SLRL: A Simple Least Remaining Lifetime File Eviction policy for HPC multi-tier storage systems

Louis-Marie Nicolas, Luis Thomas, Yassine Hadjadj-Aoul, Jalil Boukhobza

SLRL: A Simple Least Remaining Lifetime File Evicition policy for HPC multi-tier storage systems

Louis-Marie Nicolas
nicolas.louismarie@gmail.com
ENSTA Bretagne - Lab-STICC, CNRS, UMR 6285
Brest, France

Luis Thomas
luis.thomas@ensta-bretagne.org
ENSTA Bretagne - Lab-STICC, CNRS, UMR 6285
Brest, France

Yassine Hadjadj-Aoul
yassine.hadjadj-aoul@irisa.fr
IRISA - Univ. Rennes 1
Rennes, France

Jalil Boukhobza
jalil.boukhobza@ensta-bretagne.fr
ENSTA Bretagne - Lab-STICC, CNRS, UMR 6285
Brest, France

Abstract

HPC systems are composed of multiple tiers of storage, from the top high performance tier (high speed SSDs) to the bottom capacitive one (tapes). File placement in such architecture is managed through prefetchers (bottom-up) and eviction policies (top-down). Most state-of-the-art work focus on the former while using algorithm flavors of LRU, LFU and FIFO for the latter. LRU was for long considered the best choice. However, recent studies has shown that the simplicity of FIFO could make it more scalable than LRU because of metadata management, and thus more adequate in several cases. In this paper, we propose a new eviction policy based on predicted files lifetimes. It is comparable to FIFO in terms of metadata overhead and simplicity (thus scalability), while giving a hit ratio comparable to LRU (or even 10% better for some tested traces). We also propose a naive multi-tier heterogeneous storage simulator implementation to evaluate such policies.

CCS Concepts: • **Information systems** → Information storage systems; *Storage management*; **Hierarchical storage management**; **Information lifecycle management**;

Keywords: Data placement, Multi-Tier Storage, File lifetime, High Performance Computing, Simulation

ACM Reference Format:
Louis-Marie Nicolas, Luis Thomas, Yassine Hadjadj-Aoul, and Jalil Boukhobza. 2022. SLRL: A Simple Least Remaining Lifetime File Evicition policy for HPC multi-tier storage systems. In *Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS '22), April 5, 2022, RENNES, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3503646.3524297

CHEOPS '22, April 5, 2022, RENNES, France

ACM ISBN 978-1-4503-9209-9/22/04…$15.00
https://doi.org/10.1145/3503646.3524297

1 Introduction

HPC storage systems have to meet increasingly high performance and storage requirements [30][39][21]. To do so, they have multiple storage technologies at their disposal. Technologies such as NVM [16][5] and SSDs[4] offer high performance but their high cost in currency per gigabyte and thus low capacity prevent them from being used as the only form of storage (in most cases). Instead, companies are using HDD [25] for their moderate cost and higher capacity, with a smaller amount of faster SSD or NVM being used as a cache or as a complementary storage (horizontal integration) [4]. These storage devices are eventually completed with tapes [20], offering long-term low-cost but high-latency technologies as a third storage tier.

To move files from capacitive tiers to high performance tiers, the systems used generally rely on prefetchers [33] [38] [37]. Several policies were introduced in state-of-the-art studies. They mainly rely on monitoring I/Os [24] [3], analyzing patterns and classifying file/objects/blocks before prefetching them to higher tier memories.

However, even if those prefetching algorithms are very sophisticated, the focus is not on how files should be removed from the efficient tiers to capacitive ones when they are getting full. Indeed, traditional cache eviction strategies such as LRU (or LFU) and FIFO are used in this case, with LRU being considered as the most efficient one. However, a recent work by IBM Research [13] observed that LRU was not necessarily the best for modern I/O traces (for instance object stores). This is because the metadata management complexity that grows very high with modern I/O workloads. While from a hit rate point of view LRU-like algorithms are still a good choice, it can be beaten by FIFO when considering cost. Indeed, FIFO is very simple and does not necessitate complex meta data management.

In this paper, we propose SLRL, a **S**imple **L**east **R**emaining **L**ifetime file eviction policy. SLRL hits a middle ground between FIFO and LRU. Indeed, it is as simple as FIFO in terms of metadata usage, while being comparable to LRU in terms of hit rate. SLRL uses file lifetimes predicted based on the study in [34]. The different components of file paths were used in order to predict file lifetimes at their creation. SLRL simply evicts the files according to the predicted lifetimes in a queue. As such, like FIFO and unlike LRU, there is no need to update a queue at every I/O with timestamps, and the policy requires a minimal amount of memory and monitoring to run.

We evaluated the performance of the proposed eviction policy on an open-source simple multi-tier heterogeneous discrete event storage simulator which represents the second contribution of this paper [1]. This simulator was inspired from StorageSIM [32], but it has been fully redesigned from the ground up in a way that makes adding new policies and datasets for workload generation easier. Results show that SLRL efficiency varies depending on the trace used. We observed a performance slightly lower than LRU and FIFO for traces showing a very low locality. However, SLRL performed better than both FIFO (40%) and LRU (10%) when the locality is higher.

This paper is organized as follows. Section 2 gives some background. Section 3 describes our contribution. Section 4 discusses experimental results. Section 5 gives some related work and Section 6 concludes the paper.

2 Background

2.1 HPC storage hierarchies

Scientific applications such as atomic and weather simulations, geoscience and genome sequencing generate petabytes of data. While the computing power of HPC servers is growing higher, main memory performance is hardly keeping the pace [28], not to mention storage systems. Indeed, the performance gap between the main memory and the non-volatile storage memory widens. Cache and CPU registers are several orders of magnitude faster than the main memory (DRAM), which itself is up to several orders of magnitude faster than the main storage (usually HDD) [17].

Caching and prefetching strategies have the common objective to bridge part of this performance gap. For instance, one may use Flash-based drives (SSD) as cache between a slower type of storage (HDD) and a faster type of memory (DRAM) [40] [4]. In this case, there is a need for a prefetcher to select data to move from the HDD tier to the SSD one. Moreover, there is a need for an eviction policy whose responsibility is to move, when necessary, unneeded data from the SSD tier to the HDD one.

2.2 File lifetime prediction

In previous work [34], the authors used the access path of a file to accurately predict when a file will last be accessed. They used a single Convolutional Neural Network with a custom loss function in order to reduce the computing power necessary for a prediction. To achieve this, they designed the custom loss function to optimize the ratio between accuracy and under-estimations. The intuition behind the design of the custom loss function stems from multi task learning[14], they used the weighted average of the two loss functions used in another previous work [27]. By doing so, they increased the accuracy of file lifetime predictions while maintaining a low level of under-estimations.

As one can see in Figure 1, the parallel distributed filesystem Lustre [10] instances have 2 storage servers which contain different media types. Applications are connected to the Lustre Storage Server through the High Performance Data Network. The metadata server contains metadata about the other Lustre file systems and is connected to them with a specific management network with lower latency and bandwidth. The Robinhood Policy Engine [22] represents the software solution used to create migration policies between the different tiers. It reads metadata changes from the metadata server with the changelog reader. An administrator can then create a policy by periodically checking the state of the file system through the Robinhood database.

The file lifetime prediction module was integrated into Robinhood and filters only the "File creation" events. The paths of the files are processed to generate a prediction of their lifetimes. These lifetime values are used by our eviction policy presented in the next Section.

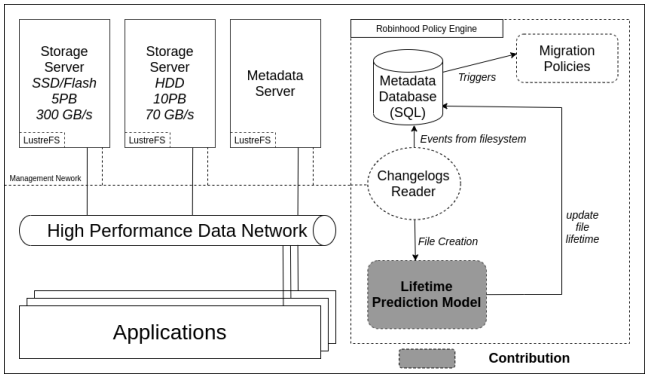

Figure 1. File lifetime prediction module [34]

3 SLRL: a Simple Least Remaining Lifetime Policy for File Eviction

In this section, we first present SLRL, the new eviction policy. Then, we describe a simple multi-tier heterogeneous discrete event storage simulator as a tool to easily implement and compare eviction policies.

[1] https://github.com/ShinySilver/Storage_Simulation_for_File_Placement_in_HPC

3.1 SLRL policy

In this paper, the term lifetime refers to the time elapsed between the creation of a file and its last access. As described earlier, SLRL is based on a file lifetime prediction model that was described in a previous work. This model is able to predict the lifetime of a file based on its absolute path [34] with an 98% accuracy.

Since an absolute path is known at the creation of a file, SLRL needs only to monitor file creation events, and has, at a given time, at its disposal a predicted remaining lifetime, positive or negative, for every stored file. The remaining lifetime is positive when it is still expected that the file will be accessed in the future, while a negative lifetime means that the file will no longer be accessed. A lifetime can be negative if the file has expired but there is still space in the storage tier (so eviction has not been performed). As such, SLRL policy places newly created file identifiers in a queue at a position that depends on their remaining lifetime prediction. File identifiers are placed in this queue once and for all, until the storage tiers are full and the policy starts evicting files from this queue in order, that is the least remaining lifetime first.

SLRL is comparable to FIFO in the sense that files are placed in a queue at their creation time (the only metric monitored for both algorithms). However, the main difference is that in SLRL, files are not necessarily inserted at the last position but at varying positions depending on their predicted remaining lifetime.

Compared to LRU, the main difference is that the queue of SLRL does not need to be updated every time a file is accessed: the only events to monitor are the file creations. This trait is shared with FIFO, and the fact that the queue does not need to be updated every time a file is accessed is a reason why FIFO was preferred over LRU for some workloads in the IBM study [13]. Moreover, our policy shares with FIFO the fact that it does not need to monitor every time a file is used, which is a strong point since this monitoring can bring a huge performance impact [29]. Finally, SLRL shares with LRU the fact that it considers file properties in the queue management. For LRU, this is done through access monitoring and queue update, and for SLRL this is done thanks to the information extracted from the file path (lifetime).

3.2 Simple multi-tier heterogeneous storage discrete event simulator

In addition to our SLRL policy and in order to easily compare it with other policies of the state-of-the-art, we designed a naive multi-tier storage simulator for file placement policy evaluation. It is inspired from state-of-the-art work [32], with a code base remade from the ground up to enable easier prototyping and modification of file placement policies.

Our storage simulator relies on the Simpy framework. That is a process-based discrete-event simulation framework

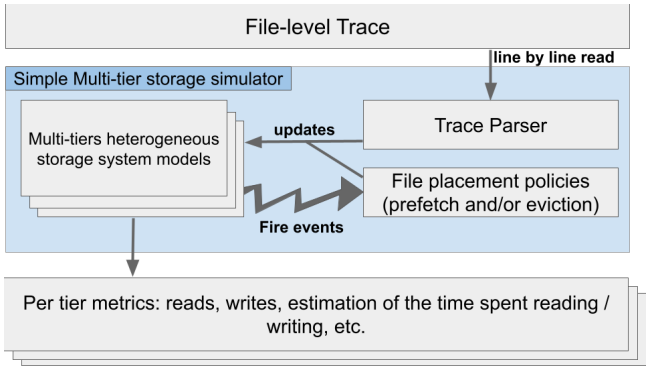

Figure 2. Simple storage simulator

based on standard Python [2]. It takes two sets of inputs. First the description of the storage tiers configuration in terms of throughput, size and latency. Second, it takes a file-level system trace containing file creation, deletion, reads and writes, with sizes and offsets (see Figure 2 for an example).

The simulator then parses the trace line by line, updating a storage model keeping track of the number of reads, of writes and the free space on each tier, and eventually firing events, namely "file creation event", "file removal event", "file read event", "file write event" and "tier nearly full event". As such, file placement policies can easily be implemented as event listeners of the storage model events.

When a policy calls for a file migration, the storage tiers model is updated with the new files location while maintaining the number of reads and writes for every tier.

At the end of a simulation, the simulator outputs, for each tier, the number of reads, writes, migrations from this tier, migrations to this tier, execution times for read and write operations. All these stats can be produced optionally while taking into account migration operations.

One way to compare the efficiency of eviction policies is to compare the amount of I/Os done on the high performance tier (without taking into account I/Os related to migrations). Indeed, the best policy should be able to maximize the amount of I/Os achieved on the high performance tier, and minimize the ones realized on the capacitive tier. This was the objective of the performance evaluation methodology described in the next Section.

4 Experimental Evaluation

4.1 Methodology

IBM Research released a 7 day object-store trace on the SNIA IOTTA repository [3] with the intention of making it a public testing ground for HPC eviction policies. We decided to use their dataset to compare our policy with FIFO and LRU. Prefetching policies are out of the scope of this paper, so

[2] https://simpy.readthedocs.io/en/latest/
[3] http://iotta.snia.org/

we did not use them in this study. When a file is created, it is stored on the high performance tier, and once this tier is overloaded the eviction policy is ran. When a file is read on a lower performance tier, it is re-loaded into the high performance tier.

4.1.1 Metrics evaluated. In this context, our comparison is simply based on the number of I/Os performed on the HDD tier (2nd tier). The lowest is the better as this means that most I/Os were done successfully on the SSD tier (1st tier). Thus we measured the miss rate on the SSD (hits on the HDD). When the number of I/Os on the HDD is high, it means that a lot of I/O operations targeted files that were evicted while still active.

4.1.2 Experiments performed. Like most large size open access traces, there were no path names available in the IBM trace. This can be justified by anonymization issues. As such, we could not run the lifetime prediction model in [34]. However, evaluating such a model is out of the scope of this paper, here we focus on the eviction policy. As a consequence, we evaluated the eviction policies by using real file lifetimes as inferred from the traces. We injected errors to these lifetimes to simulate lifetime prediction accuracy. As a reminder, in our previous work our models managed to predict lifetimes with an accuracy of 98% of the time meaning that this proportion of lifetimes were predicted in the right order of magnitude [34].

Experiment 1: We first evaluated the eviction policies with the original IBM trace on a half day basis. During this evaluation, we measured that most files in our trace subset were created once and never used again. This trace, while being a real use case, did not prove to show a high access locality. In order to test a more exhaustive set of traces, we have performed a second set of experiment.

Experiment 2: In a second part of our evaluation, we modified the IBM trace so that more files would be reused after their creation. To do so, we added randomly 20% "read" operations on already existing files to increase data locality.

For each experiment, we made two different runs with SLRL for each trace: one with the real lifetimes (no error in the prediction, named SLRL perfect lifetime prediction), and one where real lifetimes were replaced with a value picked randomly in one order of magnitude around the real value (to simulate an error in the prediction as estimated in [34], it was named SLRL noisy prediction in the Figures).

We also used 3 different sizes for the high performance tier to evaluate the effect of the pressure on the SSD tier according to the eviction policy.

4.2 Results

Experiment 1: As one can observe in Figure 3, FIFO and LRU show similar performance as the number of reads performed on the HDD tier are similar. This confirms the conclusions in [13] about the performance of FIFO for this particular trace. We also observe that SLRL performance is comparable to LRU and FIFO even it performs slightly more reads on the HDD tier. SLRL with a perfect prediction 4.8% less efficient than LRU and FIFO in case the prediction is perfect, while it is 7% less good in case of noisy prediction. The main reason behind such similar performance for the three strategies is the lack of locality. Indeed, after analyzing the trace, we have observed that most of the files were created and never accessed again, which makes the cache eviction strategy having a too small impact on the overall performance. This does not mean that the cache is inefficient, but that the performance does not depend highly on the eviction policy (a cache acting more as a buffer). Finally, we also observe that the difference between the 4 strategies decreases when the pressure on the SSD tier increases (its size decreases). This means that a correct dimensioning of the high performance tier is crucial to make profit of good eviction strategies.

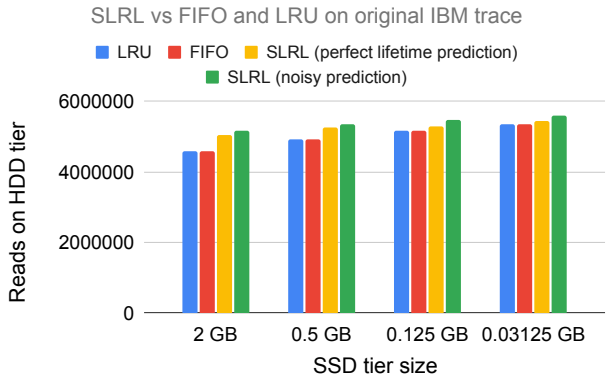

Figure 3. Number of reads on the HDD tier, **the lower the better**: SLRL vs LRU and FIFO for the original IBM trace

Experiment 2: In the second experiment (see Figure 4), as explained in the previous section, we have inserted additional read operations to the trace in order to increase the locality and rerun the trace with all policies. This made the cache more efficient as files are re accessed. That is why we have used larger sizes for the SSD tier (see Figure 4) in order to emphasize the cache effect. We can observe that for the three tier configurations used, SLRL with perfect predictions performed better than both FIFO and LRU. Compared to FIFO, SLRL with perfect prediction was 40% better while it was 10% better than LRU on average for the tested configurations. One can also observe that SLRL with a noisy prediction hits a middle ground between LRU and FIFO. This means that even if the predictions are not that good, this strategy could still perform better than FIFO for this workload as it presents more locality than in experiment 1 (modeling the efficiency of SLRL with regards to the accuracy of the lifetime prediction is a work we will perform in the future).

Even if the subset of traces used is not extremely exhaustive, one can see that LSRL is at least comparable to LRU

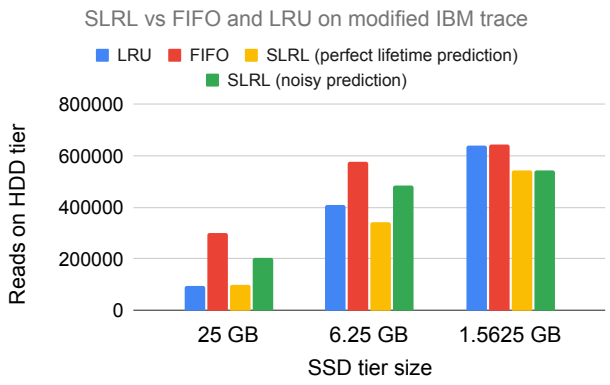

Figure 4. Number of reads on the HDD tier, **the lower the better**: SLRL vs LRU and FIFO for the modified IBM trace

and FIFO and behaves some times better. This makes it an interesting alternative to consider when one would like a strategy as simple as FIFO but still that could behave well in case of high locality.

5 Related work

Cache eviction policies are a central point in system design and more particularly in HPC storage hierarchy. While the objective of those policies is to decide about the cache content (to keep), it was mainly evaluated through cache hits comparisons. FIFO and LRU were the main opponents for some time and several papers evaluated their relevance. In early studies, LRU was the ultimate winner by far thanks to its efficiency [2, 6, 9, 11, 36]. Most cache eviction policies are based on LRU, and some times LFU for frequency, or a combination of both [1, 12, 18, 26, 31]. When it comes to managing large caches, several studies tried to cope with the high cost of managing LRU eviction policy metadata by approximating them [8, 35]. However, in several cases, falling back to FIFO could be a relevant solution as demonstrated in [13] thanks to its simplicity. In this paper, we argue that there is a middle ground between efficient but costly LRU versus simple but scalable FIFO. We explore the use of file lifetimes inferred thanks to some frugal monitoring [34] as a third alternative.

Several storage system simulators exist in state-of-the-art work such as DiskSim [7], PFSSim [23], Flashsim[19] or OGSim[15]. While the first one is a detailed reference simulator for HDD-based storage systems, PFSsim focuses on I/O schedulers for parallel file systems and FlashSim is specific to flash-based devices. OGSsim is a heterogeneous storage system simulator in which one can mix different technologies. The objective of StorageSim[32] is to simulate two-tiers storage systems with very simple configurations. We have tried first to upgrade this simulator in order to consider several tiers and different types of input traces. However, as it was difficult to operate the upgrade because of some modularity issues, we have decided to design a brand new Open Source multi-tier storage simulator presented in this paper.

6 Conclusions and Future Work

In this paper we propose a simple eviction policy based on file lifetimes predictions. SLRL evicts files according to their learnt life time and reuses state-of-the-art work to get those lifetime. Lifetime measures were extracted thanks to a very low monitoring overhead procedure. SLRL manages as little metadata as FIFO which makes it as scalable while considering the file properties making it approaching LRU in efficiency. It operates a middle ground between those policies. We would like to carry on the experimentation to evaluate the relevance of SLRL according to the types of workload used. We will also work towards the use of file lifetimes per time window (for instance per day) for the sets of files that are reused periodically. This period could be learned and set with different granularities per file for a more refined lifetime model. Also, we will investigate the use of continuous learning for updating file lifetimes according to their usage. We hope this first work towards using file lifetimes will open the way for more sophisticated strategies based on that criteria.

References

[1] Aaron Blankstein, Siddhartha Sen, and Michael J. Freedman. 2017. Hyperbolic Caching: Flexible Caching for Web Applications. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA, USA) *(USENIX ATC '17)*. USENIX Association, USA, 499–511.

[2] Allan Borodin, Prabhakar Raghavan, Sandy Irani, and Baruch Schieber. 1991. Competitive Paging with Locality of Reference. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing* (New Orleans, Louisiana, USA) *(STOC '91)*. Association for Computing Machinery, New York, NY, USA, 249–259. https://doi.org/10.1145/103418.103422

[3] Djillali Boukhelef, Jalil Boukhobza, Kamel Boukhalfa, Hamza Ouarnoughi, and Laurent Lemarchand. 2019. Optimizing the cost of DBaaS object placement in hybrid storage systems. *Future Generation Computer Systems* 93 (apr 2019), 176–187. https://doi.org/10.1016/j.future.2018.10.030

[4] Jalil Boukhobza and Pierre Olivier. 2017. *Flash Memory Integration* (1st ed.). ISTE Press - Elsevier.

[5] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. 2017. Emerging NVM: A Survey on Architectural Integration and Research Challenges. *ACM Trans. Des. Autom. Electron. Syst.* 23, 2, Article 14 (nov 2017), 32 pages. https://doi.org/10.1145/3131848

[6] Joan Boyar, Sushmita Gupta, and Kim S. Larsen. 2012. Access Graphs Results for LRU versus FIFO under Relative Worst Order Analysis. arXiv:1204.4047 [cs.DS]

[7] John S Bucy, Gregory R Ganger, et al. 2003. *The DiskSim simulation environment version 3.0 reference manual.* School of Computer Science, Carnegie Mellon University.

[8] Zhiguang Chen, Nong Xiao, Yutong Lu, and Fang Liu. 2016. ME-CLOCK:A Memory-Efficient Framework to Implement Replacement Policies for Large Caches. *IEEE Trans. Comput.* 65, 8 (2016), 2665–2671. https://doi.org/10.1109/TC.2015.2495182

[9] Marek Chrobak and John Noga. 1999. LRU is better than FIFO. *Algorithmica* 23 (02 1999), 180–185. https://doi.org/10.1007/PL00009255

[10] Sean Cochrane, Ken Kutzer, and L McIntosh. 2009. Solving the HPC I/O bottleneck: Sun™ Lustre™ storage system. *Sun BluePrints™ Online* 820 (2009). http://nz11-agh1.ifj.edu.pl/public_users/b14olsze/Lustre.pdf

[11] Asit Dan and Don Towsley. 1990. An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes. *SIGMETRICS Perform. Eval. Rev.* 18, 1 (apr 1990), 143–152. https://doi.org/10.1145/98460.98525

[12] Gil Einziger and Roy Friedman. 2014. TinyLFU: A Highly Efficient Cache Admission Policy. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 146–153. https://doi.org/10.1109/PDP.2014.34

[13] Ohad Eytan, Danny Harnik, Effi Ofer, Roy Friedman, and Ronen Kat. 2020. It's Time to Revisit LRU vs. FIFO. In *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*. USENIX Association. https://www.usenix.org/conference/hotstorage20/presentation/eytan

[14] Ting Gong, Tyler Lee, Cory Stephenson, Venkata Renduchintala, Suchismita Padhy, Anthony Ndirango, Gokce Keskin, and Oguz Elibol. 2019. A Comparison of Loss Weighting Strategies for Multi task Learning in Deep Neural Networks. *IEEE Access* 7 (2019), 141627–141632. https://doi.org/10.1109/ACCESS.2019.2943604

[15] Sebastien Gougeaud, Soraya Zertal, Jacques-Charles Lafoucriere, and Philippe Deniel. 2015. OGSSim: Open Generic data Storage systems Simulation tool. In *Eighth EAI International Conference on Simulation Tools and Techniques*. ACM, Athens, Greece. https://doi.org/10.4108/eai.24-8-2015.2261023

[16] Takahiro Hirofuchi and Ryousei Takano. 2020. A Prompt Report on the Performance of Intel Optane DC Persistent Memory Module. *IEICE Transactions on Information and Systems* E103.D, 5 (May 2020), 1168–1172. https://doi.org/10.1587/transinf.2019EDL8141 arXiv: 2002.06018.

[17] Bruce Jacob, Spencer Ng, and David Wang. 2007. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[18] Song Jiang and Xiaodong Zhang. 2002. LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (Marina Del Rey, California) *(SIGMETRICS '02)*. Association for Computing Machinery, New York, NY, USA, 31–42. https://doi.org/10.1145/511334.511340

[19] Youngjae Kim, Brendan Tauras, Aayush Gupta, and Bhuvan Urgaonkar. 2009. FlashSim: A Simulator for NAND Flash-Based Solid-State Drives. In *2009 First International Conference on Advances in System Simulation*. 125–131. https://doi.org/10.1109/SIMUL.2009.17

[20] Kathy Kincade. 2019. UniviStor: Next-Generation Data Storage for Heterogeneous HPC. Retrieved 2021-02-23 from https://cs.lbl.gov/news-media/news/2019/univistor-a-next-generation-data-storage-tool-for-heterogeneous-hpc-storage/

[21] S Klasky, Hasan Abbasi, M Ainsworth, Jong Youl Choi, Matthew Curry, T Kurc, Q Liu, Jay Lofstead, Carlos Maltzahn, Manish Parashar, Norbert Podhorszki, Eric Suchyta, F Wang, M Wolf, C.S. Chang, R. Churchill, and Stéphane Ethier. 2016. Exascale Storage Systems the SIRIUS Way. *Journal of Physics: Conference Series* 759 (Oct. 2016), 012095. https://doi.org/10.1088/1742-6596/759/1/012095

[22] Thomas Leibovici. 2015. Taking back control of HPC file systems with Robinhood Policy Engine. *International Workshop on the Lustre Ecosystem: Challenges and Opportunities* (2015). arXiv:1505.01448 http://arxiv.org/abs/1505.01448

[23] Yonggang Liu, Renato Figueiredo, Yiqi Xu, and Ming Zhao. 2013. On the design and implementation of a simulator for parallel file system research. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, Long Beach, CA, USA, 1–5. https://doi.org/10.1109/MSST.2013.6558438

[24] Glenn K. Lockwood, Wucherl Yoo, Suren Byna, Nicholas J. Wright, Shane Snyder, Kevin Harms, Zachary Nault, and Philip Carns. 2017. UMAMI: A recipe for generating meaningful metrics through holistic I/O performance analysis. In *Proceedings of PDSW-DISCS 2017 - 2nd Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems - Held in conjunction with SC 2017: The International Conference for High Performance Computing, Networking, Storage a*. 55–60. https://doi.org/10.1145/3149393.3149395

[25] Jakob Lüttgau, Michael Kuhn, Kira Duwe, Yevhen Alforov, Eugen Betke, Julian Kunkel, and Thomas Ludwig. 2018. Survey of Storage Systems for High-Performance Computing. *Supercomputing Frontiers and Innovations* 5, 1 (April 2018), 31–58–58. https://doi.org/10.14529/jsfi180103 Number: 1.

[26] Nimrod Megiddo and Dharmendra S. Modha. 2003. ARC: A Self-Tuning, Low Overhead Replacement Cache *(FAST '03)*. USENIX Association, USA, 115–130.

[27] Florent Monjalet and Thomas Leibovici. 2019. Predicting File Lifetimes with Machine Learning. In *High Performance Computing*, Vol. 11887 LNCS. Springer, 288–299. https://doi.org/10.1007/978-3-030-34356-9_23

[28] Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. *2013 5th IEEE International Memory Workshop, IMW 2013*, 21–25. https://doi.org/10.1109/IMW.2013.6582088

[29] Mohammed Islam Naas, François Trahay, Alexis Colin, Pierre Olivier, Stéphane Rubini, Frank Singhoff, and Jalil Boukhobza. 2021. EZIO-Tracer: Unifying Kernel and User Space I/O Tracing for Data-Intensive Applications. In *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems* (Online Event, United Kingdom) *(CHEOPS '21)*. Association for Computing Machinery, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/3439839.3458731

[30] John K Ousterhout. 1990. Why Aren't Operating Systems Getting Faster As Fast as Hardware? *1990 Summer USENIX Annual Technical Conference* (1990), 247–256.

[31] Sejin Park and Chanik Park. 2017. FRD: A filtering based buffer cache algorithm that considers both frequency and reuse distance. In *Proc. of the 33rd IEEE International Conference on Massive Storage Systems and Technology (MSST)*.

[32] César San-Lucas and Cristina L. Abad. 2016. Towards a fast multi-tier storage system simulator. In *2016 IEEE Ecuador Technical Chapters Meeting (ETCM)*. 1–5. https://doi.org/10.1109/ETCM.2016.7750836

[33] Woong Shin, Christopher Brumgard, Bing Xie, Sudharshan Vazhkudai, Devarshi Ghoshal, Sarp Oral, and Lavanya Ramakrishnan. 2019. Data Jockey: Automatic Data Management for HPC Multi-tiered Storage Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 511–522. https://doi.org/10.1109/IPDPS.2019.00061 ISSN: 1530-2075.

[34] Luis Thomas, Sebastien Gougeaud, Stéphane Rubini, Philippe Deniel, and Jalil Boukhobza. 2021. Predicting File Lifetimes for Data Placement in Multi-Tiered Storage Systems for HPC. In *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems* (Online Event, United Kingdom) *(CHEOPS '21)*. Association for Computing Machinery, New York, NY, USA, Article 2, 9 pages. https://doi.org/10.1145/3439839.3458733

[35] Cristian Ungureanu, Biplob Debnath, Stephen Rago, and Akshat Aranya. 2013. TBF: A memory-efficient replacement policy for flash-based caches. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. 1117–1128. https://doi.org/10.1109/ICDE.2013.6544902

[36] J. van den Berg and A. Gandolfi. 1992. LRU is better than FIFO under the independent reference model. *Journal of Applied Probability* 29, 1 (1992), 239–243. https://doi.org/10.2307/3214811

[37] Bharti Wadhwa, Surendra Byna, and Ali Butt. 2018. Toward Transparent Data Management in Multi-Layer Storage Hierarchy of HPC Systems. In *2018 IEEE International Conference on Cloud Engineering*

(IC2E). 211–217. https://doi.org/10.1109/IC2E.2018.00046

[38] Lipeng Wan, Zheng Lu, Qing Cao, Feiyi Wang, Sarp Oral, and Bradley Settlemyer. 2014. SSD-optimized workload placement with adaptive learning and classification in HPC environments. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–6. https://doi.org/10.1109/MSST.2014.6855552

[39] Wenguang Wang. 2004. *Storage Management for Large Scale Systems*. Ph.D. Dissertation. CAN. https://doi.org/10.5555/1123838 AAINR06171.

[40] Orcun Yildiz, Amelie Zhou, and Shadi Ibrahim. 2017. Eley: On the Effectiveness of Burst Buffers for Big Data Processing in HPC Systems. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 87–91. https://doi.org/10.1109/CLUSTER.2017.73