APPENDIX A
SPACE UTILIZATION OF LOWER LAYERS

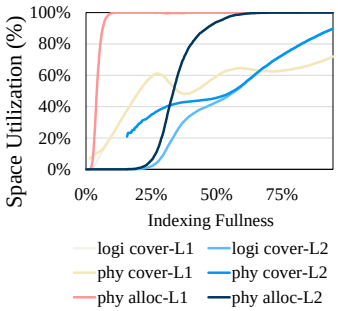


Fig. A1: Space Utilization in TieredHM.

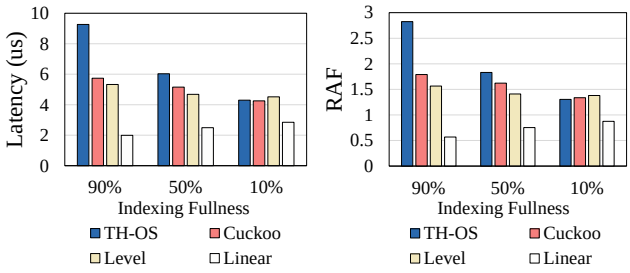
Since TieredHM preallocates all layers during initialization, and data are inserted into upper layers first and then sunk down to lower layers, the data coverage of lower layers is low initially and gradually increases when more data are inserted. However, the data coverage of lower layers in the physical NAND space quickly achieves a reasonable level. Given that MS-SSDs have a large capacity and are cost-effective compared to DRAM, the space utilization issue is less of a concern.

Note that the memory address space seen by applications (e.g., TieredHM) corresponds to the logical space exposed by MS-SSDs. And MS-SSDs employ an internal indirection layer (i.e., flash translation layer, FTL) that manages the mapping from its logical space to physical NAND space and allocates NAND pages on demand. Thus, although the layers of TieredHM are preallocated with memory space, the utilization of physical NAND space is proportional to the actual data volume.

We use the insertion workload of YCSB to demonstrate the data coverage and on-demand physical space allocation of lower layers (i.e., $L1$ and $L2$) of TieredHM. We define *logi cover-X* as the ratio of the inserted data set size to the total capacity (logically allocated space) of the corresponding layer-X. Similarly, we define *phy cover-X* as the ratio of the inserted data set size to the physically allocated space of layer-X. We also define *phy alloc-X* as the ratio of the size of physically allocated space to the total capacity of the corresponding layer-X. The *logi cover-X* and *phy cover-X* reflect the logical and physical space utilization of the corresponding layer, respectively. The *phy alloc-X* indicates how on-demand allocation affects the physical space utilization.

As shown in Figure A1, the data coverage gradually increases with the inserted data set size. In the logically allocated space, the average data coverage of $L1$ and $L2$ is 52.3% and 39.1%, respectively. The logical data coverage of $L2$ is relatively lower since its logical space is preallocated but not actually written with data initially. In the physically allocated space, the average data coverage of $L1$ and $L2$ is 53.3% and 55.2%, respectively. The physical and logical data coverage of $L1$ are similar since its physical space is also fully allocated quickly after the insertion begins. In contrast, $L2$ keeps almost unallocated until 25% indexing fullness. The physical data coverage of $L1$ and $L2$ is larger than 40% when the indexing fullness is beyond 16.7% and 30.4%, respectively. In addition, with the aid of ODM design (migrating data in batches) and on-demand allocation, the physical data coverage of $L2$ starts from 20.9%. Note that the data coverage of $L1$ fluctuates (decreases) since ODM migrates data from $L1$ to $L2$.

APPENDIX B
READ OVERHEAD OF MULTI-LAYERED STRUCTURE



(a) Search Latency. (b) Read Amplification.

Fig. A2: Search (YCSB-C) Overhead Analysis.

To gain deeper insight into the inherent overhead of multi-layered structure in TieredHM, we exclude the optimization effect of architectural designs (i.e., signature array in DRAM, placing top layer in PM, and prefetching enabled inside MS-SSD) in TieredHM, denoted as TH-OS (i.e., only indexing structure). We compare the search performance of TH-OS with single-layered hash schemes such as Linear and Cuckoo Hashing and two-layered Level Hashing on sheer MS-SSD under YCSB-C workloads.

Figure A2 shows the results. TH-OS shows 2.86X, 1.27X, and 1.33X higher latency on average than Linear, Cuckoo, and Level Hashing, with the RAF increasing by 2.96X, 1.23X, and 1.35X on average. The inherent search overhead of the multi-layered structure motivates us to propose novel designs by leveraging storage architecture. Besides, the query overhead is acceptable compared to the huge advantage brought to write efficiency.

APPENDIX C
DIFFERENCES IN WRITE OPERATIONS BETWEEN TIEREDHM AND TREE STRUCTURES

While TieredHM employs a multi-layered structure and hierarchical data movement similar to tree structures (e.g., LSM trees and B-trees), its write operations are essentially distinct. Trees are sorted structures which support range queries. They leverage multi-layered structures to provide balanced read and write performance (e.g., B-trees) or generate SSD-friendly sequential writes (e.g., LSM trees), as well as good scalability. To maintain specific tree structures, their write operations require adjustments to the data structures (e.g., rotating, splitting, and merging nodes in B-trees) or merge-sort (compaction) of large log structures (e.g., LSM trees). As a comparison, TieredHM adopts a hierarchical structure with inter-layered data movement to render write skewness among layers while also maintaining efficient point queries within each layer. The write operations in TieredHM avoid structural changes and merge-sort of large log structures by restricting a key to be inserted into limited candidate buckets indexed with hash functions, which are more lightweight.