DynaHeap: Dynamic Division of DRAM between Heterogeneous Managed Heaps

Iacovos G. Kolokasis*†
FORTH-ICS, Greece

Shoaib Akram‡
ANU, Australia

Foivos S. Zakkak
Red Hat, Inc.

Polyvios Pratikakis*†
FORTH-ICS, Greece

Angelos Bilas*†
FORTH-ICS, Greece

1 Motivation

With the high growth of data, managed big data frameworks, such as Spark [21], Giraph [12], and Flink [6], need to process larger datasets than server memory. However, the DRAM available within each single server scales slower than the data growth rate due to physical scaling limitations [11]. For this purpose, existing solutions extend the managed heap of big data applications over block-addressable fast storage devices (e.g., NVMe SSDs) [3, 4, 7, 8, 14], byte-addressable non-volatile memories (NVM) [1, 2, 9, 15, 18–20], or remote memory [10, 16, 17]. While these alternatives offer higher capacity than DRAM, they have higher latency and lower throughput, constituting a slower memory tier. Existing systems use three main strategies for organizing the managed heap over the fast and slow memory tiers: (1) uniform managed heap with caching, (2) partitioned managed heap without caching, and (3) partitioned managed heap with caching.

Systems in the first category [10, 16, 17] allocate the managed heap over the slow tier and use the fast tier as a cache. The OS hides memory tiers' heterogeneity and transparently fetches objects from the slow to the fast tier. This transparency eliminates the need for the JVM to maintain extra data structures to track objects' locations within the memory hierarchy and to adjust object references during promotions or demotions between the tiers. However, this approach leads to high garbage collection (GC) cost because the garbage collector scans objects in the slow tier, resulting in excessive swapping.

Systems in the second category [1, 2, 9, 15, 18, 19] partition the memory address space into fast and slow tiers, reducing swapping. They allocate the young generation and a portion of the old generation of the managed heap on the fast tier and the remaining on the slow tier. Instead of page swapping, they explicitly move objects between the fast and the slow tiers, which requires updating their references. This reference adjustment becomes prohibitively expensive for frequent object relocation as the garbage collector must scan objects in the slow tier to update their references. Even using lazy reference adjustment with *load reference barriers* [18], application performance decreases [5, 13].

Systems in the last category [3, 4, 7, 8, 14] overcome reference adjustment overheads and avoid scanning the slow tier. They allocate a primary managed heap (H1) over the fast tier and a second managed heap (H2) over the slow tier, reserving a portion of the fast tier as a cache for H2. The garbage collector maintains only cross-heap references while it avoids scanning objects in the slow tier, reducing GC time. However, these approaches divide the fast tier between H1 and the cache for H2 statically at JVM launch, leading to two main problems.

Problem #1: Requiring hand-tuning configuration. Finding which portion of the fast tier must be reserved as a cache to yield good performance requires iterative adjustments and experimentation. Users may need to search for a suitable configuration whenever they change dataset size or application. These experimentations are time-consuming and impractical in real-life deployments.

Problem #2: Changing application behavior. Applications have dramatically different memory requirements at different periods. H1 should use enough space in the fast tier to avoid memory pressure and frequent GC cycles. However, increasing the cache for H2 results in faster access to objects on the slow tier. Thus, the static division of the fast tier between H1 and the cache for H2 cannot adapt to dynamic changing application behavior.

2 DynaHeap Design Overview

We propose DynaHeap[1], a system that dynamically divides a fixed of DRAM budget between the primary heap (H1) and cache for the second heap (H2). DynaHeap treats applications as black boxes and defines distinct rules for memory tuning between H1 and the H2 cache. It determines whether to increase H1 or the H2 cache at runtime by tracking GC and I/O time. DynaHeap uses an adaptation mechanism that takes decisions to repartition DRAM in each minor GC cycle when mutator threads are stopped to avoid extra pauses and synchronization overheads. Although OS memory reclamation leads to memory movement between H1 and H2 cache, this is performed on-demand, introducing an inherent delay in observing the resizing action impact. For this purpose, we extend DynaHeap adaptation mechanism to a finite state machine that includes wait states, used to stop making new decisions until their effect occurs.

---

*Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS), Greece

†Department of Computer Science, University of Crete, Greece

‡Australian National University, Australia

[1] This work is partially funded by a grant from the European Union's Horizon 2020 programme under grant agreement No 10048318 (AERO).

References

[1] Shoaib Akram, Jennifer B. Sartor, Kathryn S. McKinley, and Lieven Eeckhout. Write-rationing garbage collection for hybrid memories. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '18, pages 62–77, New York, NY, USA, 2018. Association for Computing Machinery.

[2] Shoaib Akram, Jennifer B. Sartor, Kathryn S. McKinley, and Lieven Eeckhout. Crystal gazer: Profile-driven write-rationing garbage collection for hybrid memories. *SIGMETRICS Perform. Eval. Rev.*, 47(1):21–22, December 2019.

[3] Michael D. Bond and Kathryn S. McKinley. Tolerating memory leaks. In *Proceedings of the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications*, OOPSLA '08, pages 109–126, New York, NY, USA, 2008. Association for Computing Machinery.

[4] Kim T. Briggs, Baoguo Zhou, and Gerhard W. Dueck. Cold object identification in the java virtual machine. *Software: Practice and Experience*, 47(1):79–95, 2017.

[5] Zixian Cai, Stephen M. Blackburn, Michael D. Bond, and Martin Maas. Distilling the real cost of production garbage collectors. In *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, ISPASS '22, pages 46–57. IEEE Computer Society Press, 2022.

[6] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering*, 38(4), 2015.

[7] Iacovos G. Kolokasis, Giannos Evdorou, Shoaib Akram, Christos Kozanitis, Anastasios Papagiannis, Foivos S. Zakkak, Polyvios Pratikakis, and Angelos Bilas. Teraheap: Reducing memory pressure in managed big data frameworks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS 2023, page 694–709, New York, NY, USA, 2023. Association for Computing Machinery.

[8] Iacovos G. Kolokasis, Anastasios Papagiannis, Polyvios Pratikakis, Angelos Bilas, and Foivos Zakkak. Say goodbye to off-heap caches! on-heap caches using memory-mapped i/o. In *Proceedings of the 12th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage '20, USA, 2020. USENIX Association.

[9] Zhe Li and Mingyu Wu. Transparent and lightweight object placement for managed workloads atop hybrid memories. In *Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '22, pages 72–80, New York, NY, USA, 2022. Association for Computing Machinery.

[10] Haoran Ma, Shi Liu, Chenxi Wang, Yifan Qiao, Michael D. Bond, Stephen M. Blackburn, Miryung Kim, and Guoqing Harry Xu. Mako: A low-pause, high-throughput evacuating collector for memory-disaggregated datacenters. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI '22, pages 92–107, New York, NY, USA, 2022. Association for Computing Machinery.

[11] SeongJae Park, Madhuparna Bhowmik, and Alexandru Uta. Daos: Data access-aware operating system. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '22, pages 4–15, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Sherif Sakr, Faisal Moeen Orakzai, Ibrahim Abdelaziz, and Zuhair Khayyat. *Large-Scale Graph Processing Using Apache Giraph*. Springer Publishing Company, Incorporated, 1st edition, 2017.

[13] Kunal Sareen and Stephen Michael Blackburn. Better understanding the costs and benefits of automatic memory management. In *Proceedings of the 19th International Conference on Managed Programming Languages and Runtimes*, MPLR '22, page 29–44, New York, NY, USA, 2022. Association for Computing Machinery.

[14] Yan Tang, Qi Gao, and Feng Qin. {LeakSurvivor}: Towards safely tolerating memory leaks for {Garbage-Collected} languages. In *2008 USENIX Annual Technical Conference (USENIX ATC 08)*, USENIX ATC '08, pages 307–320, USA, 2008. USENIX Association.

[15] Chenxi Wang, Huimin Cui, Ting Cao, John Zigman, Haris Volos, Onur Mutlu, Fang Lv, Xiaobing Feng, and Guoqing Harry Xu. Panthera: Holistic memory management for big data processing over hybrid memories. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 347–362. Association for Computing Machinery, June 2019.

[16] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. Semeru: A memory-disaggregated managed runtime. In *14th USENIX Symposium on Operating Systems Design and*

Implementation, OSDI '20, pages 261–280, USA, 2020. USENIX Association.

[17] Chenxi Wang, Haoran Ma, Shi Liu, Yifan Qiao, Jonathan Eyolfson, Christian Navasca, Shan Lu, and Guoqing Harry Xu. MemLiner: Lining up tracing and application for a Far-Memory-Friendly runtime. In *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, pages 35–53, USA, 2022. USENIX Association.

[18] Albert Mingkun Yang, Erik Österlund, Jesper Wilhelmsson, Hanna Nyblom, and Tobias Wrigstad. Thingc: Complete isolation with marginal overhead. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management*, ISMM '20, pages 74–86, New York, NY, USA, 2020. Association for Computing Machinery.

[19] Yanfei Yang, Mingyu Wu, Haibo Chen, and Binyu Zang. Bridging the performance gap for copy-based garbage collectors atop non-volatile memory. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, page 343–358. Association for Computing Machinery, April 2021.

[20] Litong You, Tianxiao Gu, Shengan Zheng, Jianmei Guo, Sanhong Li, Yuting Chen, and Linpeng Huang. Jpdheap: A jvm heap design for pm-dram memories. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 31–36, 2021.

[21] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud '10, USA, 2010. USENIX Association.