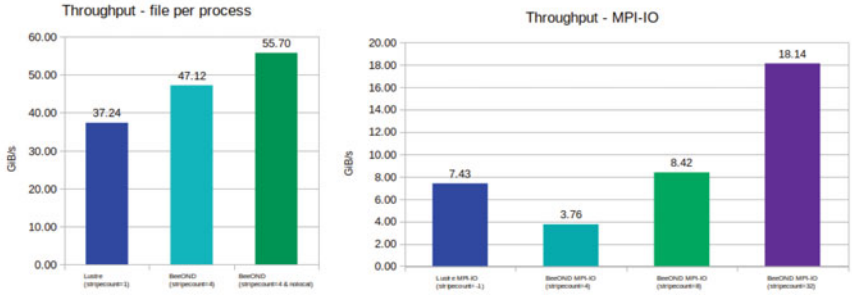


Fig. 5 Y-Axis Cumulative distribution, X-Axis Median absolute error ForHLR II

temporary fs. The challenge with the unknown node list is solved with reservations by this plugin.



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

(b) Write troughput with MPI-IO method

Fig. 6 Write benchmark with super_sph

5 Scientific Applications

We have evaluated several applications regarding to on-demand file systems. We selected applications which either generate a very high load on our system or the I/O part is identified as a bottleneck. In this paper we present only a very brief overview, other results are already published [44].

5.1 Super_sph

We evaluated the application super_sph ("Simulation for Smoothed Particle Hydrodynamics") [45] 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 6 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 file-per-process method we gain a small performance increase. When using MPI-IO it is important to hoose the right parameters for the file-system. If the chunksize is not well chosen the performance loss is tremendous.

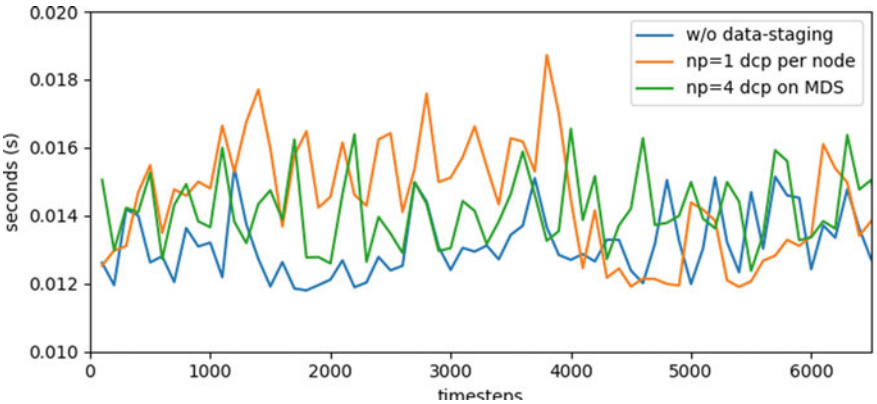


Fig. 7 Execution time per time-step. Different scenarios w/o data staging

5.2 Data Staging

We also considered the case of copying data back to the PFS while the application is running. The results are already published [44] and here we show a short summary. For this purpose, we used different NAStJA simulations on 23 nodes. The parallel copy tool dcp [46] was used to stage data. Figure 7 show the average execution time per time-step of five runs. 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. If there are enough free resources on the compute nodes, the data can be staged-out without slowing down the application.

5.3 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 file system to the application is minimal. However, we have only tested few applications and use cases to see if an on-demand file system 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.

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 senseless, but experience shows that large scale jobs usually request longer wall times. However, many more factors have to be taken into account 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 and 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 an on-demand file system and moves the required data to the temporary file system.

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.

Acknowledgements 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™ FILE SYSTEM High-Performance Storage Architecture and Scalable Cluster File System (2007), http://www.csee.ogi.edu/zak/cs506-pslc/lustrefilesystem.pdf. Accessed 05 Sept 2016
2. F. Schmuck, 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 (2014), http://www.beegfs.com/docs/Introduction_to_BeeGFS_by_ThinkParQ.pdf. Accessed 6 Sept 2016
4. O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, G. Antoniu, On the root causes of cross-application I, O interference in HPC storage systems, in *IEEE International on Parallel and Distributed Processing Symposium* (IEEE 2016), pp. 750–759
5. M. Soysal, M. Berghoff, A. Streit, Analysis of job metadata for enhanced wall time prediction, in *Job Scheduling Strategies for Parallel Processing* (2018)
6. M. Soysal, M. Berghoff, A. Streit, *Analysis of job metadata for enhanced wall time prediction, in Job Scheduling Strategies for Parallel Processing* (Springer International Publishing, Cham, 2019), pp. 1–14
7. M. Soysal, M. Berghoff, D. Klusáček, A. Streit, On the quality of wall time estimates for resource allocation prediction, in *Proceedings of the 48th International Conference on Parallel*

Processing: Workshops, ser. ICPP 2019 (ACM, New York, NY, USA, 2019) vol 23, pp. 1–23, 8. https://doi.org/10.1145/3339186.3339204
8. BeeGFS, BeeGFS Storage Pool (2018), https://www.beegfs.io/wiki/StoragePools. Accessed 18 Aug 2018
9. IBM, GPFS—highly available write cache (hawc) (2018), https://www.ibm.com/support/knowledgecenter/en/STXKQY_5.0.0/com.ibm.spectrum.scale.v5r00.doc/bl1adv_hawc.htm
10. R. Mohr, M.J. Brim, S. Oral, A. Dilger, Evaluating progressive file layouts for lustre
11. J. Xing, J. Xiong, N. Sun, 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 (ACM, New York, NY, USA, 2009) pp. 26:1–26:11. https://doi.org/10.1145/1654059.1654086
12. S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, W. Allcock, I/O performance challenges at leadership scale, in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*(2009), pp. 1–12
13. N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, C. Maltzahn, On the role of burst buffers in leadership-class storage systems, in *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, (IEEE, 2012), pp. 1–11
14. R. Thakur, W. Gropp, 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
15. R. Thakur, W. Gropp, E. Lusk, Data sieving and collective I, O in ROMIO, in *The Seventh Symposium on the Frontiers of Massively Parallel Computation, Frontiers' 99* (IEEE, 1999), pp. 182–189
16. M. Folk, G. Heber, Q. Koziol, E. Pourmal, 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
17. R. Rew, G. Davis, Netcdf: an interface for scientific data access. IEEE Comput. Graphics Appl. **10**(4), 76–82 (1990)
18. J.F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, 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
19. M. Neuer, J. Salk, H. Berger, E. Focht, C. Mosch, K. Siegmund, V. Kushnarenko, S. Kombrink, 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
20. D. Teigland, H. Mauelshagen, Volume managers in linux, in USENIX Annual Technical Conference. FREENIX Track 185–197 (2001)
21. T. Wickberg, 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 (ACM, New York, NY, USA, 2012), pp. 5:1–5:8. https://doi.org/10.1145/2318916.2318922
22. M. Hovestadt, O. Kao, A. Keller, A. Streit, Scheduling in hpc resource management systems: queuing vs planning, in *Job Scheduling Strategies for Parallel Processing*, ed. by D. Feitelson, L. Rudolph, U. Schwiegelshohn (Springer, Berlin, 2003), pp. 1–20
23. A.B. Downey, Predicting queue times on space-sharing parallel computers, in *Proceedings of the 11th International Parallel Processing Symposium* (IEEE, 1997), pp. 209–218
24. R. Gibbons, A historical profiler for use by parallel schedulers. Master's thesis, University of Toronto, 1997
25. R. Gibbons, A historical application profiler for use by parallel schedulers, in *Job Scheduling Strategies for Parallel Processing* (Springer, 1997), pp. 58–77
26. A. Matsunaga, 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

27. N.H. Kapadia, J.A. Fortes, On the design of a demand-based network-computing system: the purdue university network-computing hubs, in *Proceedings of the Seventh International Symposium on High Performance Distributed Computing* (IEEE, 1998), pp. 71–80
28. A.W. Mu'alem, D.G. Feitelson, Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
29. F. Nadeem, 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
30. W. Smith, Prediction services for distributed computing, in *IEEE International on Parallel and Distributed Processing Symposium, (IPDPS 2007)* (IEEE, 2007), pp. 1–10
31. D. Tsafrir, Y. Etsion, D.G. Feitelson, Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distribut. Syst. **18**(6), (2007)
32. M. Mohri, A. Rostamizadeh, A. Talwalkar, *Foundations of Machine Learning*, (MIT press, 2012)
33. M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in *Advances in Neural Information Processing Systems*, ed. by C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett (Curran Associates, Inc., 2015), pp. 2962–2970. http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf
34. 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, E. Duchesnay, Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
35. 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, 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
36. Slurm-schedmd, http://www.schedmd.com
37. Adaptive Computing, http://www.adaptivecomputing.com
38. IBM—platform computing, http://www.ibm.com/systems/platformcomputing/products/lsf/
39. D. Capps, W. Norcott, Iozone filesystem benchmark (2008), http://iozone.org/
40. E. Kim, SSD performance-a primer: an introduction to solid state drive performance, evaluation and test, Tech. rep. (Storage Networking Industry Association, 2013)
41. Steinbuch Center for Computing, Scc (2016), http://www.scc.kit.edu. Accessed 16 Aug 2016
42. Forschungshochleistungsrechner ForHLR 1 (2018), www.scc.kit.edu/dienste/forhlr1.php
43. Forschungshochleistungsrechner ForHLR 2 (2018), www.scc.kit.edu/dienste/forhlr2.php
44. M. Soysal, M. Berghoff, T. Zirwes, M.A. Vef, S. Oeste, A. Brinkman, W. E. Nagel, A. Streit, Using On-demand File Systems in HPC Environments, *Accepted @ The 2019 International Conference on High Performance Computing and Simulation (HPBench@HPCS)*
45. S. Braun, R. Koch, H.J. Bauer, Smoothed particle hydrodynamics for numerical predictions of primary atomization **15**(1), 56–60 (2017)
46. D. Sikich, G. Di Natale, M. LeGendre, A. Moody, mpifileutils: a parallel and distributed toolset for managing large datasets, Lawrence Livermore National Lab (LLNL) (Livermore, CA, United States, Tech. Rep., 2017)