SmartSweep: Efficient Space Reclamation in Tiered Managed Heaps

Iacovos G. Kolokasis*†‡
FORTH-ICS, Greece
kolokasis@ics.forth.gr

Konstantinos Delis*†‡
FORTH-ICS, Greece
konstadelis@ics.forth.gr

Shoaib Akram§
ANU, Australia
shoaib.akram@anu.edu.au

Foivos S. Zakkak
Red Hat, Ireland
fzakkak@redhat.com

Polyvios Pratikakis†‡
FORTH-ICS, Greece
polyvios@ics.forth.gr

Angelos Bilas†‡
FORTH-ICS, Greece
bilas@ics.forth.gr

Abstract

Using remote memory for the Java heap enables big data analytics frameworks to process large datasets. However, the Java Virtual Machine (JVM) runtime struggles to maintain low network traffic during garbage collection (GC) and to reclaim space efficiently. To reduce GC cost in big data analytics, systems group long-lived objects into regions and excludes them from frequent GC scans, regardless of whether the heap resides in local or remote memory. Recent work uses a dual-heap design, placing short-lived objects in a local heap and long-lived objects in a remote region-based heap, limiting GC activity to the local heap. However, these systems avoid scanning by reclaiming remote heap space only when regions are fully garbage, an inefficient strategy that delays reclamation and risks out-of-memory (OOM) errors.

In this paper, we propose SmartSweep, a system that uses approximate liveness information to balance network traffic and space reclamation in remote heaps. SmartSweep adopts a dual-heap design and avoids scanning or compacting objects in the remote heap. Instead, it estimates the amount of garbage in each region without accessing the remote heap and selectively transfers regions with many garbage objects back to the local heap for reclamation. Preliminary results with Spark and Neo4j show that SmartSweep achieves performance comparable to TeraHeap, which reclaims remote objects lazily, while reducing peak remote memory usage by up to 49% and avoiding OOM errors.

*Both authors contributed equally to this work.
†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

MPLR '25, Singapore, Singapore

ACM ISBN 979-8-4007-2149-6/25/10
https://doi.org/10.1145/3759426.3760981

CCS Concepts: • **Software and its engineering** → **Memory management**; **Garbage collection**; **Runtime environments**; • **Information systems** → *Data analytics*; • **Computer systems organization** → **Cloud computing**.

Keywords: Disaggregated Memory, Remote Memory, Memory Management, Garbage Collection, Big Data Systems

ACM Reference Format:
Iacovos G. Kolokasis, Konstantinos Delis, Shoaib Akram, Foivos S. Zakkak, Polyvios Pratikakis, and Angelos Bilas. 2025. SmartSweep: Efficient Space Reclamation in Tiered Managed Heaps. In *Proceedings of the 22nd ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes (MPLR '25), October 12–18, 2025, Singapore, Singapore.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3759426.3760981

1 Introduction

Big data analytics frameworks running on Java Virtual Machine (JVM) runtimes, such as Spark [48] and Neo4j Graph Data Science [18], are widely used for large-scale data processing. These frameworks demand significant memory resources, as their computations produce large volumes of long-lived objects (e.g., cached intermediate results). On the other hand, scaling DRAM within a single server is increasingly impractical, with DRAM representing up to 37% of total server ownership cost (TCO) in hyperscale data centers [12, 27, 31, 38, 43]. Far-memory techniques offer a scalable manner to use remote, idle memory [13].

In JVM runtimes, garbage collection (GC) over remote memory incurs high network traffic due to the cost of remote heap access. Prior work has tackled GC cost for big data analytics frameworks in local DRAM environments by isolating long-lived objects into regions excluded from frequent scans, often guided by application-level hints [8, 15, 42]. These techniques assume a single managed heap allocated entirely in local memory. In far-memory systems, the heap typically resides in remote memory, with local DRAM acting only as a page cache. As a result, while excluded regions limit GC operations, the garbage collector must still scan and compact the full remote heap, when it needs to reclaim memory.

To address the limitations of single-heap designs, prior work has introduced a dual-heap design [19, 21, 25, 26, 45, 47].

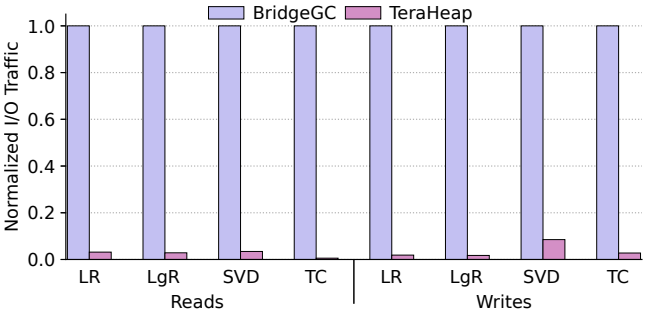


Figure 1. Remote I/O traffic for single-heap (BridgeGC) and dual-heap designs (TeraHeap) in Spark.

In the context of remote memory, the primary heap (H1) resides in local DRAM and the secondary heap (H2) for long-lived objects in remote memory. This design restricts GC activity to H1, avoiding expensive GC scans and compactions in remote memory. As shown in Figure 1, TeraHeap [19] using a dual-heap design reduces network I/O traffic by up to 177× compared to BridgeGC [42], which uses a single heap over remote memory for Spark workloads described in Section § 4. TeraHeap places Spark's cached intermediate results in H2, while BridgeGC avoids scanning these objects during GC.

However, dual-heap designs face the challenge of how to reclaim memory efficiently in H2. Some systems [25, 26, 45, 47] periodically scan H2 to reclaim unreachable objects, but this approach incurs high network traffic in the case of remote memory. Others, such as TeraHeap [19, 20], reclaim memory only when all objects in a region in H2 are unreachable. This design is suitable for storage devices that provide high, low-cost capacity, where delayed reclamation is tolerable. In contrast, remote memory is limited and requires timely reclamation, making such a lazy approach impractical, resulting in out-of-memory (OOM) errors.

In this work, we propose SmartSweep, a system that efficiently reclaims space in remote memory within dual-heap design without incurring full GC scans in remote managed heap (H2). SmartSweep partitions H2 into regions and uses approximate region-level information to quickly identify those containing a high proportion of garbage. This approach enables timely reclamation with low overhead. Our design addresses the following three challenges:

Finding dead objects without scanning H2. Traversing the object graph to identify live objects in H2 requires random remote memory access, which significantly increases network traffic. To avoid this overhead, SmartSweep estimates region liveness without accessing H2 objects. It collects statistics during H1's liveness analysis and leverages forward references from H1 to H2, along with updates to H2 objects by application (mutator) threads. SmartSweep then ranks regions for reclamation using a policy that correlates forward references and mutator updates with the number of objects in each H2 region, prioritizing those likely to contain mostly garbage.

Reclaiming space in H2. Compacting objects in remote memory is costly, as it triggers frequent page swaps. Prior work [24] offloads this task to remote servers, but it depends on spare CPU resources, which are often oversubscribed in datacenters [13] and incurs additional overhead from maintaining load barriers to update object references during mutator access [49]. SmartSweep avoids these costs by identifying regions with high garbage content and moving their objects from H2 to H1. Since GC overhead is dominated by live object processing, moving mostly dead objects to H1 adds little to no cost in subsequent GC cycles.

Maintaining cross-region references. While SmartSweep avoids scanning H2 to identify live objects, reclaiming space in H2 by moving a region back to H1 introduces a new challenge: we must update any references to objects in the reclaimed region without scanning the entire H2. One option is to relocate the entire transitive closure of the region to avoid dangling references. However, this can be costly if dependencies are widespread. We find that up to 70% of H2 regions are referenced by only two other regions due to our placement strategy that groups the transitive closure of long-lived objects into the same region during migration. This limited connectivity makes it more efficient to update references in place. Thus, we track cross-region references with a card table, updated during GC and by just-in-time (JIT) compiler post-write barriers on new references.

We implement an early prototype of SmartSweep by extending TeraHeap, the state-of-the-art dual-heap system, which is implemented in OpenJDK 17, a long-term support release widely adopted by legacy applications. To evaluate our system, we use Spark and Neo4j Graph Data Science (GDS), two popular big data analytics frameworks. Our prototype currently moves only primitive types, arrays, and leaf objects to remote memory. Although this design is simplified, these object categories account for more than 60% of heap usage in our benchmarks, showing that the prototype captures the dominant memory behavior. As a result, SmartSweep lowers peak remote memory usage by 49% compared to TeraHeap, achieves similar performance, and avoids OOM errors.

2 SmartSweep Design

The main goal of SmartSweep, is to reclaim dead objects in remote managed heaps without performing GC scans. As shown in Figure 2, SmartSweep uses a two-heap design architecture with a primary heap (H1) in local DRAM and a second region-based managed heap (H2) for long-lived objects in remote memory. Typically, H2 is mapped over DRAM using memory-mapped I/O (mmio), such as Linux

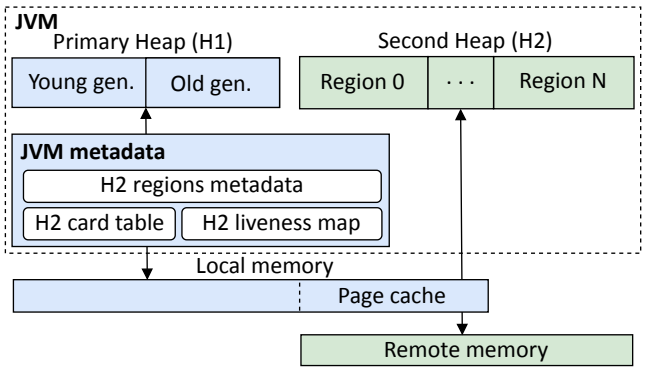


Figure 2. Overview of SmartSweep

mmap(). Thus, H2 accesses are served through the page cache, ensuring DRAM-level latency in case of page cache hits without requiring object reference updates. Only transfers between H1 and H2 necessitate updating object references.

2.1 Finding Dead Objects in Remote Heap

Scanning H2 to find live objects is slow. For this reason, we organize objects in H2 into regions, treating all objects within a region as a single unit. SmartSweep stores the metadata for each region into a region array in local memory (see Figure 3). These metadata contain metrics that enable SmartSweep to estimate the amount of garbage in each region without scanning individual H2 objects. For the estimation, SmartSweep identifies regions that contain objects directly referenced by H1 objects and monitors changes in the object references within each region. Tracking forward references from H1 to H2 pinpoints regions with live objects, while observing reference updates detects modifications in the object graph that may convert previously live objects into garbage.

Forward references (H1 to H2). To estimate live objects in each region, SmartSweep tracks where forward references from H1 land in H2. This spatial information is crucial because simple reference counting can be misleading as a region may contain one highly referenced object while most objects are garbage. To capture this detail, SmartSweep employs a liveness map, a byte array allocated in local DRAM where each byte corresponds to a fixed-size segment of H2. At the start of each marking phase, the used bits in each region's metadata are reset. When a forward reference to an H2 object is encountered, SmartSweep checks whether the corresponding byte is marked; if not, it increments the liveness counter in the region's metadata (Figure 3) and marks corresponding byte in the liveness map. Additionally, the region is flagged as used, and if it contains references to objects in other regions, SmartSweep traverses the dependency list to mark those regions as used as well. At the end of the marking phase, any region whose used bit remains unset is reclaimed. To estimate garbage density, we rank regions

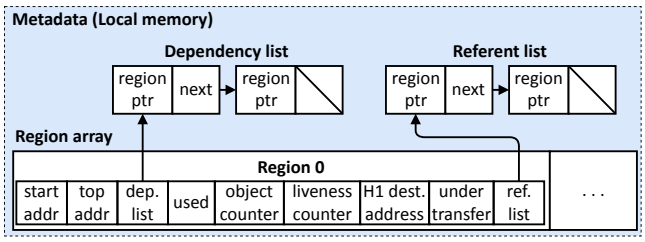


Figure 3. Metadata for each region in local memory.

by the ratio of the liveness counter to the total number of objects. In our experiments, each card maps an 4 KB segment of H2 to align with page size accesses.

Monitoring changes in inter-region references. We maintain a card table for H2 to track updates to object references. The card table is implemented as a byte array in local DRAM, with each byte corresponding to a fixed-size segment of H2, similar to the vanilla JVM. A clean card indicates that no changes have been made in that segment, whereas a dirty card signals that mutator threads have updated an object's reference through post-write barriers in both the interpreter and the JIT compilers. These updates reflect modifications in the object graph within that region, either by creating a new reference from H2 to H1 or by altering references among H2 objects. During each GC cycle, we scan the card table to count the number of dirty cards in each region and record this value in the region's metadata. In addition, we update the backward references from H2 to H1 to reflect object movements in H1.

2.2 Selecting Regions for Reclamation

At the end of the marking phase, SmartSweep marks for transfer all regions with a heuristic value U below a designated threshold and saves references to these regions in a data structure in the JVM. To guide region selection for movement from H2 to H1, SmartSweep employs a policy that combines two key metrics: the liveness ratio, derived from forwarding references, and the dirty-card ratio, which reflects objects mutability. Each region is assigned a unified score U by normalizing both metrics and applying a weighted formula:

$$U = \alpha * liveness\_ratio + (1 - \alpha) * (1 - dirty\_ratio) \quad (1)$$

These regions are then sorted based on their heuristic values. A lower threshold ensures that we primarily select regions with a high concentration of garbage objects, whereas a higher threshold also includes regions with a lower proportion of garbage. In general, increasing the threshold reclaims more memory from H2, as it leads to transferring more objects back to H1. However, this also introduces higher performance overhead, since live objects may be unnecessarily moved, increasing memory pressure in H1 and leading to

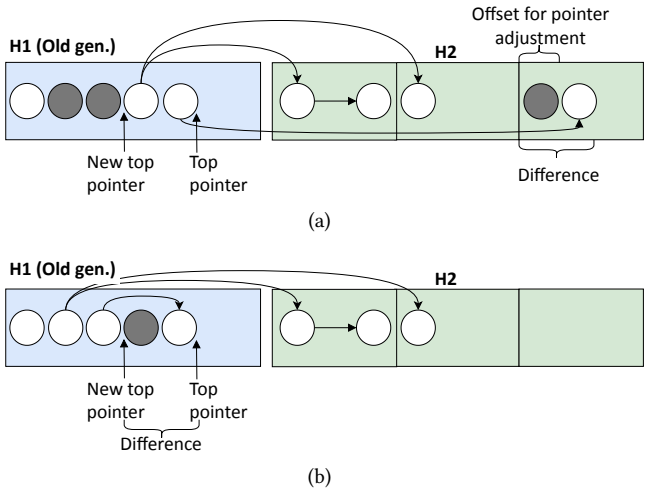


Figure 4. (a) After marking, white (live) and solid (dead) objects are identified and destination offsets are computed. (b) H1 is compacted, the chosen H2 region is moved, and references are updated.

longer GC pauses and more frequent collections. Notably, these objects can later be transferred back to H2.

Transferring regions from H2 to H1. As soon as the precompaction phase of the garbage collector concludes, destination addresses in H1 are assigned to each H2 region scheduled for transfer. This process begins by assigning the address at the end of the old generation, as indicated by the new top pointer shown in Figure 4(a). Each subsequent region that is to be transferred is assigned a destination address calculated as the new top pointer plus the cumulative difference of all so far transferred regions.

During the compaction phase, while updating pointers to the new object locations, SmartSweep checks whether an object resides in H2 and belongs to a region scheduled for transfer to H1. For each such object, SmartSweep calculates its offset from the start of its H2 region and adds it to the assigned destination address to determine its new location in H1. Once compaction is complete, the H2 regions are copied to their designated H1 destinations, as shown in Figure 4(b). SmartSweep then iterates over all transferred objects to fix their references and mark them for return to H2 in the next GC, after dead objects have been reclaimed. Finally, SmartSweep updates the top pointer of the old generation to reflect the end of the last transferred region.

Maintaining cross-region references. To maintain correctness when moving a region from H2 to H1, SmartSweep must update references from other regions that point into it. SmartSweep extends the H2 card table to track cross-region references and marks relevant cards whenever objects are moved from H1 to H2 or when mutator threads create new

Table 1. Configurations of the workloads.

	GB	DRAM Size	Data-set Size	Tera Heap H1	Smart Sweep H1
Spark	SVDPlusPlus (SVD)	28	2	12	12
	TriangleCounts (TR)	59	2	43	43
	Linear Regression (LR)	43	256	27	27
	Logistic Regression (LgR)	43	256	27	27
Neo4j	CDLP	16	70	14	14

references. Each region maintains a dependency list (regions that reference it) and a referent list (regions it references). During region transfer, SmartSweep uses the dependency list to identify which regions may contain stale references, scanning only their marked cards to locate and update pointers. After updates, the region is removed from the dependency lists of all referencing regions, ensuring consistency while avoiding full-region scans or transitive closure transfers.

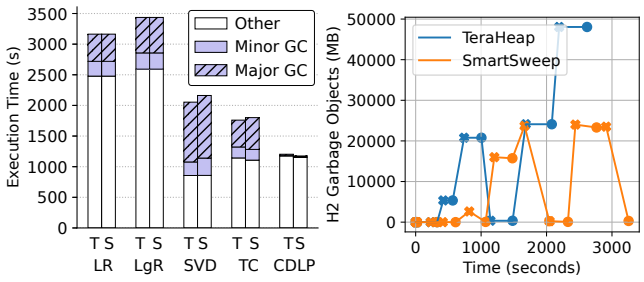
3 Preliminary Implementation

We implement an early prototype of SmartSweep by extending TeraHeap, which is built on OpenJDK 17, to evaluate the core ideas and validate its effectiveness. TeraHeap extends the Parallel Scavenge garbage collector (ParallelGC) to support a two-heap architecture. Our prototype leverages the hint-based interface of TeraHeap to migrate long-lived objects to remote memory. At this stage, we focus on transferring only primitive types, arrays, and leaf objects—those whose fields are exclusively primitive. This ensures only forward references from H1 to H2, simplifying our implementation and avoiding cross-region references in H2. Despite these simplifications, our evaluation across a range of real-world big data analytics benchmarks shows that primitive and leaf objects account for over 60% of heap usage, demonstrating that our prototype captures the dominant memory behavior. SmartSweep accesses remote memory via NVMe-over-fabric (NVMe-oF) [28], using OS support for memory-mapped I/O.

4 Preliminary Evaluation

Using the SmartSweep prototype implementation, we perform a set of experiments to estimate: (1) performance of SmartSweep compared to TeraHeap (baseline), and (2) the improvement in space utilization compared to TeraHeap.

Experimental setup. We ran all experiments in 4 dual socket servers with two Intel(R) Xeon(R) E5-2630 v3 CPUs at 2.4GHz, with 8 physical cores and 16 hyper-threads each (32 total hyper-threads), 256 GB of DDR4 DRAM and Ubuntu v24.02, with Linux kernel 5.14. We use the one server to run the application while the rest we use them as remote memory via NVMe-of. In our experiments we use OpenJDK-17, Spark v3.3.0, and Neo4j-GDS using Neo4j v.5.13. For Spark we use one executor with eight threads. For GDS-Neo4j, we use four



(a) Overall performance (b) Space waste

Figure 5. (a) Overall performance of TeraHeap and SmartSweep. (b) Space wasted in H2 due to dead objects: "x" and "o" mark the start and end of each full GC cycle, respectively.

Table 2. Runs with limited H2 size

	Execution Time (s)	H1 to H2 (GB)	H2 to H1 (GB)
TeraHeap	OOM	89.5	0
SmartSweep	2034	131	31.7

threads, as this is the maximum number of threads supported by the GDS community edition [18]. In all setups, we use eight GC threads. To reduce variability, we disable swap and set the CPU scaling governor to *performance*. When necessary for each experiment, we limit the available DRAM capacity using *cgroups*. Table 1 shows the configuration for each workload. For all experiments, we determine the minimum H2 size required for TeraHeap to run successfully.

Overall performance: Figure 5 (a) shows the execution time breakdown of the applications, divided into *MinorGC*, *MajorGC*, and application processing time. SmartSweep delivers performance comparable to TeraHeap across all configurations, while reducing wasted space in H2 by 50%. This is because *MajorGC* time depends on the size of live objects, not the volume of garbage, resulting in minimal overhead.

Space utilization improvement. Figure 5 (b) shows the execution of SVD and the accumulation of garbage objects in H2. For this analysis, we allow the garbage collector to scan H2 to determine the exact number of live and dead objects. Since live objects are actively used by the application, only garbage contributes to wasted space if not reclaimed. In TeraHeap, wasted space peaks at the end of execution, reaching 48 GB. In contrast, SmartSweep reduces peak wasted space to 24.5 GB, a 49% reduction. Due to space constraints, we show results for SVD only, but other workloads, suchs as CDLP exhibit similar trends.

We aim to simulate the behavior of a long-running application that spans several hours, which would eventually exhaust remote memory capacity if dead objects are not reclaimed in H2. For this purpose, instead of increasing the dataset size and running SVD for several hours, we simulated this scenario by constraining the available H2 size. We set H1 and H2 to be 64 GB and 97 GB, respectively. Table 2 summarizes our derived results. TeraHeap was unable to complete the execution due to an OOM error, indicating that H2 regions contain a portion of live objects, and the garbage collector cannot reclaim sufficient space to allow continued execution. In contrast, SmartSweep successfully completed its execution.

5 Discussion

Identifying objects to move to remote memory. Beyond the hint-based approach we use to identify long-lived objects for transfer to H2, a more transparent solution, as proposed in previous work [3, 4, 22, 45], would involve instrumenting load and store instructions to collect per-object access statistics. This could enable automatic identification of hot and cold objects without requiring developer intervention. However, such instrumentation introduces non-negligible runtime overhead and remains a direction for future exploration. Regardless of how objects are selected for placement in remote memory, efficient space reclamation remains a fundamental challenge. Even with accurate object classification, an inefficient GC can lead to wasted memory in remote tiers. Object selection and space reclamation are orthogonal concerns—one optimizes placement, while the other ensures remote memory remains effectively usable over time.

Page-level vs. object-level accesses. Our system relies on page-level accesses over remote memory, leveraging the operating system's virtual memory mechanisms. While object-level accesses are possible within the JVM, they require modifying the interpreter and JIT compilers to insert load barriers on every object access, introducing significant overhead [49]. Concurrent garbage collectors, such as ZGC [46] execute load barriers only at safepoints to reduce the overhead, as inserting load barriers on every load instruction would be prohibitively expensive [44]. In contrast, page-level accesses take advantage of hardware-managed virtual-to-physical translation, avoiding software overhead. If a valid page table entry exists, memory access is handled entirely in hardware, ensuring low latency [29]. This approach eliminates costly per-object lookups [30] and enables efficient remote memory integration without requiring modifications to the JVM interpreter or JIT compilers.

Remote memory access paths and applicability to CXL. Remote memory can be exposed to applications in different ways, each with trade-offs. One approach is using NVMeoF RAMDisk, which maps remote memory as block storage. This simplifies deployment but incurs unnecessary I/O overhead, as each memory access goes through the storage

stack. A more efficient alternative is a custom page fault handling path integrated with the memory manager [1]. This approach bypasses the storage stack and directly leverages RDMA or memory-mapped I/O to reduce access latency and preserve the abstraction of a single address space. Our work—SmartSweep—uses such a design, making it directly applicable not only to RDMA-backed setups but also to CXL-based systems. In CXL environments, where remote memory appears as byte-addressable and cache-coherent, SmartSweep's region-level space reclamation remains effective. CXL reduces the latency gap between local and remote memory, but it doesn't eliminate the need for efficient garbage collection in disaggregated memory tiers. SmartSweep complements CXL by minimizing GC overhead in large, memory-intensive workloads, ensuring scalable and efficient memory use regardless of the underlying fabric.

6 Related Work

Managed heaps over remote memory. Semeru [39], MemLiner [40], and Mako [24] allocate the managed heap entirely over remote memory and use local DRAM as a cache, modifying the Linux kernel swapping mechanism to evict pages remotely. To reduce page swappings due to GC operations, Semeru offloads object scanning to light JVMs running on remote servers. Memliner reorganizes the access order of the GC threads to follow a similar memory-access path with mutator threads. However, Semeru and Memliner's evacuation process involves retrieving objects from remote servers, transferring them to local servers, and rewriting them back to remote servers, causing high I/O network and GC pauses. Mako offloads concurrent object scanning and evacuation to memory servers. It uses a distributed data structure named *Heap Indirection Table* (HIT) to track the new object location in remote servers, introducing functional, yet expensive load reference barriers on every load operation. While these techniques may reduce traffic, they incur extra CPU overhead on the remote servers. In contrast, SmartSweep reduces network overhead without consuming additional remote CPU resources. Polar [25] uses a two-heap design to prevent the garbage collector from scanning remote memory. However, it employs an agent that periodically scans and compacts objects in remote memory, introducing long GC pauses, as the remote heap is significantly larger.

Resource disaggregation. Remote memory illustrates a broader trend of resource disaggregation within datacenters [2, 6, 7, 14, 23]. Numerous optimizations and systems, such FaRM [11], and others [2, 5, 9, 10, 16, 17, 32–37, 41], have been created to mitigate remote latency. Nevertheless, they all concentrate on low-level system stacks and neglect the run-time properties of programs. These works focus on remote latency but are orthogonal to SmartSweep's emphasis on space reclamation in remote memory. They are ineffective for applications running on top of managed runtimes. SmartSweep enhances runtime efficiency by concentrating on distant memory and does not necessitate co-redesign assistance from the operating system.

7 Conclusions

This paper addresses the problem of reclaiming dead objects in remote memory without GC scans in two-tiered heap architectures. SmartSweep uses a primary heap (H1) in local DRAM and a second region-based heap (H2) in remote memory. To efficiently reclaim space in H2, SmartSweep transfers regions with a high proportion of dead objects back to H1, where the garbage collector reclaims memory with low overhead. Our evaluation shows that SmartSweep reduces waste space in remote memory by 49% compared to TeraHeap, achieving similar performance and effectively preventing OOM errors. SmartSweep also has the potential to generalize to other managed runtimes (e.g., Python and Go) that rely on GC, enabling efficient memory reclamation in remote memory for big data analytics.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback, which helped improve the final version of this paper. This work was partially supported by the European Union project AERO (grant agreement No. 101092850), the EUPEX project (grant agreement No. 101033975) through the European High-Performance Computing Joint Undertaking (JU), and VMware's University Research Fund. The JU receives support from the European Union's Horizon 2020 research and innovation programme, as well as from France, Germany, Italy, Greece, the United Kingdom, Czech Republic, and Croatia.

References

[1] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novaković, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. 2018. Remote regions: a simple abstraction for remote memory. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 775–787. https://www.usenix.org/conference/atc18/presentation/aguilera

[2] Marcos K. Aguilera, Kimberly Keeton, Stanko Novakovic, and Sharad Singhal. 2019. Designing Far Memory Data Structures: Think Outside the Box. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Bertinoro, Italy) *(HotOS '19)*. Association for Computing Machinery, New York, NY, USA, 120–126. doi:10.1145/3317550.3321433

[3] Shoaib Akram, Jennifer Sartor, Kathryn McKinley, and Lieven Eeckhout. 2019. Crystal Gazer: Profile-Driven Write-Rationing Garbage Collection for Hybrid Memories. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 1, Article 9 (March 2019), 27 pages. doi:10.1145/3322205.3311080

[4] Shoaib Akram, Jennifer B. Sartor, Kathryn S. McKinley, and Lieven Eeckhout. 2018. Write-Rationing Garbage Collection for Hybrid Memories. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '18)*. Association for Computing Machinery, New York, NY, USA, 62–77. doi:10.1145/3192366.3192392

[5] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can far memory improve job throughput?. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 14, 16 pages. doi:10.1145/3342195.3387522

[6] Sebastian Angel, Mihir Nanavati, and Siddhartha Sen. 2020. Disaggregation and the Application. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association. https://www.usenix.org/conference/hotcloud20/presentation/angel

[7] Luiz Andre Barroso. 2011. Warehouse-Scale Computing: Entering the Teenage Decade. In *Proceedings of the 38th Annual International Symposium on Computer Architecture* (San Jose, California, USA) *(ISCA '11)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/2000064.2019527

[8] Rodrigo Bruno, Luís Picciochi Oliveira, and Paulo Ferreira. 2017. NG2C: Pretenuring Garbage Collection with Dynamic Generations for HotSpot Big Data Applications. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on Memory Management (ISMM '17)*. Association for Computing Machinery, New York, NY, USA, 2–13. doi:10.1145/3092255.3092272

[9] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. 2021. Rethinking software runtimes for disaggregated memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 79–92. doi:10.1145/3445814.3446713

[10] Lei Chen, Shi Liu, Chenxi Wang, Haoran Ma, Yifan Qiao, Zhe Wang, Chenggang Wu, Youyou Lu, Xiaobing Feng, Huimin Cui, Shan Lu, and Harry Xu. 2024. A Tale of Two Paths: Toward a Hybrid Data Plane for Efficient Far-Memory Applications. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 77–95. https://www.usenix.org/conference/osdi24/presentation/chen-lei

[11] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA, 401–414. https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi{ć}

[12] Alexander Fuerst, Stanko Novaković, Íñigo Goiri, Gohar Irfan Chaudhry, Prateek Sharma, Kapil Arya, Kevin Broas, Eugene Bak, Mehmet Iyigun, and Ricardo Bianchini. 2022. Memory-harvesting VMs in cloud platforms. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 583–594. doi:10.1145/3503222.3507725

[13] Alexander Fuerst, Stanko Novaković, Íñigo Goiri, Gohar Irfan Chaudhry, Prateek Sharma, Kapil Arya, Kevin Broas, Eugene Bak, Mehmet Iyigun, and Ricardo Bianchini. 2022. Memory-harvesting VMs in cloud platforms. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 583–594. doi:10.1145/3503222.3507725

[14] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2016. Network Requirements for Resource Disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 249–264. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gao

[15] Ionel Gog, Jana Giceva, Malte Schwarzkopf, Kapil Vaswani, Dimitrios Vytiniotis, Ganesan Ramalingan, Derek Murray, Steven Hand, and Michael Isard. 2015. Broom: Sweeping out Garbage Collection from Big Data Systems. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems (HOTOS '15)*. USENIX Association, USA, Article 2, 2 pages.

[16] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient Memory Disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 649–667. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu

[17] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. 2022. Clio: a hardware-software co-designed disaggregated memory system. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 417–433. doi:10.1145/3503222.3507762

[18] Amy E Hodler and Mark Needham. 2022. Graph Data Science Using Neo4j. In *Massive Graph Analytics*. Chapman and Hall/CRC, 433–457.

[19] Iacovos G. Kolokasis, Giannos Evdorou, Shoaib Akram, Christos Kozanitis, Anastasios Papagiannis, Foivos S. Zakkak, Polyvios Pratikakis, and Angelos Bilas. 2023. 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* (Vancouver, BC, Canada) *(ASPLOS '23)*. Association for Computing Machinery, New York, NY, USA, 694–709. doi:10.1145/3582016.3582045

[20] Iacovos G. Kolokasis, Giannos Evdorou, Shoaib Akram, Christos Kozanitis, Anastasios Papagiannis, Foivos S. Zakkak, Polyvios Pratikakis, and Angelos Bilas. 2024. TeraHeap: Exploiting Flash Storage for Mitigating DRAM Pressure in Managed Big Data Frameworks. *ACM Trans. Program. Lang. Syst.* 46, 4, Article 12 (Dec. 2024), 37 pages. doi:10.1145/3700593

[21] Iacovos G. Kolokasis, Anastasios Papagiannis, Polyvios Pratikakis, Angelos Bilas, and Foivos Zakkak. 2020. 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)*. USENIX Association, USA, Article 4, 1 pages.

[22] Zhe Li and Mingyu Wu. 2022. 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* (Virtual, Switzerland) *(VEE '22)*. Association for Computing Machinery, New York, NY, USA, 72–80. doi:10.1145/3516807.3516822

[23] 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 *IEEE International Symposium on High-Performance Comp Architecture*. 1–12. doi:10.1109/HPCA.2012.6168955

[24] Haoran Ma, Shi Liu, Chenxi Wang, Yifan Qiao, Michael D. Bond, Stephen M. Blackburn, Miryung Kim, and Guoqing Harry Xu. 2022. 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* (San Diego, CA, USA) *(PLDI '22)*. Association for Computing Machinery, New York, NY, USA, 92–107. doi:10.1145/3519939.3523441

[25] Dat Nguyen and Khanh Nguyen. 2024. Polar: A Managed Runtime with Hotness-Segregated Heap for Far Memory. In *Proceedings of the 15th ACM SIGOPS Asia-Pacific Workshop on Systems* (Kyoto, Japan) *(APSys '24)*. Association for Computing Machinery, New York, NY, USA, 15–22. doi:10.1145/3678015.3680490

[26] Khanh Nguyen, Lu Fang, Guoqing Xu, Brian Demsky, Shan Lu, Sanazsadat Alamian, and Onur Mutlu. 2016. Yak: A High-Performance Big-Data-Friendly Garbage Collector. In *Proceedings of the 12th USENIX*

Conference on Operating Systems Design and Implementation (OSDI '16). USENIX Association, USA, 349–365.

[27] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. 2018. Welcome to Zombieland: Practical and Energy-Efficient Memory Disaggregation in a Datacenter. In *Proceedings of the Thirteenth European Conference on Computer Systems* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 16, 12 pages. doi:10.1145/3190508.3190537

[28] NVIDIA. [n. d.]. NVIDIA Enterprise Support Portal | What is NVMe over Fabrics? https://enterprise-support.nvidia.com/s/article/what-is-nvme-over-fabrics-x [Online; accessed 2025-04-05].

[29] Anastasios Papagiannis, Manolis Marazakis, and Angelos Bilas. 2021. Memory-mapped I/O on steroids. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 277–293. doi:10.1145/3447786.3456242

[30] Anastasios Papagiannis, Giorgos Xanthakis, Giorgos Saloustros, Manolis Marazakis, and Angelos Bilas. 2020. Optimizing Memory-Mapped I/O for Fast Storage Devices. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '20)*. USENIX Association, USA, Article 56, 15 pages.

[31] SeongJae Park, Madhuparna Bhowmik, and Alexandru Uta. 2022. DAOS: Data Access-Aware Operating System. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing* (Minneapolis, MN, USA) *(HPDC '22)*. Association for Computing Machinery, New York, NY, USA, 4–15. doi:10.1145/3502181.3531466

[32] Yifan Qiao, Chenxi Wang, Zhenyuan Ruan, Adam Belay, Qingda Lu, Yiying Zhang, Miryung Kim, and Guoqing Harry Xu. 2023. Hermit: Low-Latency, High-Throughput, and Transparent Remote Memory via Feedback-Directed Asynchrony. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 181–198. https://www.usenix.org/conference/nsdi23/presentation/qiao

[33] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. 2020. AIFM: High-Performance, Application-Integrated Far Memory. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 315–332. https://www.usenix.org/conference/osdi20/presentation/ruan

[34] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 69–87. https://www.usenix.org/conference/osdi18/presentation/shan

[35] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. 2019. Shoal: A Network Architecture for Disaggregated Racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 255–270. https://www.usenix.org/conference/nsdi19/presentation/shrivastav

[36] David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, and Gustavo Alonso. 2020. StRoM: smart remote memory. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 29, 16 pages. doi:10.1145/3342195.3387519

[37] Brian R. Tauro, Brian Suchy, Simone Campanoni, Peter Dinda, and Kyle C. Hale. 2024. TrackFM: Far-out Compiler Support for a Far Memory World. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 401–419. doi:10.1145/3617232.3624856

[38] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: The next Generation. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 30, 14 pages. doi:10.1145/3342195.3387517

[39] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2020. Semeru: A Memory-Disaggregated Managed Runtime. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. USENIX Association, USA, 261–280. https://www.usenix.org/conference/osdi20/presentation/wang

[40] Chenxi Wang, Haoran Ma, Shi Liu, Yifan Qiao, Jonathan Eyolfson, Christian Navasca, Shan Lu, and Guoqing Harry Xu. 2022. MemLiner: Lining up Tracing and Application for a Far-Memory-Friendly Runtime. In *16th USENIX Symposium on Operating Systems Design and Implementation* (Carlsbad, CA) *(OSDI '22)*. USENIX Association, USA, 35–53. https://www.usenix.org/conference/osdi22/presentation/wang

[41] Chenxi Wang, Yifan Qiao, Haoran Ma, Shi Liu, Wenguang Chen, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2023. Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 161–179. https://www.usenix.org/conference/nsdi23/presentation/wang-chenxi

[42] Yicheng Wang, Lijie Xu, Tian Guo, Wensheng Dou, Hongbin Zeng, Wei Wang, Jun Wei, and Tao Huang. 2025. BridgeGC: An Efficient Cross-Level Garbage Collector for Big Data Frameworks. *ACM Trans. Archit. Code Optim.* (March 2025). doi:10.1145/3722110 Just Accepted.

[43] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. 2022. TMO: Transparent Memory Offloading in Datacenters. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanee, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 609–621. doi:10.1145/3503222.3507731

[44] Filip Wilén. 2023. Throughput Analysis of Safepoint-attached Barriers in a Low-latency Garbage Collector: Analysis of a Compiler Optimization in the HotSpot JVM.

[45] Albert Mingkun Yang, Erik Österlund, Jesper Wilhelmsson, Hanna Nyblom, and Tobias Wrigstad. 2020. ThinGC: Complete Isolation with Marginal Overhead. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management* (London, UK) *(ISMM '20)*. Association for Computing Machinery, New York, NY, USA, 74–86. doi:10.1145/3381898.3397213

[46] Albert Mingkun Yang and Tobias Wrigstad. 2022. Deep Dive into ZGC: A Modern Garbage Collector in OpenJDK. *ACM Trans. Program. Lang. Syst.* 44, 4, Article 22 (Sept. 2022), 34 pages. doi:10.1145/3538532

[47] Litong You, Tianxiao Gu, Shengan Zheng, Jianmei Guo, Sanhong Li, Yuting Chen, and Linpeng Huang. 2021. JPDHeap: A JVM Heap Design for PM-DRAM Memories. In *2021 58th ACM/IEEE Design Automation Conference* (San Francisco, CA, USA) *(DAC '21)*. IEEE, 31–36. doi:10.1109/DAC18074.2021.9586279

[48] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud '10)*. USENIX Association, USA, Article 10, 10 pages.

[49] Wenyu Zhao, Stephen M. Blackburn, and Kathryn S. McKinley. 2022. Low-latency, high-throughput garbage collection. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (San Diego, CA, USA) *(PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 76–91. doi:10.1145/3519939.3523440

Received 2025-06-21; accepted 2025-07-28