SSD-Optimized Workload Placement with Adaptive Learning and Classification in HPC Environments

Lipeng Wan*, Zheng Lu*, Qing Cao*, Feiyi Wang†, Sarp Oral†, and Bradley Settlemyer†

*Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN, US
Email: {lwan1, zhenglu, cao}@utk.edu

†Oak Ridge National Laboratory
Oak Ridge, TN, US
Email: {fwang2, oralhs, settlemyerbw}@ornl.gov

***Abstract*—In recent years, non-volatile memory devices such as SSD drives have emerged as a viable storage solution due to their increasing capacity and decreasing cost. Due to the unique capability and capacity requirements in large scale HPC (High Performance Computing) storage environment, a hybrid configuration (SSD and HDD) may represent one of the most available and balanced solutions considering the cost and performance. Under this setting, effective data placement as well as movement with controlled overhead become a pressing challenge. In this paper, we propose an integrated object placement and movement framework and adaptive learning algorithms to address these issues. Specifically, we present a method that shuffle data objects across storage tiers to optimize the data access performance. The method also integrates an adaptive learning algorithm where real-time classification is employed to predict the popularity of data object accesses, so that they can be placed on, or migrate between SSD or HDD drives in the most efficient manner. We discuss preliminary results based on this approach using a simulator we developed to show that the proposed methods can dynamically adapt storage placements and access pattern as workloads evolve to achieve the best system level performance such as throughput.**

I. Introduction

Developing resilient and high-performance storage solutions for HPC applications remains a critical challenge given the demand for both capability and capacity in the face of potential disk drive failures [1], [2]. Therefore, it is crucial to come up with effective data placement mechanisms in a heterogamous environment based on the nature of workloads, as well as the properties of the underlying hardware such as their network topology and bandwidth. On the other hand, both the workloads and the underlying hardware evolve over time, so it requires algorithms have to be designed to take these changes into their consideration.

One particular trend that has emerged in recent years is the use of SSD drives in storage solutions to provide *premium* services, as the reading and writing speed for SSD drives are typically much faster compared to hard drives [3]. On the other hand, SSD drives have different failure patterns where repeated reading and writing operations of the same address blocks will cause the drives to fail [4]. Furthermore, SSD drives are also limited in capacity, meaning that it is not yet practical to use them to completely replace conventional hard disks. Therefore, how to integrate SSD drives successfully into the design of storage solutions becomes a challenge that modern HPC applications have to handle.

Existing algorithms to integrate SSD drives can already be found in the literature [5], [6]. These methods, while effective, are largely based on heuristic algorithms that are either developed in isolation with the runtime workload, or are based on static assumptions on the workload patterns, making them unsuitable when the underlying workloads and demands change over time.

To address such drawbacks, we present a holistic approach where we aim to develop a framework that adaptively classifies the popularity of data objects, adjusts their placement in storage units by moving them between slower HDDs and faster SSDs, and fulfills the needs of users with regard to their reading/writing/backup operation requirements. Formally, our developed algorithm makes the following assumptions. First, we assume that the storage hardware consists of both slower HDD drives and faster SSD drives. Second, the user may have their own rules for the placement of data, such as, a particular data object must be stored on HDDs with three copies available for a certain period of time, among others. Therefore, such rules must be properly fulfilled during runtime.

Based on these assumptions, our proposed method makes two contributions: first, it proposes a Markov-chain based classification model to predict whether data objects will be accessed frequently in the future based on their historical access records; second, it develops an integrated data placement engine that is based on linear programming for fulfilling the requirements of throughput and reliability from the users. We next describe these two contributions separately.

In the first contribution, we classify data objects based on their access patterns, including both their access frequencies and the particular workload that accessed them, so that we can determine those objects that are most likely to be accessed frequently in the future. Our method is based on training a Markov chain model, and once such predictions are made,

978-1-4799-5671-5/14/$31.00

we move those frequently accessed objects to SSD drives under the constraint that the moving cost does not exceed the predicted savings.

In the second contribution, we consider the challeng[illegible] fulfilling the users' rules on data placements, such as their preferences on where to place the objects. To this end, we develop a data placement engine that takes users' rules and the performance variation between storage units, such as the bandwidth and delays between two HDDs or SSDs, as input, and generates a satisfying solution, if any, as the output. The theoretical foundation of this engine stems from linear programming, where we allow numerical methods to be adopted to find solutions. For example, if the user specifies that no two copies of a data object should be on the same rack, such a requirement should be guaranteed in the solution.

The rest of this paper is organized as follows. We describe the related work in Section II. The design is described in Section III. The performance evaluation is given in Section IV. We provide conclusions in Section V.

II. Related Work

In this section, we survey several existing works related to our paper. We classify these existing works into two categories. The first category consists of existing works on data placement algorithms for distributed storage systems, while the second consists of those on hybrid storage systems that aim to leverage SSD drives to improve data access performance.

As large-scale distributed storage systems have been extensively used in the HPC area, the problem of distributing several petabytes of data among hundreds or thousands of storage devices becomes more and more critical. To address this problem, many data placement algorithms have been proposed. For instance, Distributed Hash Tables (DHTs) have been used to place and locate data objects in P2P systems [7], [8], [9]. Another replica placement scheme called chain placement was also proposed and applied to some P2P and LAN storage systems [10], [11], [12]. Honicky and Miller presented a family of algorithms named RUSH [13] that utilizes a mapping function to evenly map replicated objects to a scalable collection of storage devices, so that it can support efficient additions and removals of weighted devices.

To address the reliability and replication issues of the RUSH algorithm, Weil et al. proposed a scalable pseudo-random data distribution algorithm named CRUSH [14]. Besides optimally distributing data to available resources and efficiently reorganizing data after adding or removing storage devices, CRUSH exploits flexible constraints on replica placement to maximize data safety in the case of hardware failures. Specifically, CRUSH allows the administrator to assign different weights to storage devices so that the administrator can control the relative share of data each device is responsible for. However, the device weights used in the CRUSH algorithm only reflect the capacities of storage devices, therefore, the CRUSH algorithm may not be effective anymore for hybrid storage systems consisting of both SSD and HDD devices, as these two kinds of storage devices have totally different performance characteristics.

Recently, efforts have been made to combine SSD and HDD drives together to construct hybrid storage systems. In such systems, SSDs are either used for caching purposes, or used as more independent storage devices. For example, Srinivasan et al. designed a block-level cache named Flashcache [15] between DRAM and hard disks using SSD devices. Zhang et al. proposed iTransformer [16] which exploits a small SSD to schedule requests for the data on disks so that high disk efficiency can be achieved. SieveStore [17] adopts a selective caching approach in which the accesses of each block are tracked and the most popular block is cached in SSD device. In the second approach, SSDs are more independently used. Chen et al. designed and implemented a high performance hybrid storage system named Hystor [18], which identifies data blocks that either can result in long latencies or are semantically critical on hard disks, and store them in SSDs for future accesses. In order to prolong the service life of SSDs devices, Ren et al. proposed I_CASH [19] to reduce random write traffic to SSDs. Specifically, I_CASH is an approach that exploits the spacial locality of data accesses, and only store those seldom-changed data blocks on SSDs. Finally, ComboDrive [20] concatenates SSD and HDD into one address space via a hardware-based solution, so that certain data on HDD can be moved into the faster SSD space.

There are two main differences between existing works on hybrid storage systems and our approach: first, most existing works on hybrid storage systems only consider how to improve the utilization of SSD drives, but they have ignored the reliability and replication issues in HPC environments; second, existing works have not considered the dynamic nature of workflows, a nature that makes continuous training and learning necessary. In our approach, we fully consider these issues, and our method provides up-to-date predictions on popular data blocks, so that we can store critical data on SSDs well in advance.

III. Data Placement Algorithm Design

In this section, we introduce the design of the data placement algorithm. We first present the problem formulation. Then, we present an overview of our algorithm architecture. Finally, we present a detailed description of its components and related algorithms.

A. Problem Formulation

The core problem is formulated as follows. Given a set of storage units represented by HDDs and SSDs as the hardware platform, our task is to find a data placement that 1) satisfies users' polices on data placements, and 2) maximizes the throughput for users' reading/writing/backup operations in their application workloads. This problem is challenging due to: 1) we don't have complete knowledge on future access patterns to data objects due to the dynamics of workloads, 2) user policies can be highly heterogeneous and may change over time. Therefore, if we model such a problem as an

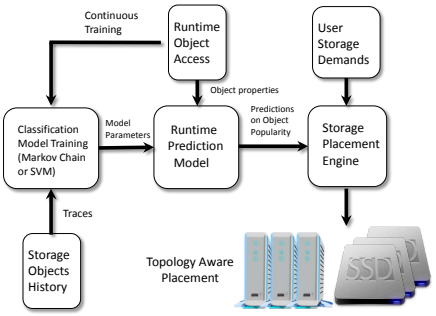

Fig. 1. The Design Architecture

optimization problem, its solution is from a very large search space, such that it is very hard to always reach the optimal configuration when something changes, even such change could be slight. Furthermore, given that the users' request may change frequently, we would like to be able to re-use previous calculation results as much as possible, so that we can avoid the re-calculation when a similar scenario is met in practice.

B. Architecture Overview

We first present the design architecture of our data placement algorithm. For this design, we consider the I/O workload from user applications to include both reading and writing operations. All I/O workload are generated to access data objects, which are minimal storage units in object-based storage systems. In practice, a large file can be divided into multiple data objects which will be stored on single or multiple object-based storage devices (OSDs). Note that the writing operations may be dominating for certain I/O workload, such as periodic checkpoint [21]. The I/O workload from user applications may also change over time, therefore, the solution should adapt to the dynamic nature of the I/O workload.

Figure 1 shows the overall architecture, where the whole procedure works as follows: the first core component, the classification model, is trained based on the access history of data objects. In our current work, we concentrate on the historical access frequency of data objects, while we leave exploiting the access pattern (sequential/random read or write) to improve data placement performance as our future work. After training, it provides parameters for the runtime prediction model that are used to predict the access popularity of data objects in the future. Specifically, the predictions decide if an object is going to have "recurring" or "non-recurring" accesses, based on its history of accesses, as well as the particular workload that accessed them. Such predictions are then used, together with user demands, as the input for the storage placement engine, whose goal is to generate an optimized placement of data objects to storage units so that the overall system level performance on access delay and bandwidth can be improved. Finally, the runtime object access traces are also used as input for the classification model for continuous training purposes, and keeping the parameters up-to-date.

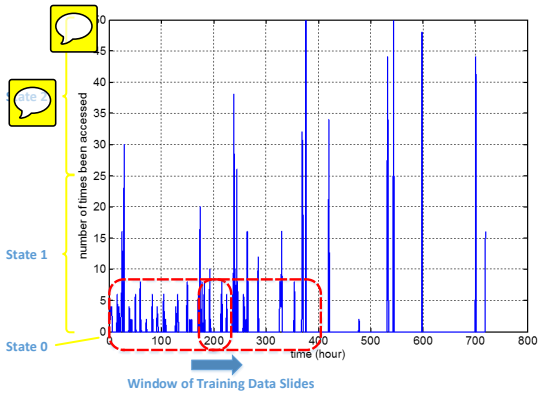

Fig. 2. Traces of Data Object Accesses in Frequency

C. Markov Chain based Workload Classification

In this section, we describe how the classification model generates model parameters based on object access traces. In our design, we adopt a Markov chain based approach.

Observe that there exists a tradeoff in the overhead of training and the prediction accuracy. Therefore, we need to achieve a good tradeoff in our design. Our approach has the following key steps. First, we assume that we can keep the access history for each data object. In reality, it may not be practical to record a long access history for each data object since in that case the storage overhead could be huge, instead, only recent access history needs to be maintained and updated periodically. Second, we model the access frequency of each data object using a discrete-time Markov chain in which each state represents a specific range of access frequency. With the access history, we can estimate the parameters of the Markov chain model. Third, by calculating the stationary distribution of the Markov chain, we can predict the likelihood for the access frequency of each data object to stay within certain ranges in the long run. Finally, we rank each data object based on the weighted sum of the stationary distribution, where the weights are chosen according to the specific range represented by each state in the Markov chain. A higher rank of the data object indicates that it is more appropriate and efficient to be moved or placed into low-latency, high-bandwidth drives such as SSDs. We next describe these steps in more detail.

1) Collection of Access History of Data Objects: Fig. 2 demonstrates the access frequency (here the "access" includes both read and write operations) of a data object during one month from the LASR traces [22], which include I/O activities of benchmark applications for the SEER project, which observes users' file access patterns across storage networks. The X axis of Fig. 2 is the range of one month time that has been divided into 720 time periods (each period is 1 hour). The Y axis represents the number of times the data object has been accessed during each time period. As the storage overhead for maintaining the entire access history of each data object is not cost-effective, we only maintain recent access history of each data object and use such access history to build a Markov chain model to predict the future access frequency. As shown in Fig. 2, only the access history in the dotted window is used

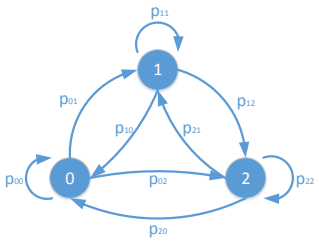

Fig. 3. Transition Diagram of Markov Chain

to train the prediction model, where the window will slide with time so that we can implement online prediction for data objects access frequency.

2) Markov Chain Predication Model: With the access history of each data object, we build a Markov chain model to predict the future access frequency of data objects. First, we need to determine how many states the Markov chain should have and the range of access frequency each state represents. For example, as shown in Fig. 2, if the maximum number of access times during an observation period is 50, then, for example, we can divide 50 evenly into two ranges, and build a Markov chain model that has three states: 0, $(0, 25]$, and $(25, 50]$, respectively. If during a time period, there is no access of the data object, then the Markov chain will stay in state 0. If the number of access times is larger than 0 but less than 25, then the Markov chain will stay in state 1, and so on. The transition diagram of the Markov chain is shown in Fig. 3.

Second, we transform the access history to the state transition sequence of the Markov chain based on the specific range each state represents. For example, after transformation the state transition sequence of access history shown in Fig. 2 is: 1,1,1,1,1,0,0... Based on this state transition sequence, we can estimate the transition probabilities between every two states and construct the transition matrix of the Markov chain shown below:

$$\mathbf{T} = \begin{pmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{pmatrix} \tag{1}$$

According to the properties of Markov chain, we have:

$$\lim_{n\to\infty} \mathbf{T}^n = \begin{pmatrix} \pi_0 & \pi_1 & \pi_2 \\ \pi_0 & \pi_1 & \pi_2 \\ \pi_0 & \pi_1 & \pi_2 \end{pmatrix} \tag{2}$$

in which $\boldsymbol{\pi} = [\pi_0, \pi_1, \pi_2]$ is called the stationary distribution of the Markov chain. We can simply calculate $\boldsymbol{\pi}$ through computing a normalized multiple of a left eigenvector $\mathbf{E}$ of the transition matrix $\mathbf{T}$ with an eigenvalue of 1:

$$\boldsymbol{\pi} = \frac{\mathbf{E}}{\sum_i e_i} \tag{3}$$

where e_i is the i-th element of eigenvector $\mathbf{E}$. Since the stationary distribution $\boldsymbol{\pi}$ reflects the probabilities that each state of Markov chain will be visited in the future, which can be used to predict the access frequency of each data object.

N	Total number of storage drives
M	Total number of data objects
cs_i	Capacity of storage drive i
ds_i	Size of data object i
f_i	Predicted frequency of access for data object i
b_{ij}	Bandwidth for the link connecting storage drives i and j
at_i	Average throughput for storage drive i
e_{ij}	Whether data object i is stored on storage drive j (0 or 1)
cp_i	The minimum number of copies for data object i

TABLE I
NOTATIONS OF SYMBOLS

Based on the predicted access frequency in the future, we rank the data objects so that we can determine which data object should be placed or moved to SSD drives. Note that, however, even if the calculated stationary distribution tells us state 1 will be visited with higher probability than state 2, to rank the importance of the data object, we must consider that state 2 represents a higher access frequency. Therefore, we use a weighted sum of the stationary distribution to rank the importance of the data objects, where the weights are defined by values that are proportional to the access frequency ranges that the states represent. For example, if we obtain the stationary distribution of the data object as $\boldsymbol{\pi} = [0.31, 0.56, 0.13]$, and we assign weights $[0, 10, 20]$ to the three different states, we can calculate the rank of the data object by $rank_{obj_x} = 0.31 \times 0 + 0.56 \times 10 + 0.13 \times 20 = 8.2$. We can then compare the ranks of objects, and provide input for the placement engine.

D. Finding Placements under User Polices

Once we obtain the predicted popularity of data objects, i.e., their ranks, the next step is to find an optimized placement solution such that the access latency is minimized, while satisfying users' placement policies. In this aspect, we assume that users' requests will be parametric, meaning that all requests will be embedded into equations or constraints. For example, by using the notations in Table I, a requirement on the number of redundant copies of a data object i stating that at least three extra copies must be made can be expressed as $cp_i > 3$.

Our solution to this problem is by formulating the placement problem in a mathematical optimization as follows:

We want to maximize:

$$\sum_{i \in M} f_i \times max[\forall j \in N, at_j \times e_{ij}] \tag{4}$$

subject to constraints such as:

$$\sum_{j \in N} e_{ij} = cp_i, \forall i \in M, \tag{5}$$

$$\sum_{\forall i \ \ s.t. \ \ e_{ij}=1} ds_i \leq cs_j, \forall j \in N, \tag{6}$$

In this short paper, we only give a description of an easier scenario in these equations. In this example, Equation 4 specifies that we want to find a way that assigns data objects to storage units such that the weighted access throughput by the access frequency is maximized. Note that we use the equation $at_j \times e_{ij}$ to filter those storage units where the particular data

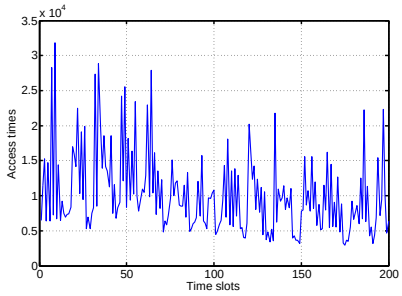

Fig. 4. Illustration of Data Access Traces Type I

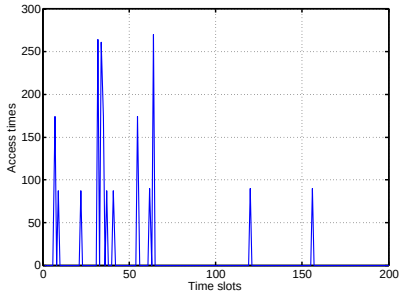

Fig. 5. Illustration of Data Access Traces Type II

object i is stored on: if i is stored on device j, we know $e_{ij} = 1$, otherwise $e_{ij} = 0$. Hence, finding the max of them translates into finding the storage unit with a copy of data object i that has the highest throughput.

On the other hand, this optimization goal is subject to two or more constraints. In this example, we only consider two of them, in Equation 5 and Equation 6, respectively. The first constraint specifies that each storage drive j should not contain more data objects than its capacity, while the second constraint specifies that the number of duplicate copies of a data object should be set according to the users' placement polices. If a data object does not need to have duplicates, its cp value is set to 1 by default.

In more complex scenarios, the users may have additional constraints. As we mentioned earlier, such constraints are parameterized, meaning that we can easily add more constraints to the formulation above. Finally, even the optimization goal can be adjusted. For example, if we only want to minimize the access delay rather than the throughput, we can change the Equation 4 accordingly.

Based on this formulation, we notice that the optimization is reduced to a linear programming problem where we can use numerical methods to recalculate its solution periodically in the placement engine. Note that the engine is operating independently of the rest of the system. Doing so does not require the engine to be tightly integrated, so that we can easily change the engine's implementation as needed, which gives us additional flexibility.

IV. System Evaluation

In this section, we systematically present the evaluation of the proposed algorithm. We first present a study of the traces of data object accesses, based on which we evaluate the performance of our learning algorithm by replaying them. We use a long-term I/O traces, LASR traces [22], which were collected at system-call level. We track the access frequency of different files during their lifetime. Specifically, we divide the time span into hundreds of time slots and each of which has same length. We then count how many times each file has been accessed during each time slot. In the LASR traces, we eliminate those files which have only been accessed several times during their lifetime (the access of these files almost has no impact on the performance of the storage system) and focus on the remaining ones (around 1,700 files) which are more frequently accessed. By analyzing the access of these frequently accessed files, we find out that these files can be roughly put into two categories according to their access patterns. The first category contains files that have constant access patterns. Files in this category have been frequently accessed during their whole lifetime, without too much difference between the maximum and minimum access periods. Fig. 4 shows a typical file falls into this category. The learning algorithm, especially the Markov chain approach can achieve a higher level of accuracy for this kind of files. The second category is files with a bursty access pattern. Files in this category have only been accessed at very few time slots, but the access counts for those time slot can be very large. Fig. 5 shows a typical file falling into this category. For files belong to second category, it is pretty hard for any learning algorithm including Markov chain approach to train a model to accurately predict their future access frequency.

To save the evaluation time, we did not use the entire LASR traces, instead we randomly select traces of 40 different files from the dataset containing both categories we mentioned above. For each trace we use the first half as the training data to train our Markov prediction model while use the other half as the testing data. As illustrated in Fig. 6, the bars represent the future access frequency of the 40 different files which are extracted from the testing dataset. Since we only have limited SSD storage space, our goal is to store the files that will be most frequently accessed in the future on SSD devices to improve the average data access throughput. For example, if we can only put 10 of 40 files on SSDs, as shown in Fig. 6, the light-colored bars illustrate the files predicted by our Markov model that are required to be placed on SSD devices. From the results we can observe that, 7 of 10 files that have the highest future access frequency have been chosen. It is true that our approach missed 3 files which should be put on SSDs, but that is because the accesses of these 3 files are very bursty (as they fall into the second category we mentioned above), which means they were often accessed many times during a very short period.

We next choose random selection approach as baseline and compare the average read throughput achieved by our Markov prediction model with that achieved by random object selection. Here the random object selection means that we randomly choose several data objects (files) and put them on SSD devices. The number of data objects that can be

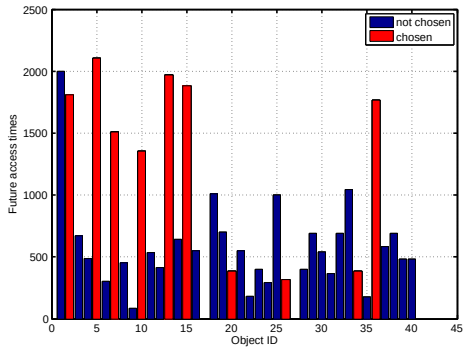

Fig. 6. Future Access Times of Data Objects

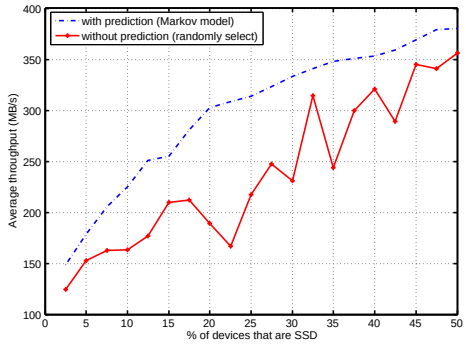

Fig. 7. Average Throughput v.s. Percentage of Objects Stored on SSDs

placed on SSD devices is also limited. For example, in the simulation, we vary the number of data objects that can be put on SSD devices from 2.5% to 50%. Besides, we set the read throughput of SSD and HDD devices as 550 MB/s and 120 MB/s respectively, consistent with typical datasheets provided by manufacturers of storage devices [4]. As shown in Fig. 7, our object selection approach can achieve higher average read throughput than random selection, demonstrating the effectiveness of our proposed approaches.

V. Conclusions

In this paper, we presented a study of developing a hybrid configuration (SSD and HDD) for storage needs of HPC environments. Specifically, we proposed an integrated object placement framework with adaptive learning algorithms. The method placed data objects with considering both the popularity of the data and the capability of different storage units, so that the data access performance can be optimized. The method also integrated a Markov chain algorithm where real-time classification is employed to predict the popularity of data object accesses, so that they can be placed on, or migrate between SSD or HDD drives in the most efficient manner. Our preliminary results based on realistic data traces demonstrate that this approach is highly promising, and achieves better performance than benchmark methods such as pure random selections.

VI. Acknowledgement

We would like to thank the anonymous reviewers for their insightful comments. The work reported in this paper was sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy, and by the National Science Foundation grant 0953238.

References

[1] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing Variability in the IO Performance of Petascale Storage Systems," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–12.

[2] B. Zhu, G. Wang, X. Liu, D. Hu, S. Lin, and J. Ma, "Proactive Drive Failure Prediction for Large Scale Storage Systems," in *MSST*, 2013, pp. 1–5.

[3] S. Park and K. Shen, "A Performance Evaluation of Scientific I/O Workloads on Flash-based SSDs," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–5.

[4] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1. ACM, 2009, pp. 181–192.

[5] H. Wang, P. Huang, S. He, K. Zhou, C. hua Li, and X. He, "A novel I/O scheduler for ssd with improved performance and lifetime," in *MSST*, 2013, pp. 1–5.

[6] Y.-S. Lee, S.-H. Kim, J.-S. Kim, J. Lee, C. Park, and S. Maeng, "OSSD: A case for object-based solid state drives," in *MSST*, 2013, pp. 1–13.

[7] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 149–160.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-addressable Network," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 161–172.

[9] M. Cai, A. Chervenak, and M. Frank, "A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, ser. SC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 56–67.

[10] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '01. New York, NY, USA: ACM, 2001, pp. 188–201.

[11] E. K. Lee and C. A. Thekkath, "Petal: Distributed Virtual Disks," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS VII. New York, NY, USA: ACM, 1996, pp. 84–92.

[12] J. MacCormick, N. Murphy, M. Najork, C. A. Thekkath, and L. Zhou, "Boxwood: Abstractions As the Foundation for Storage Infrastructure," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI '04. Berkeley, CA, USA: USENIX Association, 2004, pp. 105–120.

[13] R. J. Honicky and E. L. Miller, "Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution," in *Proceedings of the 18th International Parallel & Distributed Processing Symposium (IPDPS 2004)*, 2004.

[14] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[15] M. Srinivasan, P. Saab, and V. Tkachenko. Flashcache. [Online]. Available: https://github.com/facebook/flashcache/

[16] X. Zhang, K. Davis, and S. Jiang, "iTransformer: Using SSD to Improve Disk Scheduling for High-performance I/O," in *IPDPS*. IEEE Computer Society, 2012, pp. 715–726.

[17] T. Pritchett and M. Thottethodi, "SieveStore: A Highly-selective, Ensemble-level Disk Cache for Cost-performance," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 163–174.

[18] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 22–32.

[19] Q. Yang and J. Ren, "I-CASH: Intelligently Coupled Array of SSD and HDD," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, ser. HPCA '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 278–289.

[20] H. Payer, M. A. Sanvido, Z. Z. Bandic, and C. M. Kirsch, "Combo Drive: Optimizing Cost and Performance in a Heterogeneous Storage Device," in *Proceedings of the 1st Workshop on integrating solid-state memory into the storage hierarchy*, ser. WISH '09, 2009.

[21] N. Naksinehaboon, Y. Liu, C. Leangsuksun, R. Nassar, M. Paun, and S. L. Scott, "Reliability-aware approach: An incremental checkpoint/restart model in hpc environments," in *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*. IEEE, 2008, pp. 783–788.

[22] G. Kuenning. LASR traces. [Online]. Available: http://www.lasr.cs.ucla.edu/seer/seer_traces.html