The Case for Optimizing the Frequency of Periodic Data Movements over Hybrid Memory Systems

Thaleia Dimitra Doudali
Georgia Institute of Technology
thdoudali@gatech.edu

Daniel Zahka
Georgia Institute of Technology
dzahka3@gatech.edu

Ada Gavrilovska
Georgia Institute of Technology
ada@cc.gatech.edu

ABSTRACT

Application performance improvements in emerging systems with hybrid memory components, such as DRAM and Intel's Optane DC persistent memory, are possible via periodic data movements, that maximize the DRAM use and system resource efficiency. Similarly, predominantly used NUMA DRAM-only systems benefit from data balancing solutions, such as AutoNUMA, which periodically remap an application and its data on the same NUMA node. Although there has been a significant body of research focused on the clever selection of the data to be moved periodically, there is little insight as to how to select the frequency of the data movements, i.e., the duration of the monitoring period. Our experimental analysis shows that fine-tuning the period frequency can boost application performance on average by 70% for systems with locally attached memory units and 5x when accessing remote memory via interconnection networks. Thus, there is potential for significant performance improvements just by cleverly selecting the frequency of the data movements apart from choosing the data itself. While existing solutions empirically set the duration of the period, our work provides insights into the application-level properties that influence the choice of the period. More specifically, we show that there is a correlation between the application-level data reuse distance and migration frequency. Future work aims to solidify this correlation and build a profiling solution that provides users with the data movement frequency which dynamic data management solutions can then use to enhance performance.

CCS CONCEPTS

• **Hardware → Memory and dense storage**; **Analysis and design of emerging devices and systems**; • **Computer systems organization → Heterogeneous (hybrid) systems**.

KEYWORDS

Data Tiering, Periodic Data Movements, Page Migration Frequency, Page Scheduler, Emerging Memory Technologies, Heterogeneous Memory Systems, Hybrid Memory Systems, Non Volatile Memory

MEMSYS 2020, September 28-October 1, 2020, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8899-3/20/09…$15.00
https://doi.org/10.1145/3422575.3422788

ACM Reference Format:
Thaleia Dimitra Doudali, Daniel Zahka, and Ada Gavrilovska. 2020. The Case for Optimizing the Frequency of Periodic Data Movements over Hybrid Memory Systems. In *The International Symposium on Memory Systems (MEMSYS 2020), September 28-October 1, 2020, Washington, DC, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3422575.3422788

1 INTRODUCTION

Hybrid Memory Systems. Current data-intensive analytics in datacenter and exascale compute environments require hundreds of gigabytes and terabytes of main memory capacity to meet their performance requirements. To this extent, new memory technologies have been widely explored, to provide higher capacity density at a reasonable cost, relative to DRAM. The recent release of the Intel® Optane™ DC Persistent Memory Module (PMEM) [3] proposes its use together with a smaller capacity of DRAM, so as to offset the higher latency and lower bandwidth of PMEM [13, 22]. The *App Direct* operational mode of PMEM allows it to be used alongside DRAM as main memory managed by the operating system, similar to traditional NUMA systems. Another cost-effective solution to extend the overall memory capacity is the disaggregation of the available memory resources [19, 20]. In this case, there is a remote pool of memory that can be shared among the compute servers, alongside locally-attached memory units. Future interconnection fabrics [1, 2] promise to deliver high-speed data transfers across the servers.

The disparity in the access speeds of the available memory components, given their underlying technology and relative distance to the compute units, is exacerbated in these future systems compared to traditional DRAM-only NUMA systems. Thus, appropriate dynamic data management solutions are necessary to boost application performance through maximizing the use of local DRAM but also making efficient use of the available resources, such as bandwidth.

Data Management Solutions. There has been a notable amount of existing solutions, which we present in more detail in Section 5, that provides support for dynamic data movement intending to reduce the application performance loss compared to traditional DRAM-only systems. The predominant system-level and application-agnostic approach is to monitor the data access information and to **periodically** aggregate it into a decision on which data to move, so that the resulting data placement improves application- or system-level performance metrics. In contrast to periodic data management, runtime solutions such as Unimem [26] or Tahoe [27] manage the frequency of data movements by relying on the MPI communication phases and task based execution respectively. Either way, the selection of which and how many data to move usually involves some threshold, that aims to capture the performance benefit of moving the data to faster memory components. The most commonly used information in selecting which data to move is data access frequency during a period [5, 8, 21]. Hardware-based [18] and profiling [9–11, 23, 25] solutions enable the collection of additional information on data access behavior and allow for more sophisticated data movement selection.

However, achieving optimal data placement through clever data movement selection comes at a cost, due to the management **overheads**, which are not trivial in system-level solutions. First, the lack of hardware support on current x86-based platforms for page-level access counting results in solutions that involve **periodic** page table scans and TLB flushes. These software overheads, which can be reduced by specialized hardware and page sampling approaches [5, 12, 14, 15, 21, 24], can be significant for applications with large memory footprints that hybrid memory systems want to accommodate. Second, there are overheads associated with accesses to pages that are under migration.

Problem Statement. Although periodic data movement improves application performance in hybrid memory systems, the overheads associated with monitoring and deciding on which data to move and at what times, can have nontrivial impact to performance. However, throughout system-level existing solutions, these constant periods are **empirically** set, without a clear methodology on how to decide upon the optimal periodicity. The goal of this work is to investigate the sensitivity of application performance on the choice of the memory management period, as a step toward closing this gap.

Concretely, the **paper contributions** are:

- We demonstrate the significance of the frequency of memory management operations on application performance and discuss the factors that determine the impact of changing this management parameter (Section 3).
- Using experimention on real hardware with heterogeneous memories – DRAM and Intel Optane PMEM – and simulation-based analysis, we present evidence of the relationship of one application-level property – the reuse distance of its memory accesses – and the configuration of the memory management period (Section 4).
- Given the impending diversity in memory system configurations, and the continued growth in memory-intensive applications, this paper is a call to action for new tools that will automate the process of configuring and tuning the system-level memory management components for the target hardware and workloads.

2 METHODOLOGY

The analysis presented in the paper are performed on a hybrid memory hardware testbed with DRAM and Intel's Optane DC persistent memory, and in a simulation environment. Here we describe the platforms, methodology, and applications used in the experiments.

Applications. We select high-performance computing kernels from the Rodinia [6] benchmark suite, that show a variety of random and predictive access patterns. We also include the PageRank application from the Lonestar [16] suite, which is known to exhibit irregular behavior.

2.1 Optane DC PMEM Platform

Testbed. We have access to a server with Intel Optane DC Persistent Memory Modules (PMEM). The machine contains 375 GB of DRAM and 6 TB of PMEM. We configure the Optane Memory Modules in *App Direct* mode, which allows the operating system to manage the DRAM and PMEM on each socket as separate NUMA nodes.

Methodology. We implement a page migration module in Linux kernel version 5.4 that attaches to a target process and periodically selects several frequently accessed pages, determined by a certain threshold, and moves them appropriately between DRAM and PMEM. Every period, we identify page accesses using the available OS-level information, as also done in [12, 14]. The module determines which pages were accessed by scanning the target's page table entries and recording whether or not each accessed bit was set during that period. To estimate the page hotness, we calculate the exponential moving average (with a certain smoothing factor) of the page's accessed bit history and compare it with a hotness threshold that classifies a page as hot or cold, as also done in [17]. Then, utilizing the `move_pages()` function from the kernel's NUMA-based migration API, we asynchronously move hot pages to DRAM and cold pages to PMEM. The user specifies a target DRAM capacity percentage and the module dynamically adjusts the cutoff so that the correct percentage of the target processes pages are mapped to DRAM and PMEM respectively.

2.2 Simulation

Memory Access Trace Collection. We use Intel's Pin [4] dynamic binary instrumentation tool in order to capture the memory address of the last level cache misses out of a simulated three level data cache hierarchy. We configure the cache sizes to be proportional to the native Optane DC PMEM platform. We collect traces of the Rodinia benchmarks with appropriate input data sizes, that allow for reasonable access trace collection and analysis times, but exhibit similar last level cache miss rates to the ones when executing over the native PMEM platform. Figure 1 shows a graphic representation of the application data access behavior. While `backprop`, `kmeans` and `hotspot` have sequential access patterns in distinct strides, `lud` exhibits an irregular shrinking memory footprint that includes sequential traversals, and `bfs` validates the irregular breadth-first graph traversal behavior.

Hybrid Memory System Simulation. We develop a Python-based simulation environment of a hybrid memory system that contains two components with independently configurable technology parameters (e.g., access latency and bandwidth) and relative distance, managed by the operating system. This light-weight simulation environment permits quick exploration of application performance across a variety of hybrid memory system configurations. The simulated system contains a *fast* (i.e., DRAM) and a *slow* memory component, such as PMEM or disaggregated DRAM. For the PMEM simulation, we follow the access characteristics described in [13], which also correspond to our hardware testbed, and set a 1:3 latency and 1:0.37 bandwidth ratio between the fast and slow memory. For the disaggregated DRAM simulation, we use information about projected speeds of future high speed interconnects [1]. For this platform, we set a 1:22.2 latency and 1:0.1 bandwidth ratio between

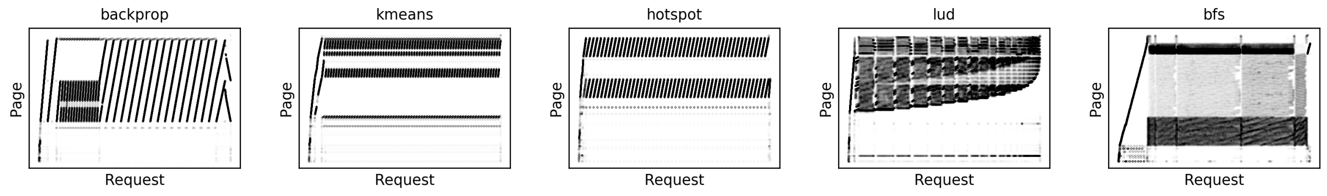


Figure 1: Memory access traces of the selected applications including sequential and irregular data access behavior.

the fast and slow memory. We assume that the overall capacity of the memory system is equal to the application's memory footprint, split at a 1:8 capacity ratio between fast and slow memory, which is representative of commercially available platforms, such as with Optane DC PMEM.

Performance model. We estimate performance given the distribution of accesses between fast and slow memory, following some properties of the analytical model used in [21]. First, we aggregate the access latency of the memory requests given their dynamic placement throughout execution. Second, we track possible delays of issuing memory requests, because of limited bandwidth availability. Third, we include a constant page migration overhead and a constant delay at the beginning of each period, that captures the software overheads of scanning the page tables and selecting which pages to migrate. All constant overheads used in the model follow the proposed values in [15, 21].

Methodology. We implement a page scheduling component that periodically selects and migrates pages between the two simulated memory components. Pages are initially allocated across the available memory components in an interleaved order. Since we do not use a cycle-accurate simulation infrastructure, we assume that a period is the time duration when a fixed number of memory accesses are issued, e.g., a thousand accesses. Upon the start of a period, the page scheduler aggregates the per page access counts (hotness) during the previous period, thus using access *history* to make predictions for the page hotness during the current period, as done in [5, 12, 14, 15, 21, 24]. The pages that were accessed during the previous period are sorted in descending hotness order. Since the overall memory capacity is equal to the application's number of pages, the scheduler needs to swap hot pages that are currently in slow memory with least recently used (LRU) pages residing in fast memory. We use a static upper threshold to determine how many pages to move between the two memories, which is the capacity of the destination memory, as done in [8]. Page swaps happen asynchronously, assuming DMA support, and sequentially in order of (hot, LRU) page pairs.

3 SENSITIVITY ANALYSIS

3.1 Optane DC PMEM Platform

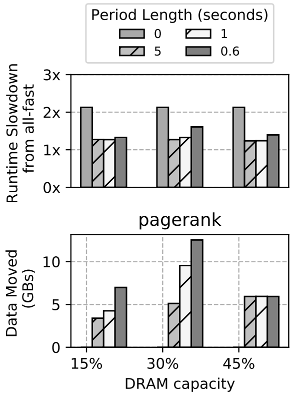
Using the Optane-based hardware testbed and the data management methodology described in Section 2.1, we analyze the performance of two applications with irregular and sequential access patterns, `pagerank` and `lud`. We explore the effects of period length on the application runtime and the number of migrations performed by running experiments with monitoring periods of 600ms, 1s, and 5s. We emulate different hardware configurations by restricting the DRAM capacity used by the application, from 10% of the pages mapped by the application up to 50%, at which point, for both applications, the entire working set fits in DRAM.

Irregular Access Behavior. The results from running `pagerank` with a memory footprint of 12.8 GB are shown in Figure 2a. At 15% DRAM capacity, `pagerank` benefits greatly from moving a small subset of pages into DRAM early in the application execution and then by doing a small number of migrations for the remaining time. By analyzing access bit history from the application execution, we found that 13.5% of the pages are accessed at least $\frac{3}{4}$ of the scanning periods. Moving this set of pages into DRAM closes much of the gap between DRAM-only and PMEM-only execution times. After this set of pages is moved into DRAM, trying to find the next hottest 2.5% of pages results in a large amount of less useful migrations, especially at shorter period lengths, where there is less stability in the hotness ordering of these lesser used pages.

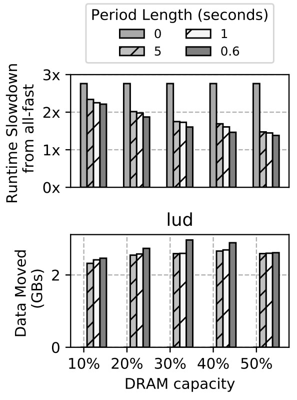
At 30% DRAM capacity, the scheduler keeps DRAM full by lowering its cutoff to include pages that are more sporadically used. This next tier of pages consists of 22% of pages that are accessed between $\frac{1}{4}$ and $\frac{3}{4}$ of the periods. Because these pages are less frequently used and only a portion of them can fit into DRAM, just a single access can push a page sitting near the hotness threshold above or below the DRAM cutoff point. Having a shorter period causes the scheduler to be over-responsive to these frequent page reclassifications, causing an extreme number of page migrations that create only a small performance benefit.

At 45% DRAM capacity, there is enough space to fit the pages that fluctuated between hot and cold status in the case with 30% DRAM capacity into DRAM. Once these pages are moved into DRAM, there are almost no migrations left to perform. This is why the same amount of data is moved for all three period lengths.

Sequential Access Behavior. Figure 2b shows results from a similar set of experiments for `lud` with a memory footprint of 1.9 GB. The application characteristics of `lud` are distinct from `pagerank` in that accesses are highly sequential and the size of the working set is dynamic and shrinks throughout the application's execution. Most of the pages in `lud` exhibit the same access pattern, where they are accessed continually until some point where they become idle for the rest of the program execution. In this situation, having a shorter period can be beneficial as it allows the scheduler to be more responsive to changes in the working set. As soon as pages become cold, it is good to flush them out of DRAM and replace them with a page that is still consistently being accessed. This is reflected across all of the capacity configurations in Figure 2b, where the



(a) pagerank



(b) lud

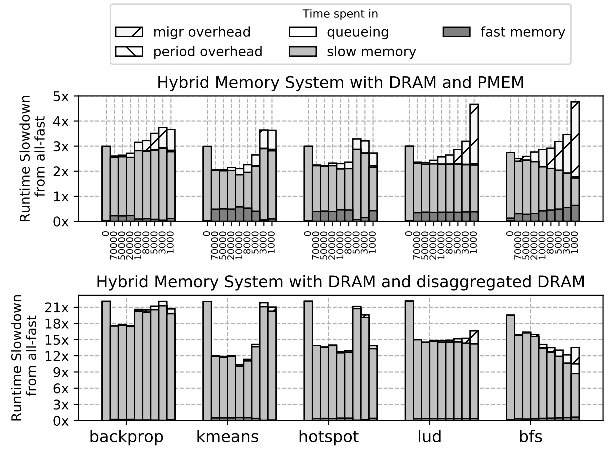
Figure 2: Performance and bytes moved for pagerank and lud when executing over the Optane PMEM platform across variable DRAM capacity and shorter period lengths. Zero length corresponds to static allocation of all data in PMEM.

shorter periods always perform best. A similar amount of overall data is moved for each period length, but with a shorter period, cold data is moved out to PMEM and hot data into DRAM in a more timely fashion.

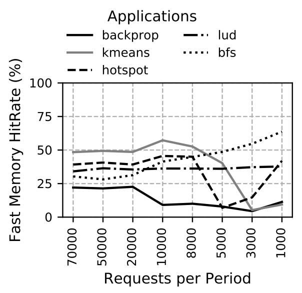
Summary. Looking at these two applications, we can see how the period length and the DRAM capacity interact with the characteristics of the target application to influence which pages are migrated and when. The heuristic that powers our page scheduler serves as an indicator for the reuse distance of a page. Having a shorter period means that the scheduler can respond faster and more often to changes in *page reuse distance*. This is beneficial in situations where there are changes in measured page reuse distance that are strongly correlated with actual changes in future page reuse distance. This is the case with `lud`, where an increase in the measured reuse distance of a given page is a good indicator that the page will become idle. In situations where the measured reuse distances are stable with respect to the DRAM capacity cutoff, as in `pagerank` at DRAM capacities of 15% and 45%, extra calls to the scheduler are not necessary and result in similar DRAM usage but with unnecessary CPU usage and data movement. In cases where the measured reuse distances are unstable with respect to the cutoff, as in `pagerank` at DRAM capacity 30%, the shorter period is more responsive to noise in measured reuse distance as opposed to true changes in page reuse distance. In this case, having more time in between scheduler invocations serves to smooth out this noise and achieve good use of DRAM, but at lower overhead. By examining these two applications on our Optane platform, we have seen evidence that the optimal scheduling period for an application is dependent on the distribution of page reuse distances of the application as well as how dynamic that distribution is over the execution of the application.

3.2 Simulation

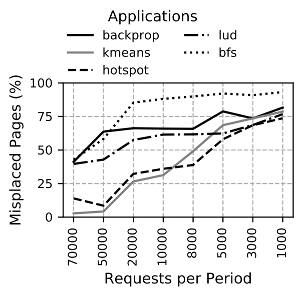
Our simulation setup, described in Section 2.2, allows us to break down the time an application spends in accessing memory and to capture the overheads associated with the periodic data monitoring and movements, as shown in Figure 3a. The different bars correspond to shorter lengths of the data monitoring and movement period. Zero requests per period represent the case of static data placement, interleaved across a 1:8 fast:slow memory capacity ratio. Here follows an analysis on how the different dominant break down parameters get affected over shorter period lengths.



(a) Runtime slowdown from the case where all data fits in fast memory (DRAM).



(b) Fast Memory Hit Rate.



(c) Misplaced Pages.

Figure 3: Application performance across shorter period lengths and simulated PMEM and disaggregated hybrid memory systems.

Fast Memory Hit Rate. First, we observe that shorter periods generally allow for an increase in the fast memory accesses, indicating that the scheduler is more responsive and performs more timely data movements. Figure 3b captures the fast memory hit rate validating this observation. The steady increase in the hitrate for `bfs` is due to the fact that there is a page subset that is being predominantly accessed throughout its execution and shorter periods allow the scheduler to better identify it, especially when it does not fit into the available DRAM capacity. On the other hand, the shrinking memory footprint of `lud` doesn't allow the history-based scheduler to make efficient data movements, resulting in minor improvements in hitrate. Most interestingly, the hitrate of `backprop`, `kmeans` and `hotspot`, who have distinct sequential access patterns in strides, significantly drops at specific period lengths and increases again only at very short ones. Section 4 sheds light into the application-level data access characteristics that result in this behavior, which is the data reuse distance.

Migration Overhead. Applications with sequential access patterns over a large portion of their working set don't always benefit from very short periods. At such intervals, the pages become less frequently accessed and the subset of pages accessed across periods even more distinct. The history-based page scheduling component tries to adapt to these frequent changes in the subset of pages being accessed across periods and generates a big number of unnecessary data movements. Figure 3c captures the percentage of pages that were misplaced by the history-based page scheduler at least during one period. A page is misplaced if at the beginning of a period the scheduler fails to accurately predict its hotness and appropriately place it across the hybrid memory. The page misplacements are used to capture the effectiveness of the history-based page migration selection [8]. We observe that the scheduler struggles to adapt and predict quick changes in the data access behavior over shorter periods. This results in potential unecessary data movements, whose overheads dominate over shorter periods, as shown in Figure 3a. This is prominent especially across `bfs` and `lud` whose irregular access pattern and shrinking memory footprint trigger unnecessary data movements given the limited history-based data access information.

Execution Platform. Figure 3a also includes the application runtime slowdown, given disaggregated DRAM as the slow memory component. In such systems, the slow memory access latency dominates the application runtime, because it is projected to be an order of magnitude higher than fast memory. However, runtime can still be significantly reduced, just by shortening the period interval of data movements and increasing the number of fast memory requests. Given the bigger disparity in the fast and slow memory access speeds of this system configuration, we don't necessarily observe the same sweet spot of period lengths compared to the ones when PMEM is the slow memory. This is due to the fact that in disaggregated systems the overheads and slow memory access latency are in the same order of magnitude, compared to local systems with PMEM as slow memory. Thus, there are applications like `bfs` who still benefit from shorter periods in disaggregated systems, but not in local systems where the aggregate migration and period overheads dominate. Therefore, changes in platform configurations can completely affect the best period length for the same application.

Summary. The simulation analysis validates observations made over the real PMEM hardware testbed. These are that applications with irregular access behavior can benefit from shorter period lengths, as long as the available DRAM capacity is enough to fit the part of their constantly accessed working set, else the overheads of unnecessary data movements will become dominant. For applications with more predictive access patterns, they benefit from shorter periods that still capture the pattern, but not from very short periods that *break the page reuse distance*, as will be further explained in Section 4. Finally, execution over a different platform with a more distinct disparity in the memory access speeds may lead to a completely different choice of period length that benefits performance.

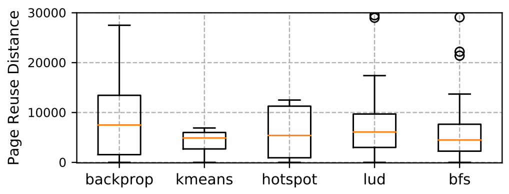


Figure 4: Page reuse distance distribution across evaluated applications.

3.3 Takeaways

This sensitivity analysis shows how tightly coupled is the interaction of the application data access behavior and the page scheduler's ability to quickly adapt to pattern changes, for a given period length, together with the configuration of the execution platform, such as the available DRAM capacity and memory access speed. Here we summarize the different parameters that contribute to the selection of period length that yields best application performance:

- **Execution Platform.** The configuration of the testbed with respect to the capacity of the available fast memory compared to the application's active working set, together with the difference in access latency and bandwidth of the available hybrid memory components.
- **Page Scheduling Policy.** The selection of data movements plays a crucial role in maximizing the use of the available fast memory. However, history-based information can not always capture future data access behavior, especially over shorter monitoring periods that break the repetitiveness of the access pattern, resulting in a potentially large number of page misplacements and unnecessary data movements, whose overheads may become dominant.
- **Data Access Behavior.** Applications with irregular versus more regular access patterns seem to benefit from distinctly different ranges of period length for a given execution platform configuration and page scheduling effectiveness.

4 IMPACT OF DATA ACCESS BEHAVIOR

Section 3 highlights the different parameters that get affected as we vary the length of the data monitoring and movement period of page scheduling solutions across hybrid memory systems. The complex relation between these parameters leads to the empirical tuning of the period across existing solutions. In this section, we further explore the data access characteristics that can be leveraged to hint towards a good selection of period length, *irrespective* of the execution platform and page scheduling policy.

A commonly used metric to indicate the behavior of the access pattern is data *reuse distance*. In the context of our simulation analysis, we define the reuse distance of a certain page, as the number of memory requests to other pages that passed between two consecutive accesses to that particular page. Figure 4 shows the distribution of the reuse distance across the pages of the applications we evaluated. We split our observations given the a-priori knowledge we have on the access patterns of the applications, as depicted in Figure 1.

Irregular Access Behavior. The outliers of reuse distance in `lud` and `bfs` hint towards their irregular access behavior. More specifically, the outliers of `lud` correspond to the part of its memory footprint that is shrinking, whose reuse distance is not repeating over time, compared to its page subset which gets accessed throughout execution in sequential traversals. In contrast, the majority of `bfs` pages are accessed throughout execution, however the graph traversal in a breadth-first style results in page reuse distances that are distinctly different across pages depending on the corresponding graph layers.

Sequential Access Behavior. On the other hand, applications with sequential access patterns have no outliers in page reuse distance, since there are specific page subsets that participate in the pattern and get reused at very specific distances (strides), that repeat throughout execution. There seems to be a correlation between the period length for which the application's fast memory hitrate dips (Figure 3b) and the median page reuse distance of the application. More specifically, when the history-based scheduler operates at a frequency which is less than the reuse distance of a significant group of pages, such as around 10,000 requests per period for `backprop`, 3,000 for `kmeans` and 5,000 for `hotspot`, then its ability to correctly predict the access behavior drops. Thus, it's critical that the history-based page scheduler operates at a frequency which doesn't break the page reuse distance, so that it can properly identify the repetition of any sequential strides over a certain page subset.

Takeaways. The page reuse distance helps us categorize an application into regular or irregular access pattern behavior. More importantly, it also seems to highly correlate with the choice of period length that optimizes performance for a given platform execution and page scheduling policy. More specifically, the operation of a history-based page scheduler at a frequency which **breaks the page reuse distance** of a significant page subset, can be detrimental to performance. Future work, aims to solidify these correlations and build a holistic modeling infrastructure, that binds all three parameters (execution platform, page scheduling policy and data access behavior) into the selection of an appropriate data monitoring and movement period length.

5 RELATED WORK

In this section we summarize *dynamic* data management solutions for hybrid memory systems focusing on the monitoring frequency of data access behavior and at what times data is moved, which varies primarily depending on the level of implementation.

System-level solutions keep track of application access behavior and *periodically* decide upon which pages to migrate. Such solutions include either actual OS-level prototypes or proposals evaluated in simulation environments. These solutions empirically set the period time interval (often referred to as *epoch*), while some share certain insights to justify their choice.

More specifically, when Thermostat [5] runs every 10 seconds or more, then the overheads of selecting page migrations results in negligible CPU activity and potential application slowdown. The authors of Kleio [8] fix the period to 0.01 seconds, so as to allow for enough period samples in their machine intelligent placement methodology. Next, the authors of HMA [21] show that shorter periods (e.g., from 1 second to 0.1 seconds) allow the placement policy to adapt more quickly to the data access pattern, but when it becomes too fine grained (e.g. 0.001 seconds) the per period software overheads dominate. Similar exploration is also done by the authors of [15], who propose page migration support for disaggregated systems with DRAM and PMEM memory components.

Our work not only validates all of the above observations, but takes it a step further and identifies the data access behavior that leads to these insights.

Runtime solutions rely on application-specific detection of distinguishable execution to trigger data movements. More specifically, Unimem [26] leverages the MPI communication phases to launch data movements and Tahoe [27] targets task-based execution to trigger the necessary data movements. These solutions are tailored to the specific runtimes and explore a range of data movement frequencies aligned with computational phases.

Our analysis does not depend on application phases and specific runtimes, but rather explores application agnostic solutions and operating system-level page scheduling.

Specialized hardware across related works helps reduce the overheads of acquiring the necessary information for data movement decisions, like page hotness tracking [5, 12, 14, 15, 21, 24]. Additionally, custom hardware automates threshold-based data movement triggers [7, 18]. Finally, additional hardware for buffering data copies and allowing access to data that is under migration allows for reducing access stalls and improving application performance [18].

Even when the per period overheads are reduced via hardware support, the question remains of which period duration better captures data access behavior.

6 SUMMARY AND FUTURE WORK

In this paper we show that the choice of the parameter determining the periodicity of memory management operations can have significant impact on the attainable application performance and on the efficiency of the memory management methods. Yet, existing solutions fall short on proposing a methodology for how this parameter should be configured. We identify that the execution platform and effectiveness of the page scheduling policy in combination with the application-specific data access behavior lead to a period length sweet spot which maximizes fast memory use with minimal overheads. Instead of empirically setting the period length, we envision a profiling tool that leverages characteristics of the data access pattern, using metrics such as the page reuse distance, in order to identify a range of period lengths that can improve fast memory locality, which should then be combined with knowledge of the execution platform and page scheduling policy.

ACKNOWLEDGMENTS

We thank the anonymous reviewers on their constructive feedback. This work is partially supported by NSF award SPX-1822972, and via funding and infrastructure donations from Facebook and Intel.

REFERENCES

[1] 2020. Gen-Z Consortium: Computer Industry Alliance Revolutionizing Data Access. https://genzconsortium.org/.

[2] 2020. Intel Omni-Path Architecture (Intel OPA) Driving Exascale Computing and HPC Driving Exascale Computing and HPC with Intel. https://www.intel.com/content/www/us/en/high-performance-computing-fabrics/omni-path-driving-exascale-computing.html.

[3] 2020. Intel® Optane^{TM} DC Persistent Memory. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html.

[4] 2020. Pin - A Dynamic Binary Instrumentation Tool. https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html.

[5] Neha Agarwal and Thomas F. Wenisch. 2017. Thermostat: Application-Transparent Page Management for Two-Tiered Main Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xiapos;an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 631–644. https://doi.org/10.1145/3037697.3037706

[6] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC) (IISWC '09)*. IEEE Computer Society, Washington, DC, USA, 44–54. https://doi.org/10.1109/IISWC.2009.5306797

[7] Chiachen Chou, Aamer Jaleel, and Moinuddin Qureshi. 2017. BATMAN: Techniques for Maximizing System Bandwidth of Memory Systems with Stacked-DRAM. In *Proceedings of the International Symposium on Memory Systems* (Alexandria, Virginia) *(MEMSYS '17)*. Association for Computing Machinery, New York, NY, USA, 268–280. https://doi.org/10.1145/3132402.3132404

[8] Thaleia Dimitra Doudali, Sergey Blagodurov, Abhinav Vishnu, Sudhanva Gurumurthi, and Ada Gavrilovska. 2019. Kleio: A Hybrid Memory Page Scheduler with Machine Intelligence. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (Phoenix, AZ, USA) *(HPDC '19)*. ACM, New York, NY, USA, 37–48. https://doi.org/10.1145/3307681.3325398

[9] Thaleia Dimitra Doudali and Ada Gavrilovska. 2017. CoMerge: Toward Efficient Data Placement in Shared Heterogeneous Memory Systems. In *Proceedings of the International Symposium on Memory Systems* (Alexandria, Virginia) *(MEMSYS '17)*. Association for Computing Machinery, New York, NY, USA, 251–261. https://doi.org/10.1145/3132402.3132418

[10] T. D. Doudali and A. Gavrilovska. 2019. Mnemo: Boosting Memory Cost Efficiency in Hybrid Memory Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 412–421.

[11] Subramanya R. Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. 2016. Data Tiering in Heterogeneous Memory Systems. In *Proceedings of the Eleventh European Conference on Computer Systems* (London, United Kingdom) *(EuroSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 15, 16 pages. https://doi.org/10.1145/2901318.2901344

[12] Vishal Gupta, Min Lee, and Karsten Schwan. 2015. HeteroVisor: Exploiting Resource Heterogeneity to Enhance the Elasticity of Cloud Platforms. In *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Istanbul, Turkey) *(VEE '15)*. Association for Computing Machinery, New York, NY, USA, 79–92. https://doi.org/10.1145/2731186.2731191

[13] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, and Steven Swanson. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. arXiv:1903.05714 [cs.DC]

[14] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. 2017. HeteroOS: OS Design for Heterogeneous Memory Management in Datacenter. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 521–534. https://doi.org/10.1145/3079856.3080245

[15] Vamsee Reddy Kommareddy, Simon David Hammond, Clayton Hughes, Ahmad Samih, and Amro Awad. 2019. Page Migration Support for Disaggregated Non-Volatile Memories. In *Proceedings of the International Symposium on Memory Systems* (Washington, District of Columbia) *(MEMSYS '19)*. Association for Computing Machinery, New York, NY, USA, 417–427. https://doi.org/10.1145/3357526.3357543

[16] Milind Kulkarni, Martin Burtscher, Calin Casçaval, and Keshav Pingali. 2009. Lonestar: A Suite of Parallel Irregular Programs. In *ISPASS '09: IEEE International Symposium on Performance Analysis of Systems and Software* (Boston, MA, USA). http://iss.ices.utexas.edu/Publications/Papers/ispass2009.pdf

[17] Youngjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and Emmett Witchel. 2016. Coordinated and Efficient Huge Page Management with Ingens. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 705âĂŞ721.

[18] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu. 2017. Utility-Based Hybrid Memory Management. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 152–165.

[19] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. 2009. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *Proceedings of the 36th Annual International Symposium on Computer Architecture* (Austin, TX, USA) *(ISCA '09)*. Association for Computing Machinery, New York, NY, USA, 267–278. https://doi.org/10.1145/1555754.1555789

[20] Kevin Lim, Yoshio Turner, Jose Renato Santos, Alvin AuYoung, Jichuan Chang, Parthasarathy Ranganathan, and Thomas F. Wenisch. 2012. System-Level Implications of Disaggregated Memory. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA '12)*. IEEE Computer Society, USA, 1–12. https://doi.org/10.1109/HPCA.2012.6168955

[21] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh. 2015. Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Vol. 00. 126–136. https://doi.org/10.1109/HPCA.2015.7056027

[22] Onkar Patil, Latchesar Ionkov, Jason Lee, Frank Mueller, and Michael Lang. 2019. Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using intel optane DC persistent memory modules. In *Proceedings of the International Symposium on Memory Systems, MEMSYS 2019, Washington, DC, USA, September 30 - October 03, 2019*. ACM, 288–303. https://doi.org/10.1145/3357526.3357541

[23] A. J. PeÃśa and P. Balaji. 2014. Toward the efficient use of multiple explicitly managed memory subsystems. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*. 123–131.

[24] A. Prodromou, M. Meswani, N. Jayasena, G. Loh, and D. M. Tullsen. 2017. MemPod: A Clustered Architecture for Efficient and Scalable Migration in Flat Address Space Multi-level Memories. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 433–444.

[25] Du Shen, Xu Liu, and Felix Xiaozhu Lin. 2016. Characterizing Emerging Heterogeneous Memory. *SIGPLAN Not.* 51, 11 (June 2016), 13–23. https://doi.org/10.1145/3241624.2926702

[26] Kai Wu, Yingchao Huang, and Dong Li. 2017. Unimem: Runtime Data Managementon Non-volatile Memory-based Heterogeneous Main Memory. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '17)*. ACM, New York, NY, USA, Article 58, 14 pages. https://doi.org/10.1145/3126908.3126923

[27] Kai Wu, Jie Ren, and Dong Li. 2018. Runtime Data Management on Non-volatile Memory-based Heterogeneous Memory for Task-parallel Programs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis* (Dallas, Texas) *(SC '18)*. IEEE Press, Piscataway, NJ, USA, Article 31, 13 pages. http://dl.acm.org/citation.cfm?id=3291656.3291698