Could Blobs Fuel Storage-Based Convergence between HPC and Big Data?

Pierre Matri*†, Yevhen Alforov‡, Álvaro Brandon*, Michael Kuhn§, Philip Carns¶, Thomas Ludwig‡

*Universidad Politécnica de Madrid, Madrid, Spain, {pmatri, abrandon}@fi.upm.es
†Inria Rennes Bretagne-Atlantique, Rennes, France, pierre.matri@inria.fr
‡Deutsches Klimarechenzentrum GmbH, Hamburg, Germany, {alforov, ludwig}@dkrz.de
§Universität Hamburg, Hamburg, Germany, michael.kuhn@informatik.uni-hamburg.de
¶Argonne National Laboratory, Lemont, IL, USA, carns@mcs.anl.gov

***Abstract*—The increasingly growing data sets processed on HPC platforms raise major challenges for the underlying storage layer. A promising alternative to POSIX-IO- compliant file systems are simpler blobs (binary large objects), or object storage systems. They offer lower overhead and better performance at the cost of largely unused features such as file hierarchies or permissions. Similarly, blobs are increasingly considered for replacing distributed file systems for big data analytics or as a base for storage abstractions like key-value stores or time-series databases. This growing interest in such object storage on HPC and big data platforms raises the question: Are blobs the right level of abstraction to enable storage-based convergence between HPC and Big Data? In this paper we take a first step towards answering the question by analyzing the applicability of blobs for both platforms.**

I. Introduction

Extracting value from big data involves large-scale computation and complex queries. The scale of the computation and the associated performance requirements increase with the size of the data sets involved. This trend is highlighted by the integration of high-performance computing (HPC) technologies (e.g., GPU computing or fast interconnects) on cloud platforms [1], [2]. Such convergence is especially important for life sciences, climate simulations, computational fluid dynamics, and physics experiments, where the data sets are huge and the computations are intensive [3], [4], [5].

New extreme scale and data-intensive applications raise crucial challenges for HPC. Storage systems for HPC platforms have typically been designed assuming relatively small data sets compared with the scale of the computation required [6]. These challenges have already largely been explored for big data applications (e.g., Hadoop and Spark [7]). Such applications rely on storage systems regularly offering significantly relaxed guarantees and functionality compared with the strict POSIX-compliant file systems widely available on HPC platforms. However, HPC applications usually do not need these POSIX-IO features [8], [9], [10]. Indeed, the libraries commonly used to access the file system, such as MPI-IO, provide relaxed semantics. Therefore, strict semantics at the storage level is often unnecessary for HPC as well.

Big data applications have long moved away from strict POSIX-IO compliance. For example, the Hadoop File System (HDFS) [11] largely used as the storage layer for Hadoop and Spark applications implements neither the full POSIX-IO API nor strict POSIX semantics. Yet, most big data platforms still rely on an underlying file-based storage interface.

Similarly, with regard to HPC platforms, a considerable amount of research has been devoted to moving away from the the strict POSIX-IO compliance provided by Lustre, to adopt more configurable consistency models [12], [13] allowing users to fine-tune the performance of their applications. Yet, the POSIX-IO API, or file system interface, imposes features that hinder the performance of the storage stack, such as a hierarchical namespace or file permissions. While these features are often provided for convenience, they are rarely needed by the applications. We, therefore, argue that the performance cost associated with these features can be avoided by using a flat namespace, as provided by modern blob storage systems like Týr [14] or RADOS [15].

The challenges associated with exascale are strikingly similar on HPC and big data platforms. Blob storage seems to be a strong candidate for future data platforms on both sides. Therefore, the following question arises: Could blobs fuel storage-based convergence between HPC and Big Data by acting as a common, converging storage layer? In this paper we take a first step to answering this question:

- After briefly describing our goals (Section II) and reviewing related work (Section II), **we propose blobs as potential candidates for addressing the converging storage needs of HPC and Big Data** (Section III).
- We leverage a representative set of HPC and Big Data applications to prove that the vast majority of the **I/O calls performed can be covered by current state-of-the-art blob storage systems** (Section IV).

We finally conclude with future work that further demonstrates the relevance of our approach (Section V).

II. Related work and motivation

A. HPC: From strict to relaxed POSIX-IO API and semantics

Increasingly large amounts of data are generated by HPC applications as the result of simulations and large-scale experiments. Thus, storage systems need to provide concurrent access to the data for large numbers of tasks and processes. Such parallel storage operations rely on the usage of a parallel file system (PFS) implementing the POSIX-IO interface as the

storage layer. Typical examples of such file systems used on most HPC platforms are Lustre [16] and OrangeFS [17].

Beyond the POSIX-IO interface lies the POSIX-IO semantics that a fully compliant file systems must implement. This standard has advantages regarding portability, but its inflexibility can cause considerable performance degradation [9]. For example, this standard requires that changes made to a shared file must be visible immediately by all processes. Because an application has no way of telling the file system that POSIX-IO semantics are unnecessary or unwanted, it cannot avoid this performance penalty. For many file systems, these performance issues are noticeable even for small numbers of client processes and straightforward I/O patterns [6], [18], [19]. The issues also affect higher levels of the I/O stack because an underlying POSIX-compliant file system effectively forces POSIX-IO semantics upon all other layers . For instance, this applies to the common HPC I/O stack with Lustre [20].

Yet, the actual applications used in HPC usually do not need such strict semantics. Indeed, most HPC applications do not talk to the file system directly. They use intermediate libraries like MPI-IO [21], [22], either directly or via intermediate libraries such as HDF5 [23], [24] or ADIOS [25], [26]. These libraries often provide relaxed semantics compared with POSIX. For example, MPI-IO requires a write to be visible by all processes only after the file is closed or synced [10]. Therefore, paying the performance cost of strict semantics at the storage level when these semantics are not required at higher levels is unnecessary.

Accordingly, considerable research has focused on relaxing POSIX-IO semantics. For example, OrangeFS, itself based on PVFS, relaxes the semantics of parallel writes on shared files to match those of MPI-IO. Vilayannur et al. [12] propose a subset of POSIX I/O extensions for PVFS. This work seeks to further relax the strict rules of POSIX-I/O semantics that limits application scalability.

B. Big data: from distributed file systems to object storage

Big data applications require a storage model that follows a write-once, read-many model. This requirement drove the design of many distributed file systems by sacrificing some of the POSIX-IO operations in order to gain data throughput. Google FS (GFS) [27] implements only a set POSIX-compliant operations needed by data-intensive applications, namely, create, delete, open, close, read, and write. HDFS [28] is based on GFS and is designed to work in commodity hardware. It implements some additional POSIX-IO requirements such as directory operations and file permissions, but it discards some others such as concurrent reads and writes. Ceph [29] follows the same trend, discarding some POSIX-IO semantics and implementing only those that allow a distributed file system to work with most applications. GlusterFS eliminates the metadata server and claims to be fully POSIX compliant. However, work has shown that this compliance can impact throughput [30].

We can clearly see a trend where the file system POSIX-IO API or semantics such as providing a hierarchical namespace, file permissions, or strict file access parallelism are unnecessary. Thus, they can be traded for performance and adaptability for big data applications.

C. Storage-based convergence between HPC and Big Data

During the past decade many research projects and workshops were dedicated to the opportunities and possibilities of running large scale scientific applications by using cloud computing technologies ([3], [31], [32], [33]). In general, many efforts there were made to investigate and evaluate the performance of HPC applications (mostly form life sciences [34], [35], [36]) on clouds (with and without virtualization [37]) highlighting cost efficiency or trade-offs [38]. For example, Gupta et al. [39] write that low communication-intensive applications are more suitable for cloud deployments.

Several research efforts focus on building optimized or customized distributed-computing platforms that meet the requirements of HPC applications and scientific simulations [40], [41]. Many of those are based on big data frameworks such as Spark or Hadoop / MapReduce. In contrast, Pan et al. [42] propose to port parallel file systems to cloud environments in order to support a wide range of applications expecting POSIX-IO on cloud applications. However, the application use cases considered in that work are rarely data-intensive. In the same way other researchers also aimed to provide the features of PFS in the cloud storage. For instance, Y. Abe and G. Gibson in [43] presented a storage model which gives data access to a user through the storage service layer (S3 interface) and directly through a PFS.

File systems are not the only paradigms considered for storage-based convergence. As such, with BlobSeer [48], Nicolae et al. presented a promising work on Blobs, and how they could be used alongside Hadoop to provide a solid storage base to MapReduce applications. This work proved useful in some specific HPC use-cases [49] such as checkpoint-restart.

III. Could blobs be the enabling factor?

Although the set of tools and techniques used for HPC and big data environment differ, many objectives are similar. The most important is probably to provide the highest-possible data access performance and parallelism. As such, the storage stack for HPC and big data looks similar. Indeed, the related work showed that a common trend for both HPC and big data is to relax many of the concurrent file access semantics, trading such strong guarantees for increased performance. Nevertheless, some differences remain. Specifically, while the big data community increasingly drops POSIX-IO altogether, the HPC community tends to provide this relaxed set of semantics behind the same API. Although this choice increases backwards compatibility with legacy applications, it also has significant performance impact.

Very few HPC applications actually rely on strong POSIX-IO semantics. For instance, the MPI-IO standard does not only relaxes many of these semantics but also drops many of its operations altogether. For example, it does not expose the file hierarchy or permissions to the end user. Therefore, applications leveraging these libraries do neither need nor expect these features to be provided.

Consequently, we ask ourselves whether the underlying storage technologies from both worlds could be unified, leading to specific software stacks and libraries running atop a low-level, low-opinionated storage paradigm, such as the one

TABLE I. APPLICATION SUMMARY

Platform	Application	Usage	Total reads	Total writes	R / W ratio	Profile
HPC / MPI	mpiBLAST (BLAST) [44]	Protein docking	27.7 GB	12.8 MB	2.1×10^3	Read-intensive
	MOM [45]	Oceanic model	19.5 GB	3.2 GB	6.01	Read-intensive
	ECOHAM (EH) [46]	Sediment propagation	0.4 GB	9.7 GB	4.2×10^{-2}	Write-intensive
	Ray Tracing (RT) [47]	Video processing	67.4 GB	71.2 GB	0.94	Balanced
Cloud / Spark	Sort	Text Processing	5.8 GB	5.8 GB	1.00	Balanced
	Connected Component (CC)	Graph Processing	13.1 GB	71.2 MB	0.18	Read-intensive
	Grep	Text Processing	55.8 GB	863.8 MB	64.52	Read-intensive
	Decision Tree (DT)	Machine Learning	59.1 GB	4.7 GB	12.58	Read-intensive
	Tokenizer	Text Processing	55.8 GB	235.7 GB	0.24	Write-intensive

provided by blob storage systems. Indeed, blob-based storage systems such as Týr [14] or RADOS [15] could provide a strong alternative to file-based storage on both sides. These systems typically have a much more limited set of primitive compared with file systems:

- **Blob Access**: random object read, object size,
- **Blob Manipulation**: random object write, truncate,
- **Blob Administration**: create blob, delete blob,
- **Namespace Access**: scan all blobs.

These operations are similar to those permitted by the POSIX-IO API on a single file. Therefore, most file operations performed on a file system can be mapped directly to the corresponding primitives of blob storage systems. In that model we classify file open and unlink as file operations.

In contrast, directory-level operations do not have their blob counterpart, because of the flat nature of the blob namespace. Should applications need them, such operations can be emulated using the scan operation. Obviously, this emulation is far from optimized. Yet, since we expect these calls to be vastly outnumbered by blob-level operations, this performance drop is likely to be compensated by the gains permitted by using a flat namespace and simpler semantics.

Legacy application, which could rely on a fully compliant POSIX-IO interface, could leverage a POSIX-IO interface implemented atop such blob storage. This is proven possible by the Ceph file system, a file-system interface to RADOS.

IV. ANALYZING THE DISTRIBUTION OF STORAGE CALLS

In this section we demonstrate that the actual I/O calls made by both HPC and big data applications are not incompatible with the set of features provided by state-of-the-art blob storage systems. Our intuition is that read and write calls are vastly predominant in the workloads of those applications and that other features of distributed file systems such as directory listings are rarely used, if at all.

A. Applications

We summarize the applications we use in Table I. The application. We selected these applications for their very different I/O profiles. The HPC applications we use are based on MPI. They all leverage either large input of output datasets associated with large-scale computation atop centralized storage usually provided by a distributed, POSIX-IO-compliant file system such as Lustre [16] or OrangeFS [17]. On the cloud side, as the leading open-source big data processing and analytics framework, Apache Spark [50] appears as an ideal candidate for this research. Chosen applications are extracted from SparkBench [51], a benchmarking suite for Spark. It comprises a representative set of workloads belonging to four application types: machine learning, graph processing, streaming, and SQL queries.

B. Experimental configuration

We have deployed these applications on the Grid'5000 [52] experimental testbed, distributed over 11 sites in France and Luxembourg. For these experiments, the *parapluie* cluster of Rennes was used. Each node is outfitted with 2 x 12-core 1.7 Ghz 6164 HE, 48 GB of RAM, and 250 GB HDD. Network connectivity can be handled either by Gigabit Ethernet connectivity (MTU = 1500 B) or by 4 x 20G DDR InfiniBand.

We run all applications and analyze all storage calls they perform. We log these calls for HPC applications using a FUSE interceptor. Logging for big data applications requires modifying Hadoop / HDFS to intercept all storage calls made by Spark. The results for HPC applications are obtained by using 24 compute / 8 storage nodes. Using 4 or 12 storage nodes does not lead to any significant difference in the results.

C. Tracing HPC applications

Figure 1 summarizes the relative count of storage calls performed by our set of HPC applications. The most important observation for all four applications is the predominance of reads and writes. Except for ECOHAM, no application performed any other call to the storage system that reads or writes files, thus confirming our first intuition. This was expected because the MPI-IO standard does not permit any other operation.

The few storage calls other than read and write (mainly extended attributes reads and directory listings) are due to the run script necessary to prepare the run and collect results after it finishes. These steps can be performed offline from the I/O-heavy MPI part of the application. This results in only reads and writes being performed (EH / MPI).

We conclude that the only operations performed by our set of HPC applications, namely, file I/O, can be mapped to blob I/O on a blob storage system. Consequently, these applications appear to be suited to run unmodified atop blob storage.

D. Tracing Big Data applications

Figure 2 shows the relative count of storage calls performed by our set of big data applications to HDFS. Similar to what we observed with HPC applications, the storage calls are vastly dominated by reads and writes to files. In contrast with

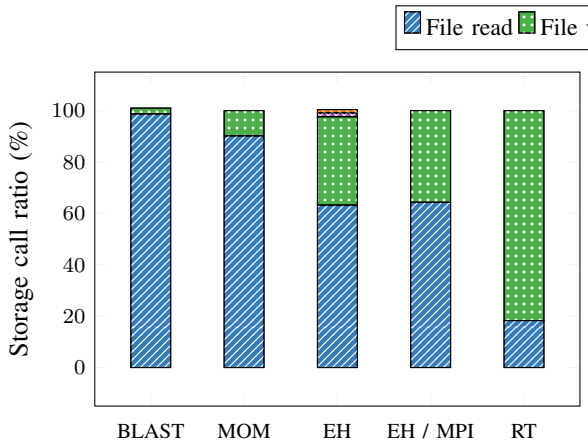

Fig. 1. Measured relative amount of different storage calls to the persistent file system for HPC applications

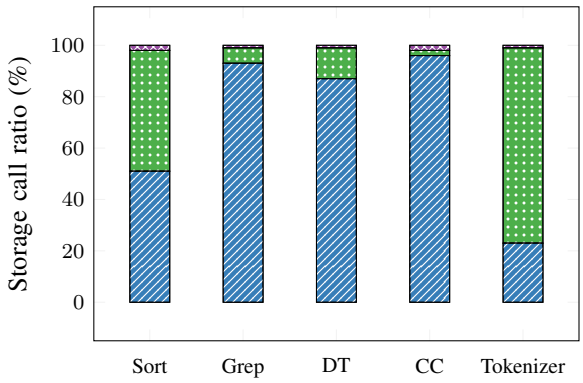

Fig. 2. Measured relative amount of different storage calls to the persistent file system for Big Data applications

TABLE II. SPARK DIRECTORY OPERATION BREAKDOWN

Operation	Action	Operation count
`mkdir`	Create directory	43
`rmdir`	Remove directory	43
`opendir` *(Input data directory)*	Open / List directory	5
`opendir` *(Other directories)*	Open / List directory	0

HPC, however, all applications also cause Spark to perform a handful of directory operations (86 in total across all our applications). These directory operations are not related to the data processing because input / output files are accessed directly by using read and write calls.

Analyzing these directory operations, we notice that they are related solely to (*a*) creating the directories necessary to maintain the logs of the application execution, (*b*) listing the input files before each application runs if the input data is set as a directory, and (*c*) maintaining the `.sparkStaging` directory. This directory is internally used by Spark to share information related to the application between nodes and is filled during the application submission. It contains application files such as the Spark jar or the application jar, as well as distributed cache files [53].

We analyze in detail the directory operations performed by big data applications. Table II shows the breakdown of all such directory operations across all applications by storage call. We note that only the input data directories are listed, meaning that Spark accesses directly all the other files it needs with their path. Consequently, a flat namespace such as the one provided by blob storage systems could probably be used.

V. CONCLUSION

As the data size used by both HPC and big data application increases, new challenges regarding managing the amount of data generated by data-intensive applications arise. The solutions widely adopted for both worlds tend to diverge because of different sets of tools and techniques being available on each platforms. Typically, the HPC community tend to favor relaxing the POSIX-IO guarantees while retaining the POSIX-IO interface to maintain support for legacy applications. In contrast, the big data community generally drops all or part of the POSIX-IO interface altogether, in order to further increase performance. Yet, in both worlds, the storage systems used for processing large sets of data still largely rely on file-based interfaces. This situation implies maintaining complex file hierarchies or permissions, which have a clear impact on the performance of storage operations.

In this paper we argue that blob storage is a strong candidate for replacing traditional storage for both HPC and big data. Its simple data model is enough to map directly file operations to blob operations. We prove in our paper that these calls constitute the vast majority of the storage calls made by HPC or Big Data applications. In Section IV, we use a simple logging adapter for HPC and big data applications. The results demonstrate that HPC applications running atop MPI-IO do not perform any call other than file operations, where file reads and writes are almost all of them. One application (ECOHAM) requires a few directory operations that are merely part of the run preparation scripts, and can be run offline before the application. On the big data side also, we prove that more than 98% of the storage calls are file operations. Finally, the hierarchical namespace is used by Spark only for convenience and is not necessary for the application to run.

In future work we will replace file systems by blog storage systems for the representative set of applications presented in this paper. We will demonstrate factually that the gains obtained by transitioning to a hierarchical namespace to a flat one leads to significant I/O performance improvements.

ACKNOWLEDGMENTS

This work is part of the "BigStorage: Storage-based Convergence between HPC and Cloud to handle Big Data" project, H2020-MSCA-ITN-2014-642963, funded by the European Commission within the Marie Skłodowska-Curie Actions framework. This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under Contract DE-AC02-06CH11357. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities and organizations.

REFERENCES

[1] "Microsoft Azure – HPC & Batch," 2017, https://azure.microsoft.com/en-us/solutions/big-compute/.

[2] "Amazon Web Services – High performance computing," 2017, https://aws.amazon.com/hpc/.

[3] "BDEC – Big Data and Extreme-Scale Computing," 2017, http://www.exascale.org/bdec/.

[4] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, jun 2015.

[5] G. Aloisio, S. Fiore, I. Foster, and D. Williams, "Scientific big data analytics challenges at large scale," BDEC, Tech. Rep., 2013.

[6] J. Cope, K. Iskra, D. Kimpe, and R. B. Ross, "Bridging HPC and grid file I/O with IOFSL," in *Applied Parallel and Scientific Computing - 10th International Conference, PARA 2010, Reykjavík, Iceland, June 6-9, 2010, Revised Selected Papers, Part II*, 2010, pp. 215–225.

[7] C. Pei, X. Shi, and H. Jin, "Improving the memory efficiency of in-memory mapreduce based HPC systems," in *Algorithms and Architectures for Parallel Processing - 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015, Proceedings, Part I*, 2015, pp. 170–184.

[8] J. F. Lofstead and R. Ross, "Insights for exascale IO APIs from building a petascale IO API," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, Denver, CO, USA - November 17 - 21, 2013*, 2013, pp. 87:1–87:12.

[9] D. Kimpe and R. Ross, "Storage models: Past, present, and future," *High Performance Parallel I/O*, pp. 335–345, 2014.

[10] R. Latham, R. B. Ross, and R. Thakur, "The impact of file systems on MPI-IO scalability," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 19-22, 2004, Proceedings*, 2004, pp. 87–96.

[11] J. Shafer, S. Rixner, and A. Cox, "The Hadoop distributed filesystem: Balancing portability and performance," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, March 2010, pp. 122–133.

[12] M. Vilayannur, S. Lang, R. Ross, R. Klundt, and L. Ward, "Extending the POSIX I/O interface: A parallel file system perspective," Argonne National Laboratory, Tech. Rep. ANL/MCS-TM-302, 10 2008.

[13] M. Kuhn, J. M. Kunkel, and T. Ludwig, "Dynamically Adaptable I/O Semantics for High Performance Computing," in *High Performance Computing*, ser. Lecture Notes in Computer Science, no. 9137. Switzerland: Springer International Publishing, 06 2015, pp. 240–256.

[14] P. Matri, A. Costan, G. Antoniu, J. Montes, and M. S. Pérez, "Týr: Blob storage meets built-in transactions," in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2016, pp. 573–584.

[15] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn, "Rados: A scalable, reliable storage service for petabyte-scale storage clusters," in *Proceedings of the 2nd international workshop on petascale data storage: held in conjunction with Supercomputing'07*. ACM, 2007, pp. 35–44.

[16] "The Lustre File System," 2017, http://lustre.org/.

[17] M. Moore, D. Bonnie, W. Ligon, N. Mills, , S. Yang, B. Ligon, M. Marshall, E. Quarles, S. Sampson, and B. Wilson, "OrangeFS: Advancing PVFS," in *2011 9th USENIX Conference on File and Storage Technologies (FAST)*, 2011.

[18] D. Huang, J. Yin, J. Wang, X. Zhang, J. Zhang, and J. Zhou, "UNIO: A unified I/O system framework for hybrid scientific workflow," in *Cloud Computing and Big Data - Second International Conference, CloudCom-Asia 2015, Huangshan, China, June 17-19, 2015, Revised Selected Papers*, 2015, pp. 99–114.

[19] S. Patil and G. A. Gibson, "Scale and concurrency of GIGA+: file system directories with millions of files," in *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011*, 2011, pp. 177–190.

[20] M. Kuhn, "A Semantics-Aware I/O Interface for High Performance Computing," in *Supercomputing*, ser. Lecture Notes in Computer Science, J. M. Kunkel, T. Ludwig, and H. W. Meuer, Eds., no. 7905. Berlin, Heidelberg: Springer, 06 2013, pp. 408–421.

[21] T. Sterling, E. Lusk, and W. Gropp, *Beowulf Cluster Computing with Linux*, 2nd ed. Cambridge, MA, USA: MIT Press, 2003.

[22] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J.-P. Prost, M. Snir, B. Traversat, and P. Wong, *Overview of the MPI-IO Parallel I/O Interface*. Boston, MA: Springer US, 1996, pp. 127–146.

[23] "Hierarchical data format version 5," 2017, https://hdfgroup.org/HDF5.

[24] C. Bartz, K. Chasapis, M. Kuhn, P. Nerge, and T. Ludwig, "A best practice analysis of HDF5 and NetCDF-4 using lustre," in *High Performance Computing*, ser. Lecture Notes in Computer Science, J. M. Kunkel and T. Ludwig, Eds., no. 9137. Switzerland: Springer International Publishing, 06 2015, pp. 274–281.

[25] "The adaptable io system (ADIOS)," 2017, https://www.olcf.ornl.gov/center-projects/adios/.

[26] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. F. Lofstead, R. Oldfield, M. Parashar, N. F. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu, "Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, 2014.

[27] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[28] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *2010 IEEE 26th symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2010, pp. 1–10.

[29] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[30] S. Mikami, K. Ohta, and O. Tatebe, "Using the gfarm file system as a POSIX compatible storage platform for Hadoop MapReduce applications," in *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*. IEEE Computer Society, 2011, pp. 181–189.

[31] Z. Zhang, K. Barbary, F. A. Nothaft, E. R. Sparks, O. Zahn, M. J. Franklin, D. A. Patterson, and S. Perlmutter, "Scientific computing meets big data technology: An astronomy use case," in *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*, 2015, pp. 918–927.

[32] W. Lu, J. Jackson, and R. S. Barga, "AzureBlast: A case study of developing science applications on the cloud," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC 2010, Chicago, Illinois, USA, June 21-25, 2010*, 2010, pp. 413–420.

[33] J. L. Vázquez-Poletti, D. Santos-Muñoz, I. M. Llorente, and F. Valero, "A cloud for clouds: Weather research and forecasting on a public cloud infrastructure," in *Cloud Computing and Services Sciences - International Conference in Cloud Computing and Services Sciences, CLOSER 2014, Barcelona, Spain, April 3-5, 2014, Revised Selected Papers*, 2014, pp. 3–11.

[34] H. A. Duran-Limon, J. Flores-Contreras, N. Parlavantzas, M. Zhao, and A. Meulenert-Peña, "Efficient execution of the WRF model and other HPC applications in the cloud," *Earth Science Informatics*, vol. 9, no. 3, pp. 365–382, 2016.

[35] E. D. Carreño, E. Roloff, and P. O. A. Navaux, "Porting a numerical atmospheric model to a cloud service," in *High Performance Computing - Second Latin American Conference, CARLA 2015, Petrópolis, Brazil, August 26-28, 2015, Proceedings*, 2015, pp. 50–61.

[36] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. L. Salzberg, "Searching for SNPs with cloud computing," *Genome Biology*, vol. 10, no. 11, p. R134, 2009.

[37] A. Jaikar and S. Noh, "Cloud computing: Read before use," *T. Large-Scale Data- and Knowledge-Centered Systems*, vol. 30, pp. 1–22, 2016.

[38] A. Gupta and D. Milojicic, "Evaluation of HPC applications on cloud," in *Open Cirrus Summit (OCS), 2011 Sixth*. IEEE, 2011, pp. 22–26.

[39] A. Gupta, L. V. Kale, F. Gioachin, V. March, C. H. Suen, B.-S. Lee, P. Faraboschi, R. Kaufmann, and D. Milojicic, "The who, what, why, and how of high performance computing in the cloud," in *2013 IEEE 5th*

International Conference on Cloud Computing Technology and Science (CloudCom), vol. 1. IEEE, 2013, pp. 306–314.

[40] R. Ledyayev and H. Richter, "High performance computing in a cloud using OpenStack," *Cloud Computing*, pp. 108–113, 2014.

[41] P. Jakovits, S. N. Srirama, and I. Kromonov, "Stratus: A distributed computing framework for scientific simulations on the cloud," in *14th IEEE International Conference on High Performance Computing and Communication & 9th IEEE International Conference on Embedded Software and Systems, HPCC-ICESS 2012, Liverpool, United Kingdom, June 25-27*, 2012, pp. 1053–1059.

[42] A. Pan, J. P. Walters, V. S. Pai, D. I. D. Kang, and S. P. Crago, "Integrating high performance file systems in a cloud computing environment," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov. 2012, pp. 753–759.

[43] Y. Abe and G. Gibson, "pwalrus: Towards better integration of parallel file systems into cloud storage," in *Cluster Computing Workshops and Posters (Cluster Workshops), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–7.

[44] "mpiBLAST: Open-Source Parallel BLAST," 2017, http://www.mpiblast.org/.

[45] "Ocean circulation models," 2017, https://www.gfdl.noaa.gov/ocean-model/.

[46] U. H. Institute of Oceanography, "ECOHAM," https://wiki.zmaw.de/ifm/ECOHAM, 2015.

[47] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "BigDataBench: A big data benchmark suite from Internet services," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, feb 2014.

[48] B. Nicolae, G. Antoniu, L. Bougé, D. Moise, and A. Carpen-Amarie, "BlobSeer: Next-generation data management for large scale infrastructures," *J. Parallel Distrib. Comput.*, vol. 71, no. 2, pp. 169–184, 2011.

[49] B. Nicolae and F. Cappello, "BlobCR: Efficient checkpoint-restart for HPC applications on IaaS clouds using virtual disk image snapshots," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 34:1–34:12.

[50] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[51] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, "Sparkbench: a comprehensive benchmarking suite for in memory data analytic platform Spark," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*. ACM, 2015, p. 53.

[52] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. Ivanov, M. Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[53] "Running Spark on YARN," 2017, https://spark.apache.org/docs/latest/running-on-yarn.html.