HFetch: Hierarchical Data Prefetching in Multi-Tiered Storage Environments

Hariharan Devarajan
Illinois Institute of Technology, Chicago
IL
hdevarajan@hawk.iit.edu

Anthony Kougkas
Illinois Institute of Technology, Chicago
IL
akougkas@hawk.iit.ed

Xian-He Sun
Illinois Institute of Technology, Chicago
IL
sun@iit.edu

KEYWORDS

Data Prefetching, Heterogeneous Prefetching, Layered prefetching, Deep memory hierarchy, Burst buffers

ACM Reference Format:
Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. 2019. HFetch: Hierarchical Data Prefetching in Multi-Tiered Storage Environments. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 EXTENDED ABSTRACT

Data-intensive computing offers unprecedented opportunities for scientific discovery, high-fidelity insights, and data-driven decision making with timely data access being a driving factor of the overall execution time [1]. The computational efficiency of modern applications is closely related to the ability of the storage systems to push data to the compute units as the performance of the latter has progressed significantly faster than disk capabilities [8]. Modern applications spend significant amounts of time in reading data; in some cases up to 80% of the overall execution time [5]. As we move towards exascale, this trend is expected to exacerbate further the I/O bottleneck (e.g., I/O wall problem [7]). While modern storage systems are adapting quickly to the challenges of today's fast-paced computation environment, they still struggle to address the demand for low data access latency.

To address the gap between the data consumption from the compute and data supply from the storage system, recent research has proposed solutions broadly categorized in two disconnected directions: innovative software (that sits between applications and the storage) which is responsible to mask the access latency and new hardware devices (that offer lower access latency and higher throughput). There are several software techniques to optimize read-intensive workloads such as data concurrency, data locality, I/O reordering, and data prefetching. On the hardware end, many of the leadership computing facilities have already deployed fast node-local NVMe SSDs and/or shared specialized buffering nodes (i.e., burst buffers) creating a new multi-tiered storage environment, called deep memory and storage hierarchy. Many supercomputing facilities have widely deployed specialized buffering solutions such as Cray's Datawarp [2] and DDN's IME [3]. However, traditional file systems are not equipped to handle this new hierarchy and users are left to manually manage the layers of the hierarchy. To handle data movement through the hierarchy, some software platforms such as Data Elevator [4], Univistor [9], and Hermes [6] have been developed. All the above systems are designed to only optimize write-heavy workloads via data buffering. Therefore, the read operation optimizations, that leverage the new multi-tiered storage environment, have to be further explored.

In this study, we focus on data prefetching optimization in multi-tiere storage environments. We observe several significant challenges came to light, when optimizing read operations by using the existing methods. *Firstly*, a truly hierarchical data prefetching in a multi-tiered storage environment is not supported by any of the existing solutions. All prefetching solutions have to answer two main questions: a) when to prefetch data, and b) what data to prefetch. Additionally, the presence of multiple tiers of the storage hierarchy gives rise to a third question: *where to prefetch data*? Modern architectures, in extreme scale computing, suggest a decrease in the amount of RAM per core. This requires the existing memory-based data staging and prefetching solutions to evolve and include the multi-tiered storage. *Secondly*, existing prefetching solutions rely on identifying application's data access patterns which poses several issues such as erroneous access pattern detection, unwanted data evictions, cache pollution, cache redundancy and resource interference. To alleviate these issues, an application-agnostic approach should be employed. *Lastly*, prefetching data in a smaller granularity than the entire file is unavailable or, at best, limited. Finer granularity can lead to better prefetching resource utilization and, therefore, higher performance. As the complexity of computation workflows increases, solutions relying on identifying data access patterns become in-feasible.

To address the above challenges, we present HFetch, a new hierarchical data prefetcher that supports multi-tiered storage environments. HFetch is primarily a data-centric prefetching decision engine that utilizes system-generated events, while leveraging the presence of multiple tiers of storage, to perform hierarchical data placement at the required time. HFetch can obtain a global system-wide view of how data is accessed, regardless of which application or process is performing the access, by monitoring the file system and collecting statistics for each data segment. Based on a global scoring function that ranks the importance or urgency of the targeted data, it makes intelligent decisions as to when, what, and where to prefetch data. In modern scientific workflows, data might be read multiple times across applications which might create severe issues for prefetching cache management. Cache pollution, cache redundancy, and unnecessary data evictions leading to increased miss ratios are the norm, and not the exception, especially in extremely large scale workloads. HFetch addresses these issues by maintaining global file heatmaps that represent how a file is accessed across processes or applications. It uses those heatmaps to express the placement of data in a hierarchical system.

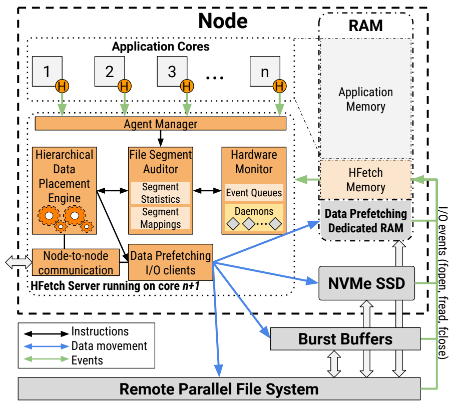

Figure 1: HFetch overview.

1.1 HFetch Overview

The main idea behind HFetch is to fetch portions of a file to a tier of the hierarchy based on access frequency, recency, and relationship between segments (i.e., file segment sequencing). In other words, instead of guessing what an application will access next, HFetch collects access statistics of file regions (which we call file segments) from the file systems themselves and pro-actively loads them in the hierarchy, based on a segment score, that reflects the urgency to access the chosen segment. This score incorporates the frequency, recency and sequencing of the segments thereby creating a file access heatmap. The file heatmap is then used to naturally match it to a hierarchical environment. In effect, HFetch answers the three prefetching questions (what to prefetch, when to prefetch, and where to place prefetched data) by naturally mapping the spectrum of segment scores to the appropriate tier.

Figure 1 shows the architecture of HFetch. HFetch follows a client-server model. Each compute node is equipped with an HFetch server running on one of the cores. Each application dynamically links to the HFetch library and a background HFetch thread, we call *Agent*, is spawned alongside each application process (depicted as *H*). Upon application initialization, a small fraction of the main memory is allocated for HFetch internal structures. Our proposed architecture incorporates system designs with a local NVMe drive, a shared collection of burst buffer nodes, and a remote PFS. Essentially, the flow of operations in HFetch is as follows. Each tier independently pushes its I/O events into a queue that resides in HFetch memory. A hardware monitor collects events and passes them to the file segment auditor who calculates statistics for each file segment. An engine periodically devises a data placement plan in the hierarchy based on each segment's attributes and pushes it to the I/O clients to be executed while updating the segment mapping.

1.2 Initial Results

To test our system under real workloads, we compare it with state-of-the-art prefetching solutions.

Montage is a collection of programs comprising an astronomical image mosaic engine. It is a use-case of a workflow where multiple kernels share data for different purposes and access this common data concurrently. Each phase of building the mosaic takes an input from the previous phase and outputs intermediate data to the next

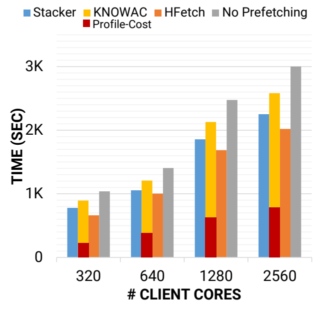

Figure 2: Montage (weak scaling).

one. Figure 2 shows the results for Montage. During this test, each process does 10 MB of I/O operations in 16 time steps for a total of 400 GB for the largest scale. We weak scaled the execution of Montage by increasing the number of processes from 320 to 2560. Required data are initially staged in the burst buffer nodes. The system is overall configured with prefetching cache organized in 1.5 GB RAM space, 2 GB in local NVMe drives and 400 GB burst buffers. As can be seen, the best read performance is achieved by KnowAc, a history-based prefetcher, since the prefetcher knows exactly what to load next. However, such approach suffers from prolonged profiling costs. Stacker avoids pre-processing steps and build its models as it goes, but demonstrated a lower hit ratio due to some cache conflicts and unwanted data evictions. HFetch was able to utilize all available tiers and performed the best, offering from 5% to 25% better end-to-end performance when compared to Stacker and 10% to 30% better than KnowAc.

We have introduced, HFetch, a truly hierarchical data prefetcher that implements a data-centric design. Results show a promising solution to a growing problem of extreme scale data access. We showcase the benefits of such approach by evaluating its scalability, effectiveness, and overall performance where it achieves 10-35% higher read throughput than prefetching solutions tested.

REFERENCES

[1] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)* 7, 3 (2011), 8.
[2] Cray. 2016. Datawarp documentation. https://pubs.cray.com/browse/datawarp/software
[3] DDN. 2018. IME burst buffers documentation. https://www.ddn.com/products/ime-flash-native-data-cache/
[4] Bin Dong, Suren Byna, Kesheng Wu, Hans Johansen, Jeffrey N Johnson, Noel Keen, et al. 2016. Data elevator: Low-contention data movement in hierarchical storage system. In *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*. IEEE, Hyderabad, India, 152–161.
[5] Bin Dong, Teng Wang, Houjun Tang, Quincey Koziol, Kesheng Wu, and Suren Byna. 2018. ARCHIE: Data Analysis Acceleration with Array Caching in Hierarchical Storage. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE.
[6] Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. 2018. 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*. ACM, USA, 219–230.
[7] John Shalf, Sudip Dosanjh, and John Morrison. 2010. Exascale computing technology challenges. In *International Conference on High Performance Computing for Computational Science*. Springer, USA, 1–25.
[8] X-H Sun, Yong Chen, and Ming Wu. 2005. Scalability of heterogeneous computing. In *2005 International Conference on Parallel Processing (ICPP'05)*. IEEE, USA, 557–564.
[9] Teng Wang, Suren Byna, Bin Dong, and Houjun Tang. 2018. UniviStor: Integrated Hierarchical and Distributed Storage for HPC. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, USA, 134–144.