Optimizing Data Placement on Hierarchical Storage Architecture via Machine Learning

Peng Cheng, Yutong Lu, Yunfei Du, Zhiguang Chen, Yang Liu

Optimizing Data Placement on Hierarchical Storage Architecture via Machine Learning

Peng Cheng[1,2], Yutong Lu*[3], Yunfei Du[3], Zhiguang Chen[3], and Yang Liu[4]

[1] College of Computer, National University of Defense Technology, Changsha, China
[2] State Key Laboratory of High Performance Computing, Changsha, China
[3] National Supercomputer Center in Guangzhou, School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China
[4] Department of Computer Science and Technology, Tsinghua University, Beijing, China
{peng.cheng, yutong.lu, yunfei.du, zhiguang.chen}@nscc-gz.cn, liuyang2011@tsinghua.edu.cn

Abstract. As storage hierarchies are getting deeper on modern high-performance computing systems, intelligent data placement strategies that can choose the optimal storage tier dynamically is the key to realize the potential of hierarchical storage architecture. However, providing a general solution that can be applied in different storage architectures and diverse applications is challenging. In this paper, we propose adaptive storage learner (ASL), which explores the idea of using machine learning techniques to mine the relationship between data placement strategies and I/O performance under varied workflow characteristics and system status, and uses the learned model to choose the optimal storage tier intelligently. We implement a prototype and integrate it into an existing data management system. Empirical comparison based on real scientific workflows tests shows that ASL is capable of combining workflow characteristics and real-time system status to make optimal data placement decisions.

Keywords: Storage Optimization · Machine Learning · Hierarchical Storage · Data Placement

1 Introduction

With the converging of High-Performance Computing (HPC) and big data, massive datasets are produced and analyzed by HPC systems. For example, the large N-body simulation that evolved more than a trillion particles on the BG/Q Mira system generates approximately 5PB of raw outputs [1]. The exascale deep learning on the Summit system analyzes 3.5TB climate data [2] to detect extreme weather. As scientific workflows become more complex and more data-intensive, supporting these workflows on HPC systems present serious challenges in I/O performance [3,4].

To improve the I/O performance, many modern HPC systems use middleware or heterogeneous storage devices to expand the storage subsystem in a

hierarchical manner. Memory-based staging solutions, such as DataSpaces [5], use memory space of compute nodes to stage intermediate data. Shared burst buffer strategy, such as Cori system [6], uses SSDs near I/O nodes and Cray DataWarp software [7] to implement a shared staging area for coupling applications. Local burst buffer strategy, such as Summit system [8], equips per-node attached SSD for each compute node to buffer intermediate data locally. Meanwhile, current research on storage class memories (SCM) is expected to be able to add additional layers to the memory and storage hierarchy in future HPC systems [9].

Due to the largely different latency, bandwidth and capacity of heterogeneous storage devices, different data placement strategies could lead to wildly varying I/O performance. However, existing storage systems usually apply one-layout-fits-all strategy or leave the burden of making data placement decisions to users [10]. Such fixed and manual data placement might result in the inefficient use of hierarchical storage architecture because of load imbalance [11] and resource contention [12]. Previous works [13,14] provide workflow-aware data placement mechanism but depend on user-provided hints, which is infeasible for complex workflows. While Stacker [15] leverages hierarchical n-grams model to predict upcoming read request and guide data prefetching, smart data placement decisions across different storage tiers remain to be researched.

In this paper, we propose adaptive storage learner (ASL), which explores the idea of leveraging machine learning techniques to mine the relationship between data placement strategies and I/O performance under varied workflow characteristics and system status, and uses the learned model to choose the optimal storage tier dynamically during the workflow execution. Compared with previous works, ASL focus on scientific workflows and enables intelligent data placement strategy to make the most benefit of hierarchical storage architecture without any user-provided hints. We provide a prototype implementation of ASL and integrate it with Alluxio [16] data management system. Our evaluations of two scientific workflows validate the effectiveness of ASL in combining workflow characteristics and real-time system status to make optimal data placement decisions. While this paper focuses on optimizing data placement across different storage tiers, the idea of training a classification model to guide storage optimization can be applied in different scenarios. Our contributions in this paper can be summarized as follows.

- A workflow simulator with I/O performance model that reflects the changing workloads, heterogeneous storage devices, and varying configurations.
- A classification model that leverages workflow characteristics and system status to make optimal data placement decisions.
- A prototype implementation that manages data on tiered storage architecture and enables intelligent data management decisions.
- An extensive evaluation with two scientific workflows on a typical HPC system.

The rest of this paper is organized as follows. Section 2 presents a brief background and the motivation of this paper. We explore the idea of training a

classification model for storage optimization in Section 3 and present the design and implementation in Section 4. We validate the effectiveness of ASL in Section 5. Section 6 discusses some related studies currently existing in the literature. We conclude the paper and talk about future works in Section 7.

2 Background and Motivation

Scientific workflows: A scientific workflow is the assembly of complex sets of scientific data processing activities with data dependencies between them [17]. Due to the repetitive nature of scientific discovery, scientific workflow management system (SWfMS) like Pegasus [18] and Swift [19] are increasingly used in HPC environments to manage the complex simulations and analyses. Workflow description file contains the necessary information about the entire workflow, including the tasks to be executed and data-flow/control-flow dependencies between these tasks. SWfMSs take the description of the abstract workflow as input and coordinate and execute workflow tasks over available computing resources.

Motivation: Data placement strategies on hierarchical storage architecture can be divided into horizontal placement (how data are distributed inside a storage layer) and vertical placement (how data are distributed across different storage layers). In this paper, we focus on vertical data placement since heterogeneous storage devices show largely different latency, bandwidth, and capacity. While different data placement strategies could lead to wildly varying I/O performance, existing data management systems [15,16] often apply the one-layout-fits-all strategy that uses the top storage tier (e.g., memory tier) as the performance tier and uses lower storage tier (e.g., SSD tier or HDD tier) as capacity tier. However, such fixed data placement strategy might result in the inefficient use of hierarchical storage architecture. Firstly, serious **load imbalance** occurs since the fixed data placement strategy keeps staging data to a specific storage layer, while other storage layers keep unused [11]. Secondly, as the available space of that layer gets insufficient, backend data migration requests need to move data to lower tiers. The **resource contention** between regular write requests and backend data migration requests could even lead to almost 70% performance degradation [12].

3 Training Classification Model for Storage Optimization

The basic idea that guides our design is that both data access patterns and real-time system status should be taken into consideration to make the optimal data placement decision. For example, if the top storage layer has plenty of space to stage all the intermediate data during the execution of scientific workflows, choosing the top storage layer to serve every write request will provide superior I/O performance. Otherwise, only data will be accessed by subsequent task immediately can be written into the top storage layer, and other data should be written to the lower storage layer to keep load balance. While setting these rules

manually may perform well for some applications, providing a general solution that can be applied in different storage architectures and diverse applications is challenging. In this paper, we propose Adaptive Storage Learner (ASL), which explore the idea of using machine learning techniques to solve this challenge.

3.1 Problem Definition

Selecting the optimal storage layer can be regarded as a multi-classification problem. We want to learn a model that takes parameters related to workflow characteristics and system status as input and predicts the optimal storage layer for each output file. We maintain three principles to train the prediction model:

- Both workflow characteristics and real-time system status are taken into consideration to make the optimal data placement decision.
- The optimization goal of the prediction model is not to minimize the I/O time of a single task, but to minimize the overall I/O time of the entire workflow by leveraging data access patterns and preventing resource contention. In other words, the optimal storage tier for an output file may not be the fastest storage tier, even if there is plenty of space currently.
- We set the granularity of a data placement decision to a file instead of each write request since dividing a file across a slow and a fast tier may lead to the problem that a slow tier becomes the bottleneck.

To train such a multi-classification model, we first identify parameters that affect I/O performance.

3.2 Parameters Affecting I/O Performance

Scientific workflows might demonstrate different I/O performance based on the workflow characteristics and system specifications. We summarize three sets of parameters that might affect the I/O performance and explain some of them because of space limitations.

Workflow characteristics: These parameters include the scale of the workflow, the control-flow dependencies and data-flow dependencies between tasks. Specifically, data-flow dependencies contain data access pattern information of each output file. As previous works have demonstrated [14], data access patterns can be leveraged to improve I/O performance. For example, staging a file that will be accessed immediately to the top storage tier can reduce the data read time. Compared with previous works depend on user-provided hints to identify the data access patterns, we don't set any rules manually but provide all these information to the prediction model and let the model learns from it. All these parameters are statically determined before running a workflow and can be retrieved by parsing the workflow description file.

Runtime storage information: These parameters describe the storage information of generated intermediate files during the workflow execution, such as the size of each intermediate file and the storage tier that file resides. All these parameters are collected during the execution of the workflow.

Table 1. Variables contained in each I/O record

Variable	Description
V1	ID of the current output file
V2	Type of current task
V3	Number of tasks of the current type
V4	Number of input files of the current task
V5	Number of output files
V6	Number of tasks that is dependent with current output file
V7	Minimum distance between the output file and dependent tasks
V8	Total size of input files of the current task
V9	The remaining capacity of the memory storage tier
V10	The remaining capacity of the SSD storage tier
V11	The remaining capacity of the HDD storage tier
Prediction	The optimal storage tier

System status: These parameters are specifications of the system where the workflow runs, including the deployment of the storage subsystem, the performance metrics of different storage tier, etc. All these parameters affect the I/O performance of a given workflow. While parameters like bandwidth and latency of each storage tier can be obtained statically, parameters like the remaining capacity of each storage tier require real-time monitoring.

3.3 Collecting I/O records

After identifying parameters that affect the I/O performance, we are able to collect the I/O records during the workflow execution. We model these parameters into 11 variables listed in Table 1. Each I/O record represents a data placement decision for a given output file under the conditions described by 11 variables. Specifically, V1 and V2 are used to identify the data producer for each file create request. V3-V7 reflect the workflow characteristics and V8-V11 describe the real-time storage information and system status.

It's deserved to be mentioned that the size of each output file will definitely influence the I/O performance, but we abandon it to avoid the contradiction against the usage of the prediction model. The goal of the prediction model is to make data placement decisions before data are written to the target storage tier. However, the size of an output file can be calculated only after the write operation is finished. A compromise solution is to train another regression model to predict the size of the output file, but we do not implement it in our current work.

Collecting I/O records is complicated and cumbersome for two reasons. Firstly, all the I/O records need to be labeled since multi-classification problem requires supervised learning. In other words, each record must be labeled with the target storage tier explicitly before they can be used to train the prediction model. Secondly, recall the principle that the optimization goal of the prediction model is to minimize the overall I/O time of the entire workflow by leveraging data

access patterns and preventing resource contention. This principle exacerbates the complexity of labeling records since impropriate training data leads to the inaccurate prediction model.

To solve this challenge, we extend a workflow simulator [20] to simulate the I/O performance and label the I/O records automatically. In general, the extended workflow simulator has the following design considerations:

Tiered storage architecture: we add hybrid storage module consists of three storage tiers to model the I/O performance. Specifically, the specification of each storage tier is set based on real system tests. Detailed configurations are discussed in Section 5.

Simulation rules: Several rules are set empirically to simulate the actual I/O performance. These include: the location of data contribute to the maximum read/write bandwidth, the bandwidth degradation once the available space of a storage tier exhausted. We do not list all the rules here because of space limits. To validate the effectiveness of the simulation, we compare the simulated I/O time and the actual I/O time of the Binary-Tree workflow [21]. Fig. 1 illustrates the result of each type of task. When storage tier is set to memory, SSD, and HDD, the I/O time generated by workflow simulator are noted as *Sim-Mem*, *Sim-SSD*, and *Sim-HDD*, respectively. Similarly, the I/O time of running workflows on the real system is noted as *Real-Mem*, *Real-SSD*, and *Real-HDD*. Overall, the difference between simulation and real system tests is less than 10%.

Genetic Algorithm (GA): To search the optimal combinations of storage tiers for a given workflow, we implement a genetic algorithm in the workflow simulator. We treat a candidate storage tier combination as an individual, and the storage tier of each output file is represented as 2 genes (since each gene is a Boolean variable, at least two genes are needed to represent 3 storage tiers). The GA starts from a population of randomly generated individuals and evolves in an iterative process. The overall I/O time of a workflow is used to calculate the fitness of each individual. An individual is qualified to have the next generation only when its fitness is no less than the average fitness. The iterative crossover and mutation between qualified individuals improve the quality of the represented solution. Finally, the best individual is chosen to be the optimal combinations of storage tiers.

We randomly chose 58 workflows with varying scales and I/O characteristics from the synthetic workflow dataset [22]. We ran these workflows on top of the simulator and collect 3810 labeled I/O records.

3.4 Model training

Gradient boosting algorithm with Classification and Regression Tree (CART) as base learners is used to train the prediction/classification model. While there are lots of machine learning algorithms, including logistic regression and support vector machines, we chose CART as the basic learner for two reasons. Firstly, CART is easy to understand and interpret. Secondly, the prediction overhead of CART is negligible. Since a single CART model might suffer from the poor generality, we use the gradient boosting technique, which averages over multiple

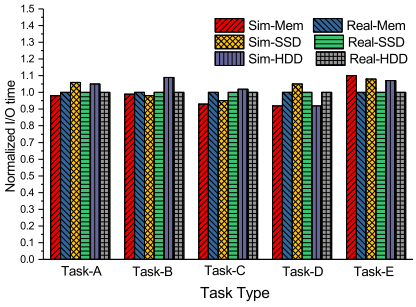


Fig. 1. Accuracy of simulation output

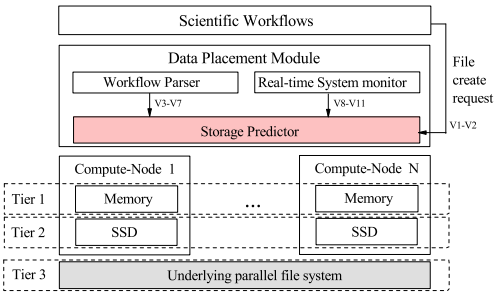


Fig. 2. ASL architecture overview

CART classifiers and produces the final prediction, to enhance its generalization ability. The final prediction model can be treated as an ensemble of CART models.

4 Design and Implementation

We design and implement a prototype of ASL that uses the prediction model presented in section 3 to make optimal data placement decisions.

Fig. 2 presents the architecture overview of ASL. ASL acts as a middleware integrated with an existing data management system that can manage data on tiered storage architecture. The key component of ASL includes workflow parser, real-time system monitor and storage predictor. The workflow parser extracts workflow characteristics form the workflow description file before running a given workflow. The real-time system monitor collects system status during the workflow execution. For each file create request, the storage predictor combines workflow characteristics and system status to make data placement decisions.

Parser and Monitor: To enable workflow-aware storage optimization, we implement a workflow parser that extracts workflow characteristics from the workflow description file. The extracted data are stored in multiple in-memory data structures and transform into variables V3-V7 as listed in Table 1. Although the workflow description file contains lots of valuable information, information like the size of a specific input file and the remaining capacity of each storage tier can only be collected during the workflow execution. The system monitor is used to collect such information dynamically. The dynamically collected information transformed into variables V8-V11 as listed in Table 1.

Prediction: The storage predictor uses the prediction model to make the data placement decision for every incoming file create request. Specifically, the decision-making process can be summarized into the following steps: 1. After receiving the file create request, the storage predictor verifies the type of the

current task based on the name of the created file and the extracted workflow description info. 2. Retrieving workflow characteristics and system status from the workflow parser and the system monitor, respectively. 3. Constructing variables related to workflow characteristics and system status and feed these variables to the prediction model. 4. Predicting the optimal storage tier for the newly created file. As in the case of new workflows, the storage predictor can also predict the result since none of the input variables depend on historical information.

Implementation: We have implemented a prototype of ASL and integrated it with Alluxio. The workflow parser is implemented as a command line utility. All the parsed workflow dependent information are sent to Alluxio master, which manage the metadata of the storage system and serve metadata requests. We add extra modules to manage workflow dependent data structures. The system monitor and the storage predictor are also implemented in Alluxio master to guide data placement for every incoming file create request.

5 Evaluation

To demonstrate the effectiveness of ASL, we evaluate its performance on a typical HPC system with real scientific workflows.

5.1 Experimental Setup

Our testbed consists of 32 nodes configured in one rack on the on data analytics cluster of the Tianhe-2 system [23]. Each node is equipped with two 2.20GHz Intel Xeon E5-2692-v2 processors (24 cores per node), 64 GB of RAM and one PCIe 1.5TB SSD. Alluxio is used to manage data on top of heterogeneous storage devices. Specifically, two nodes are used as the master nodes to manage the global metadata, and the other 30 nodes are used as the workers to stage data into local memories or per-node attached SSDs. We allocate 2-15GB RAM of each node to constitute the memory storage tier. The per-node attached SSDs constitute the SSD storage tier, and the underlying Lustre file system acts as the HDD storage tier. Detailed specifications of each storage tier under the management of Alluxio are listed in Table 2.

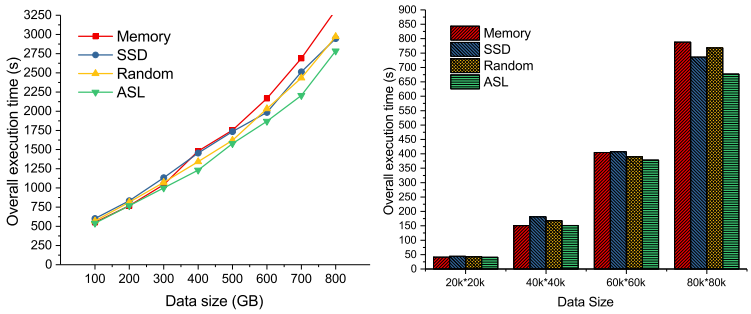
We use Binary tree workflow[21] and GenBase workflow [24] to validate the effectiveness of ASL. These workflows are also used to train the prediction model, but none of the simulated workflow scales are used during the real-system evaluation. All intermediate data are staged into hierarchical storage architecture managed by Alluxio.

5.2 Decision-making under varied workflow scales

Firstly, we validate the effectiveness of ASL in making optimal data placement decisions based on workflow characteristics. We set capacity of memory storage layer to 300GB and vary the scales of workflows. For GenBase workflow with

Table 2. Storage configurations under the management of Alluxio

Tier	Allocated Capacity	Write BW	Read BW
Memory Tier	60-450GB	950MB/s	1100MB/s
SSD Tier	1200GB	800MB/s	850MB/s
HDD Tier	1200GB	500MB/s	550MB/s



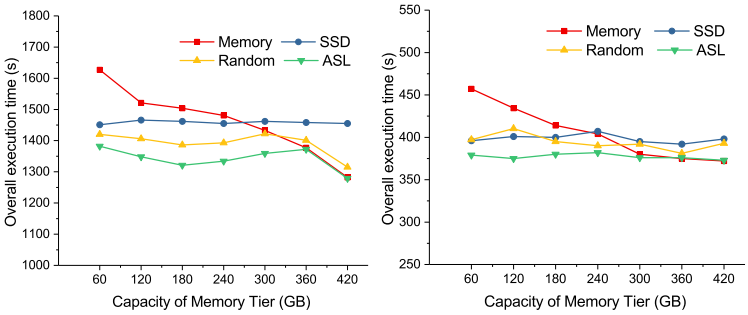
(a) Binary-Tree workflow (b) GenBase workflow

Fig. 3. Performance of varied scales workflows

80k*80k input data scales, 120GB of raw data are processed and will generate more than 400GB intermediate data. While the initial storage configuration of each run is fixed, we evaluate the performance of four data placement strategies. Staging all data into memory storage tier and SSD storage tier are noted as *Memory* and *SSD*, respectively. Selecting the memory or the SSD tier for each file randomly is noted as *Random*. Using the predictor and make data placement decision intelligently is noted as *ASL*. Since the SSD storage tier is sufficient to stage all the intermediate data during the evaluation, HDD tier is not used in consideration of I/O performance. Fig. 3 shows the performance of the Binary-Tree and the GenBase workflow.

For the Binary-Tree workflow, when data size is smaller than 300GB, memory storage layer has enough space to stage all the intermediate data. As a result, *Memory* strategy performs better than *SSD* and *Random* strategy. As data size keeps increasing, *Memory* strategy migrates data from the memory tier to the SSD tier to make room for the newly created file. The resource contention between regular write request and backend data migration request leads to performance degradation. Since SSD tier has enough space to stage all intermediate data during our evaluations, resource contention problem does not occur in *SSD* strategy. The *Random* strategy alleviates the load imbalance problem to some extent, but data access patterns of workflows are not taken into consideration.

Contrast to these strategies, *ASL* combines information including available space of each storage layer, input size of the current task, and distance of dependent task to choose the optimal storage tier for each intermediate file. When the memory storage tier has plenty of space to stage all the intermediate data, *ASL*



(a) Binary-Tree workflow (b) GenBase workflow

Fig. 4. Performance of varied memory capacity

choose the memory tier as the primary storage tier to minimize the data access time. As data size increases and the memory storage tier gets exhausted, *ASL* leverages the hidden data access pattern info and only stage data that will be accessed by subsequent tasks intermediately to the memory tier to prevent resource contention. As a result, *ASL* shows the best performance in all cases. For the GenBase workflow, *ASL* shows similar performance with *Memory* strategy at first and outperforms other strategies as data size increases.

5.3 Decision-making under varied system status

Secondly, we validate the effectiveness of ASL in making optimal data placement decisions based on system status. The scales of Binary-Tree and GenBase workflow are set to 400GB and 60k*60k, respectively. We vary the capacity of the memory storage tier from 60GB to 420GB and show the result in Fig. 4. For the Binary-Tree workflow, when the capacity of the memory storage tier is less than 240GB, *Memory* strategy performs the worst because of the serious load imbalance and resource contention. As available capacity keeps increasing, *Memory* strategy shows better performance. While *SSD* strategy shows stable performance as expected, *Random* strategy performs better as memory capacity increases. In comparison, *ASL* performs the best by making optimal data placement decision based on workflow characteristics and system status. For the GenBase workflow, since 60k*60k input data scale generates almost 250GB intermediate data, *ASL* performs the best at first and shows similar performance with *Memory* strategy as the capacity of memory tier is larger than 300GB.

In summary, our evaluations validate the effectiveness of ASL in combining workflow characteristics and real-time system status to make intelligent data placement decisions.

6 Related work

As storage hierarchies are getting deeper on HPC systems, managing data on tiered storage architecture are getting increased attention. Data Elevator [12] enables asynchronously data flushing from burst buffer to the PFS, but different storage layers are managed separately. Heterogeneity-Aware Tiered Storage [25] and OctopusFS [11] extend HDFS to support tiered data storage architecture. They propose data placement and retrieval policies based on I/O throughput and capacity of storage devices, however, data access patterns are not used to make data management decisions. While Alluxio [16] and UniStor [26] provide a unified view across different storage layers, both of them lack the ability to choose the optimal storage tier dynamically. Multi-tiered data staging framework [13], Hermes [27] and TDMS [14] provide application-aware data placement mechanism but depend on user-provided hints to identify the data access patterns. Compared with these works, we treat selecting the optimal storage tier as a multi-classification problem and use machine learning techniques to make data placement strategies intelligently.

Many efforts have been made to enable adaptive and intelligent storage optimization. Stacker [15] chooses hierarchical n-grams model to predict upcoming read request and guide data prefetching on hierarchical storage architecture. Since stripe size and the distribution of correlated blocks dominate the aggregate bandwidth of parallel file systems, Dong Dai et al. [28] explored the idea of using word embedding technique to mine the block correlations. Erica Tomes et al. [29] combined graph coloring, bin packing, and network flow techniques to distribute correlated data blocks to different storage servers adaptively. Compared with these works, we focus on data placement and load balance across different storage hierarchies. Ziggurat [30] profiles the application's access stream online to predict the behavior of individual writes and chooses non-volatile main memory or disks to serve the write request. While the classification criteria in Ziggurat is set empirically, ASL does not depend on any manual rules and combine both workflow characteristics and real-time system status to choose the optimal storage tier.

7 Conclusion

Due to the largely different performance characteristics of hierarchical storage layers and the variety of scientific workflows, providing a general solution that can make intelligent data placement decisions is challenging. In this paper, we explore the idea of using machine learning techniques to solve this challenge. We propose that selecting the optimal storage layer under varied workflow characteristics and system status can be regarded as a multi-classification problem. We implement a workflow simulator to collect labeled I/O records automatically and use the gradient boosting algorithm with CART as base learners to train the classification model. We implement a prototype and integrate it into Alluxio system. Our evaluations on two scientific workflows validate the effectiveness of

using machine techniques to optimize I/O performance. In our current implementation, the prediction model is not modified once it is deployed. In future work, we plan to collect histories records of workflows and update the model dynamically.

Acknowledgment

This work was supported by the National Key R&D Program of China under Grant No. 2017YFB0202204 and No. 2017YFB0202201, the National Science Foundation of China under Grant NO.U1811464, and the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant NO. 2016ZT06D211.

References

1. Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, and Tom Peterka. Hacc: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy*, 42:49–65, 2016.
2. Thorsten Kurth, Sean Treichler, Joshua Romero, Mayur Mudigonda, Nathan Luehr, Everett H. Phillips, Ankur Mahesh, Michael Matheson, Jack Deslippe, Massimiliano Fatica, Prabhat, and Michael Houston. Exascale deep learning for climate analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*, pages 51:1–51:12, 2018.
3. Takemasa Miyoshi, Guo Yuan Lien, Shinsuke Satoh, Tomoo Ushio, Kotaro Bessho, Hirofumi Tomita, Seiya Nishizawa, Ryuji Yoshida, Sachiho A. Adachi, and Jianwei Liao. Big data assimilation toward post-petascale severe weather prediction: An overview and progress. *Proceedings of the IEEE*, 104(11):2155–2179, 2016.
4. Ning Liu, Jason Cope, Philip H. Carns, Christopher D. Carothers, and Robert B. Ross et al. On the role of burst buffers in leadership-class storage systems. In *IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST 2012, April 16-20, 2012, Asilomar Conference Grounds, Pacific Grove, CA, USA*, pages 1–11, 2012.
5. Ciprian Docan, Manish Parashar, and Scott Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15(2):163–181, 2012.
6. Wahid Bhimji, Deborah Bard, and Melissa Romanus. Accelerating science with the nersc burst buffer early user program. In *LBNL LBNL-1005736*, 05 2016.
7. Cray. Datawarp user guide s-2558-5204, June 2016. `http://docs.cray.com/books/S-2558-5204/S-2558-5204.pdf`.
8. Oak Ridge National Laboratories. Summit user guide, May 2019. `https://www.olcf.ornl.gov/for-users/system-user-guides/summit`.
9. Shivam Swami and Kartik Mohanram. Reliable non-volatile memories: Techniques and measures. *IEEE Design and Test*, PP(99):1–1, 2017.
10. Haoyuan Li, Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA*, pages 6:1–6:15, 2014.

11. Elena Kakoulli and Herodotos Herodotou. Octopusfs: A distributed file system with tiered storage management. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 65–78, 2017.
12. Bin Dong, Suren Byna, Kesheng Wu, Prabhat, Hans Johansen, Jeffrey N. Johnson, and Noel Keen. Data elevator: Low-contention data movement in hierarchical storage system. In *23rd IEEE International Conference on High Performance Computing (HiPC'16) , Hyderabad, India*, pages 152–161, 2016.
13. Tong Jin, Fan Zhang, Qian Sun, Hoang Bui, Melissa Romanus, Norbert Podhorszki, Scott Klasky, Hemanth Kolla, Jacqueline Chen, Robert Hager, Choong-Seock Chang, and Manish Parashar. Exploring data staging across deep memory hierarchies for coupled data intensive simulation workflows. In *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS'15*, pages 1033–1042, 2015.
14. Peng Cheng, Yutong Lu, Yunfei Du, and Zhiguang Chen. Accelerating scientific workflows with tiered data management system. In *IEEE International Conference on High Performance Computing and Communications*, 2018.
15. Pradeep Subedi, Philip E. Davis, Shaohua Duan, Scott Klasky, Hemanth Kolla, and Manish Parashar. Stacker: an autonomic data movement engine for extreme-scale data staging-based in-situ workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'18)*, pages 73:1–73:11, 2018.
16. Alluxio Inc. Alluxio overview, May 2019. `https://docs.alluxio.io/os/user/stable/en/Overview.html`.
17. Ewa Deelman, Dennis Gannon, Matthew S. Shields, and Ian J. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Comp. Syst.*, 25(5):528–540, 2009.
18. Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, and Miron Livny. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.
19. Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, and Ian Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.
20. Weiwei Chen and Ewa Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *8th IEEE International Conference on E-Science*, pages 1–8, 2012.
21. Nicholas Hazekamp, Nathaniel Kremer-Herman, Benjamin Tovar, Haiyan Meng, Olivia Choudhury, Scott Emrich, and Douglas Thain. Combining static and dynamic storage management for data intensive scientific workflows. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1, 2018.
22. Pegasus. Pegausus syntheticworkflows, February 2019. `https://download.pegasus.isi.edu/misc/SyntheticWorkflows.tar.gz`.
23. Xiangke Liao, Liquan Xiao, Canqun Yang, and Yutong Lu. Milkyway-2 supercomputer: system and application. *Frontiers Comput. Sci.*, 8(3):345–356, 2014.
24. Rebecca Taft, Manasi Vartak, Nadathur Rajagopalan Satish, Narayanan Sundaram, Samuel Madden, and Michael Stonebraker. Genbase: A complex analytics genomics benchmark. Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SGIMOD'14). ACM, 2014.
25. K. R. Krish, Ali Anwar, and Ali R. Butt. hats: A heterogeneity-aware tiered storage for hadoop. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 502–511, 2014.

26. Teng Wang, Suren Byna, Bin Dong, and Houjun Tang. Univistor: Integrated hierarchical and distributed storage for HPC. In *IEEE International Conference on Cluster Computing, CLUSTER 2018, Belfast, UK, September 10-13, 2018*, pages 134–144, 2018.
27. Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. Hermes: a heterogeneous-aware multi-tiered distributed I/O buffering system. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'18)*, pages 219–230, 2018.
28. Dong Dai, Forrest Sheng Bao, Jiang Zhou, Xuanhua Shi, and Yong Chen. Vectorizing disks blocks for efficient storage system via deep learning. *Parallel Computing*, 82:75–90, 2019.
29. Erica Tomes, Everett Neil Rush, and Nihat Altiparmak. Towards adaptive parallel storage systems. *IEEE Trans. Computers*, 67(12):1840–1848, 2018.
30. Shengan Zheng, Morteza Hoseinzadeh, and Steven Swanson. Ziggurat: A tiered file system for non-volatile main memories and disks. In *17th USENIX Conference on File and Storage Technologies, FAST 2019, Boston, MA, February 25-28, 2019.*, pages 207–219, 2019.