# Locality-Aware Data Placement for NUMA Architectures: Data Decoupling and Asynchronous Replication

Shuhan Bai, Haowen Luo, Burong Dong, Jian Zhou*, Fei Wu*
*Huazhong University of Science and Technology*

***Abstract*—Non-Uniform Memory Access (NUMA) architectures bring new opportunities and challenges to bridge the gap between computing power and memory performance. Their complex memory hierarchies feature non-uniform access performance, known as NUMA locality, indicating data placement and access without NUMA-awareness significantly impact performance. Existing NUMA-aware solutions often prioritize fast local access but at the cost of heavy replication overhead, suffering a read-write performance tradeoff and limited scalability. To overcome these limitations, this paper presents** Ladapa**, a scalable and high-performance locality-aware data placement strategy. The key insight is decoupling data into metadata and data layers, allowing independent management with adaptive asynchronous replication for lower overhead. Additionally,** Ladapa **employs multi-level metadata management leveraging fast caches for efficient data location, further boosting performance. Experimental results show that** Ladapa **outperforms typical replication techniques by up to 27.37× in write performance and 1.63× in read performance.**

***Index Terms*—NUMA, memory, cache, data placement, asynchronous, multi-core systems**

## I. Introduction

Contemporary applications are marked by the explosive growth of data scales and escalating data manipulation needs. These demands have outpaced the modest enhancements in core processing performance, driving the development of large multi-core systems [3]. Non-Uniform Memory Access (NUMA) architectures, equipped with numerous processor cores and large-volume memory, are necessities for providing massive bandwidth and storage capacity along with enormous computational power. These architectures embed complex memory hierarchies, spanning from registers to private and shared caches, local main memory, and remote memory accessed via interconnects [1], [20], [24]. However, non-uniform access performance, known as NUMA locality, arises from the varying types of memory and the distance between processors and memory modules on NUMA architectures, leading to severe performance degradation. Worse, as core counts increase, the side effect brought by asymmetric local and remote memory access becomes more pronounced [19]. Consequently, managing data placement and access patterns for NUMA architectures requires careful locality awareness to reduce cross-node communication and scale performance effectively in large multi-core environments [2].

Many studies focus on overcoming the potential negative impact of NUMA locality to provide basic NUMA support [4], [10], [11], [14]–[16]. Techniques centered on replication and migration are commonly employed to reduce cross-node communication, though they face challenges related to replica maintenance and consistency assurance. Current NUMA-aware strategies involve maintaining complete or partial replicas across nodes, coupled with synchronization through compact logs. On one hand, complete replication techniques [5], [7], [22], while effective at reducing remote memory access and improving read performance, incur significant overhead from maintaining replicas, which impairs write performance and limits scalability in the face of growing core counts and data scales. On the other hand, partial replication techniques [8], [9], [18], [21], [23], which prioritize replicating frequently accessed or critical data, impose less burden on write operations. However, they struggle to adapt to the diverse characteristics of workloads and may suffer from performance degradation due to unavoidable remote accesses during read operations.

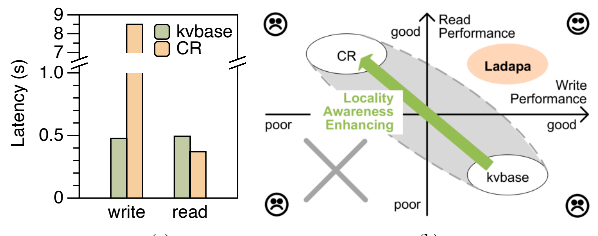


Fig. 1. Quantitative and qualitative evaluations of read and write performance for CR and kvbase. CR propagates complete replicas across nodes; kvbase maintains zero replicas.

Our experimental findings support the preceding analysis. Fig. 1 provides quantitative and qualitative evaluations of read and write performance for CR, a typical complete replication technique, and kvbase, a partial replication technique in the extreme scenario of zero replicas. It is regrettable that replication-based techniques present a trade-off between read and write performance, complicating the resolution of NUMA locality challenges. Enhancing locality awareness, characterized by increased replicas, facilitates read performance due to the prevalence of local access. However, this advantage comes at the cost of worsening write performance due to the escalating replication overhead. This dynamic shifts overall performance into an indeterminate zone, as depicted by the gray area with green arrow in Fig. 1(b). Thus, achieving optimal locality awareness for both good read and write performance is nontrivial, as it necessitates reconciling often competing factors and goals, such as weighing the potential benefits of local replicas against the sharp increase in replication overhead due to scalability, as well as making near-optimal replication decisions in runtime amidst changing workload conditions

*Corresponding authors: jianzhou@hust.edu.cn, wufei@mail.hust.edu.cn to adapt to versatility. Therefore, this paper delves into a fundamental question—**how to design an efficient locality-aware data placement strategy that surpasses read-write performance trade-off and delivers high performance in terms of scalability and adaptability?**

In this paper, we propose Ladapa, a robust locality-aware data placement strategy that achieves high performance and scalability across diverse workloads on NUMA architectures. Recognizing the distinct replication behaviors of data and metadata, the key idea behind Ladapa is to separate the data plane into two components, a metadata layer and a data layer, and individually optimize the way data is allocated and accessed according to specific replication characteristics of each layer. This decoupling enables adaptive replication approaches to effectively harness NUMA locality. In the data layer, asynchronous replication offloads the replication process from the critical path, mitigating the negative impact on write performance and ensuring scalability. Meanwhile, read performance is enhanced through increased local access, thereby overcoming the trade-offs between read and write performance. In the metadata layer, Ladapa adaptively selects between synchronous and asynchronous replication approaches, depending on workload features and system conditions. This flexibility allows the system to respond dynamically to changing workload demands with versatility. Additionally, Ladapa employs a multi-level metadata organization, promoting frequently accessed metadata entries to faster shared caches, further improving data lookup efficiency and gaining additional performance benefits.

The contribution of this paper are as follows:

- We propose Ladapa a new design paradigm of locality-aware data placement that provides high performance with scalability and versatility by data decoupling and asynchronous replication.
- We pioneer integrating shared caches into multi-level metadata management, combined with asynchronous replication, to eliminate replication overhead and accelerate key operations, effectively overcoming the read-write performance trade-off while ensuring accuracy.
- We evaluate Ladapa against typical replication methods, demonstrating up to 27.37× improvement in write performance and 1.63× in read performance.

## II. Background and Motivation

### A. NUMA Architectures with Non-Uniform Performance

One prevalent trend in memory system design is NUMA architectures, which accommodate a multitude of processor cores and manage large-scale memory clusters with good concurrency and scalability, addressing the escalating computation and storage demands to enable efficient manipulation of vast datasets in contemporary computing environments. Fig. 2(a) shows an example NUMA system, where cores are grouped into nodes. Each Node directly connects to its local memory with an on-chip memory controller, offering high-bandwidth and low-latency local memory access. Inter-node communication is facilitated by a cross-chip interconnect, which introduces additional transfer overhead, thereby rendering remote memory

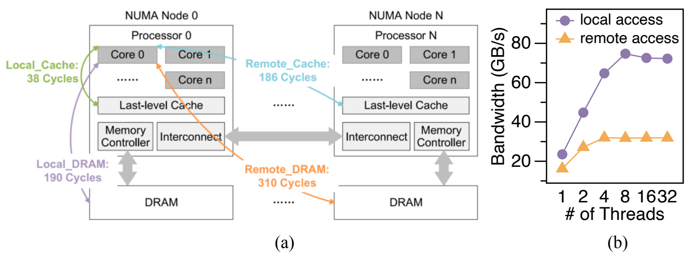


Fig. 2. (a) NUMA architecture: complex memory hierarchies embedded with non-uniform access performance. (b) Bandwidth with varying threads: NUMA locality issues become more significant with concurrency.

access relatively more costly. Typically, cores within a node share a last-level cache. Cache access exhibits superior transferring characteristics; specifically, the latency for remote cache access is comparable to that of local memory access, whereas local cache access is notably more efficient [12], [17].

Moreover, non-uniform access performance featured in NUMA architectures, commonly referred to as **NUMA locality**, intensifies under conditions of concurrent access, as depicted in Fig. 2(b). The disparity in memory access bandwidth between local and remote accesses widens from 30.9% to 57.5% with an increasing number of threads. Worse, the bandwidth resource becomes constrained due to contention when the concurrency exceeds 8 threads. Consequently, the way data is allocated and accessed on NUMA architectures substantially influence performance and should be considered with locality awareness.

### B. Performance Optimization Challenges and Opportunities

To better understand performance optimization potential in NUMA architectures through locality awareness, we conducted a motivational study to uncover previously unexplored challenges and opportunities. The test data were obtained from our experimental platform described in §IV.

Much work on NUMA memory management incorporates locality awareness to enhance performance through data replication [5], [7]–[9], [18], [22]. While creating replicas across nodes effectively mitigates expensive remote access, thereby improving read performance, it concurrently degrades write performance due to the high coping costs. This performance trade-off when reading or writing data complicates the efficient utilization of NUMA locality, as illustrated in Fig. 1(b). We quantify the performance trade-offs under the assumption of typical scenarios: a benchmark comprising $M$ write operations and $qM (q \geq 0)$ read operations, with $pqM (0 \leq p \leq 1)$ of the read operations accessing data resident in local memory, is executed across a NUMA architecture consisting of $N$ nodes. The execution time without data replication is calculated in

$$\begin{aligned} T_{w/o} &= \underline{T_{wr}} + T_{rd} = \underline{Mt_{local}} + pqMt_{local} + (1-p)qMt_{remote} \\ &= M(t_{local} + qt_{remote}) + pqM(t_{local} - t_{remote}), \end{aligned} \tag{1}$$

where $T_{wr}$ and $T_{rd}$ denote the execution times for write and read operations, and $t_{local}$ and $t_{remote}$ represent the access times to local and remote memory ($t_{local} < t_{remote}$). It is obvious that more read operations accessing local memory (a larger $p$) allow for a reduced execution time, underscoring the significance of incorporating NUMA locality in the optimization design. When replicating data at a proportion $x(0 < x \leq 1)$, the ideal minimum execution time is

$$\begin{aligned} T = \underline{T_{wr}} + T_{rd} = \underline{Mt_{local} + xM(N-1)t_{remote}} \\ + min\{qM, xM + pqM\}t_{local} \\ + max\{0, qM - (xM + pqM)\}t_{remote}. \end{aligned} \quad (2)$$

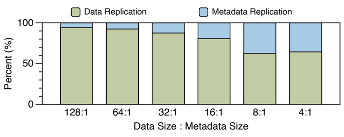


Fig. 3. Decomposition of replication overhead with data decoupling under varying data-to-metadata size ratios.

Compared to Eq. (1), extra replication overhead slows down the write performance, while read performance benefits from more local memory access. Blindly exploiting NUMA locality in data replication strategies is suboptimal; instead, the execution may be faster at any trade-off ($x$), yet determining this dynamic balance is inherently challenging due to its dependency on various access patterns ($q$) and ever-changing system status ($p$).

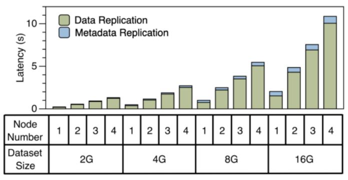


Fig. 4. Replication overhead with varying node numbers and dataset sizes.

Given that replication overhead dominates write performance degradation when regarding NUMA locality, we decompose its components by decoupling metadata from data to explore optimization opportunities. Fig. 3 shows the respective contributions of data and metadata to replication overhead across different data-to-metadata size ratios that encompass various workloads. Data constitutes a larger percentage of replication overhead relative to metadata. Additionally, in workloads where data and metadata sizes are comparable, the overhead associated with metadata replication becomes substantial, escalating to 37.4%. In response to the trend of NUMA architectures in multicore scalability and data scalability, replication overhead is further analyzed with varying node numbers and dataset magnitudes, as shown in Fig. 4. The results reveal a consistent increment in replication overhead with increasing core counts and dataset sizes, indicating a more serious trade-off.

The aforementioned findings elucidate the distinct replication behaviors of data and metadata when adopting NUMA locality and highlight the escalating performance trade-off dilemma associated with scalability. These insights guide targeted optimization paradigms that leverage decoupled data with different characteristics. Furthermore, acknowledging that metadata access is emerging as a considerable source of runtime overhead [13], it is necessary to exploit the superior cache access capabilities inherent in NUMA architectures, which is unexplored.

## III. Design

### A. Design Goals

Ladapa has three main design goals to be a generic locality-aware data placement for NUMA architectures:

- **NUMA Locality:** Ladapa should facilitate memory access patterns that are cache efficient and mostly NUMA-local to avoid costly remote memory access. Data replication and migration are considered effective options.
- **Scalability:** The performance of Ladapa should scale or exhibit stability with increasing core numbers and data scales, aligning with the predominant development trend of modern high-performance servers.
- **Versatility:** Ladapa should be performant across workloads with diverse data access and organization patterns.

Ladapa accomplishes these goals through **data decoupling** and **adaptive asynchronous replication**, capitalizing on the distinct replication characteristics observed between data and metadata when incorporating NUMA locality. Furthermore, **faster cache access** are leveraged to efficiently locate data, thereby reaping further performance advantages.

### B. Data Plane: Decoupling

The evolution of memory systems profoundly affected the design paradigms of efficient locality-aware data placement for NUMA architectures. Increased memory capacity and core counts exacerbate the negative performance impact associated with replication, which is commonly employed to enhance NUMA locality. For large-scale storage data, the heavy replication overhead may potentially outweigh the performance gains achieved by mitigating remote memory access. While metadata typically has a smaller footprint than the data itself, making its replication less costly and allowing efficient data location via metadata replicas, this advantage diminishes in cases where metadata size approaches that of the data or when dealing with extremely large datasets. In such scenarios, the overhead from metadata replication becomes substantial and cannot be overlooked. In light of the observation that metadata and data exert distinct degrees of influence on performance when accounting for NUMA locality, Ladapa introduces a stratified approach by **decoupling** the data plane into two layers: a metadata layer and a data layer, as shown in Fig. 5. This separation enables more efficient management of data placement and access patterns, optimizing them according to specific replication behaviors and characteristics of each layer.

As memory capacities expand, accommodating a larger dataset leads to an increased scale of keys, thereby extending the search path required to locate data. Empirical studies have indicated that metadata access assumes a significant proportion of runtime overhead, ranging from 14-94% [13]. Consequently, facilitating efficient metadata access is imperative. Drawing inspiration from the rapid access times of local and remote caches, Ladapa integrates the last-level cache of each node to form a globally accessible *First-Level Metadata Table*. This design leverages the intrinsic cache strategies and cache consistency protocols to make frequently accessed metadata entries visible to all cores from a global view. Concurrently, per-node *Second-Level Metadata Tables* retain less frequently accessed entries, which are either flushed or replaced from the first-level table, thereby aiding in the effective data retrieval. This **tiered metadata management** approach capitalizes on the low latency of system caches and the access patterns of workloads, accelerating metadata lookup operations and significantly gain benefits on both read and write performance. Note that Ladapa optimizes data placement within the data plane and adapts to various index structures like hash tables, B-Trees, skiplists, etc.

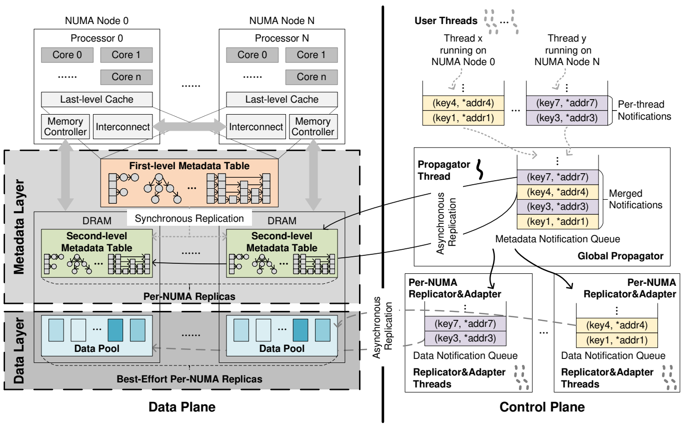


Fig. 5. Ladapa overview: data plane and control plane.

## C. Control Plane: Replication

Strategic optimization of replication, customized to the distinct behaviors and characteristics of each layer, plays a critical role in enhancing system performance. Ladapa employs asynchronous replication across nodes for large-scale storage data; and selectively applies replication strategies—either asynchronous or synchronous—for metadata, depending on the specific features of each application scenario.

The control plane in Fig. 5 illustrates adaptive asynchronous replication in Ladapa which uses three types of background threads and three software components: *Global Propagator*, *per-NUMA Replicator* and *per-NUMA Adapter*. When requests arrive, user threads independently record relevant metadata entries and assess workload characteristics to determine whether to apply synchronous or asynchronous replication of metadata. In cases other than those involving small-scale metadata workloads, when metadata entries are evicted from the first-level metadata table due to space limitations, the metadata is actively and synchronously replicated across all nodes. This ensures that subsequent read and write operations can accurately locate the data via the multi-level metadata tables. For asynchronous replication, metadata entries are sequentially enqueued as metadata replication notifications within the global propagator's metadata notification queue. A background thread, named propagator thread, monitors this queue, ensuring the timely propagation of pending metadata to the second-level metadata tables across all nodes. The thread subsequently transforms the metadata entries into data replication notifications, which are then enqueued into the data notification queue of each node's per-NUMA Replicator. Its replicator thread locates data based on the metadata to asynchronously replicate its across nodes. Concurrently, each node's per-NUMA adapter operates an adapter thread that dynamically identifies workload characteristics during runtime. This thread is crucial in adaptively accelerating asynchronous data replication, further boosting a smart trade-off between read and write performance while incorporating NUMA locality.

This **adaptive asynchronous replication** approach is essential for effectively incorporating NUMA locality while considering scalability and versatility. Replication operations are offloaded from the critical path in case their overhead threatens performance, ensuring robust write performance while reducing remote access through best-effort per-NUMA data replicas. Besides, the fast propagation of metadata is adaptively managed to create per-NUMA metadata replicas, further enhancing both read and write performance while maintaining correctness.

One problem associated with maintaining multiple replicas is the increased consumption of storage space. Since the first-level metadata table stores frequently accessed entries, Ladapa addresses the issue of limited node memory by identifying and evicting cold data through the least frequently accessed metadata entry selected from the second-level metadata table, thereby reclaiming memory space and optimizing utilization efficiency. This issue lies beyond the primary focus of Ladapa and will not be elaborated here.

## D. Ladapa Walkthrough

Fig. 6 depicts the workflow of write and read operations in Ladapa, which are illustrated respectively below.

*1) Write Operations:* Write operations in Ladapa are divided into three stages: writing or updating data and metadata, fast replicating metadata, and asynchronous replicating data. Details of each stage are presented following.

**Writing or Updating data and metadata.** Upon receiving a write request from the upper layer, the user thread first tries to quickly locate the data by checking the *First-Level Metadata Table* residing in the shared cache (①). If fails, the thread advances to the *Second-Level Metadata Table* of the local node (②). Once the data is located, a direct update is applied to the corresponding data within the *Data Pool*. If the data is not present, the thread allocates the required space in the data pool and writes the data (③). Subsequently, the relevant metadata is updated or inserted into the first-level metadata table (④), taking advantage of the fast cache access. In scenarios featuring workloads with small-scale metadata, the overhead of metadata replication is negligible, allowing a synchronous replication strategy to be sufficient. In such cases, the metadata is promptly replicated to the second-level metadata tables across all nodes (⑤). However, for larger datasets,

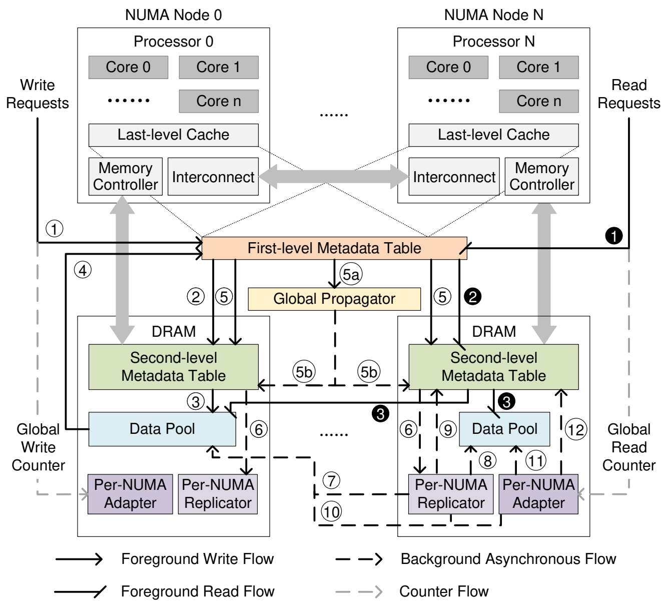


Fig. 6. Write and read operations in Ladapa.

the thread generates a metadata replication notification and enqueues it in the *Global Propagator*'s metadata notification queue (⑤a), facilitating efficient metadata propagation across nodes via an asynchronous background thread.

**Fast replicating metadata.** An exclusive background asynchronous propagator thread, specialized in metadata replication, polls the metadata notification queue managed by the global propagator during runtime. Upon encountering queued notifications, the thread sequentially extracts metadata entries and replicates them to the second-level metadata table of each node (⑤b). Then, the thread crafts corresponding data replication notifications based on the metadata and places them into the data notification queue of each node's *per-NUMA Replicator* (⑥), ensuring the consistent maintenance of data replicas.

**Asynchronous replicating data.** Within each node, an exclusive background asynchronous replicator thread, specialized in data replication, actively listens to the data notification queue of the local per-NUMA replicator in runtime. When the queue contains entries, the thread sequentially retrieves each notification, decodes the embedded metadata to locate the data, and obtain relevant data from the corresponding node (⑦). Thereafter, the data is copied to the local data pool (⑧), and the corresponding metadata entry in the local node's second-level metadata table is updated to redirect the data replica, with the memory space pointed by the obsolete address being reclaimed (⑨). Note that asynchronous data replication operates on a best-effort basis to mitigate remote memory access. To further enhance read performance, each node's *per-NUMA Adapter* persistently monitors the *Global Read/Write Counters* and profiles the workload features. For read-intensive workloads, a dedicated adapter thread collaborates with the replicator thread in extracting notification entries from the local per-NUMA replicator's data notification queue, thereby accelerating the replication of data to local memory (⑩, ⑪, ⑫).

*2) Read Operations:* Read operations in Ladapa are simply performed in two stages: metadata lookup and data retrieval. Upon receiving a read request from the upper layer, the user thread initiates a multi-level lookup to identify the relevant metadata entry (❶, ❷). Based on the metadata, the thread then locates and reads the requested data (❸), which may reside in either local or remote memory, depending on the efficiency of the asynchronous data replication.

## IV. Evaluation

### A. Experimental Setup

**Testbed.** Experiments are conducted on a 4-socket (NUMA node) machine. Each node is populated with a 36-core Intel(R) Xeon(R) Gold 6254 CPU@3.10GHz, eight 32GB DDR4 DIMMs, resulting in a machine with 144 cores and 1024GB DRAM. Moreover, 24MB L3 caches are equipped. Our machine runs Ubuntu 20.04 with Linux kernel version 5.4.0.

**Workloads.** We leverage the Yahoo! Cloud Serving Benchmark (YCSB) [6], a benchmark for key-value stores widely recognized in the industry, to evaluate the performance of Ladapa. The benchmark contains five types of workloads: 1) write-only: 100% update, 2) read-only: 100% read, 3) write-intensive: 95% update and 5% read, 4) read-intensive: 95% read and 5% update, 5)write-read balance: 50% read and 50% update. The default configuration specifies a key space of 200 million (i.e., the range of keys), with key popularity adhering to a Zipfian distribution with parameter 0.99, consistent with YCSB's default settings. In each experiment, we first load 4.19 million items and then execute the workloads with a substantial dataset of 16GB, which contains about 4.19 million operations. We utilize 32-byte metadata (key-value pair) to index 4096-byte data, emulating scenarios featured by large-scale datasets and a small ratio of metadata to data. Our experiments, under diverse workload configurations, could provide a comprehensive evaluation of Ladapa scalability and versatility.

**Baselines.** We compare Ladapa with a basic technique without replica, as a variant of partial replication techniques and a typical complete replication technique:

- *kvbase*: no replicas among nodes.
- *CR* [22]: synchronous complete replication.
- Ladapa: asynchronous replication; multi-level metadata table across cache and DRAM.

We implemented Ladapa in C++, with about 1,200 lines of code. The first-level metadata table is configured with a size of 32MB, and its memory is allocated in one certain NUMA node, declared as a global variable accessible by all threads. Ladapa leverages `moodycamel::ConcurrentQueue` to construct notification queues for replication of metadata and data across nodes. Since Ladapa is independent of the underlying index structure, we uniformly employed a classic high-performance hash table, `google::dense_hash_map`, as the indexing mechanism for all comparative baselines.

### B. Write Performance Evaluation

Fig. 7 shows the comparison in terms of throughput under write-only workload with uniform and Zipfian distribution. Under uniform distribution, Ladapa benefits from the adaptive asynchronous replication mechanism. By decoupling the time-consuming replication process from the critical path, it achieves a significant performance improvement of 10.23-25.96× with the increase in thread count, surpassing the synchronous replication method employed by CR. Additionally, the first-level metadata table in shared caches greatly shortens the search path of metadata entries, facilitating rapid data localization. Therefore, Ladapa realizes a 1.03-1.20× performance gain over kvbase. Similarly, under Zipfian distribution, Ladapa improves write performance by 11.41-27.37× and 1.20-1.60× relative to CR and kvbase, respectively, as thread increases. The enhanced performance improvement under Zipfian distri-

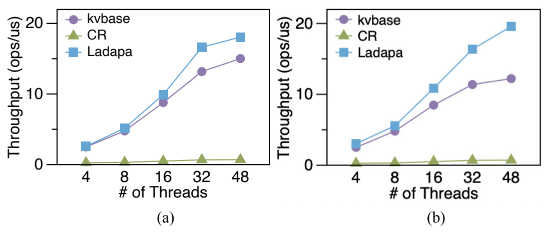
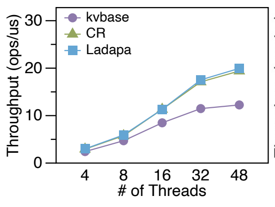


Fig. 7. Throughput under write-only workload with (a) uniform distribution and (b) Zipfian distribution.


(a)

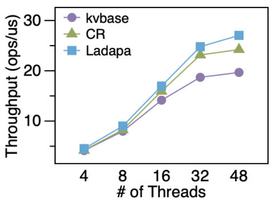

(b)

Fig. 8. Throughput under read-only workload with (a) uniform distribution and (b) Zipfian distribution.

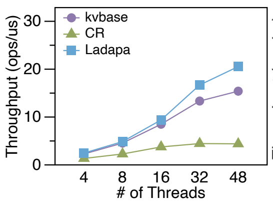
bution, as opposed to a uniform one, can be attributed to the workload's inherent locality, which affords greater optimization potential for the first-level metadata table. This locality-driven optimization using faster cache access is pivotal for Ladapa's superior performance under skewed data access patterns.

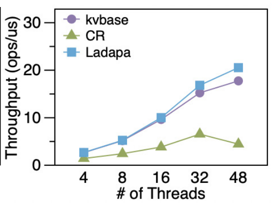
## C. Read Performance Evaluation

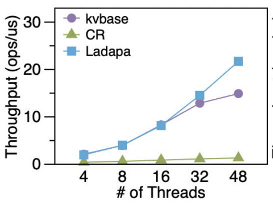
Fig. 8 shows the comparison in terms of throughput under read-only workload with uniform and Zipfian distribution. Regardless of the distribution, both Ladapa and CR are capable of retrieving most or all of the required data from local memory owing to their data replication techniques, resulting in performance advantages over kvbase, which does not account for NUMA locality. Moreover, Ladapa further enhances metadata lookup efficiency by leveraging its first-level metadata table, thereby achieving superior read performance with a speedup of up to 1.63× compared to kvbase and 1.12× compared to CR. Besides, the inherent workload locality characteristic of the Zipfian distribution facilitates rapid data retrieval. Relative to the uniform distribution, all three approaches demonstrate increased throughput.

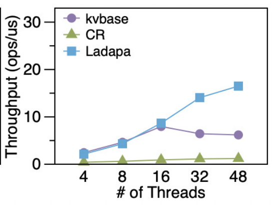
## D. Mixed Workload Performance Evaluation

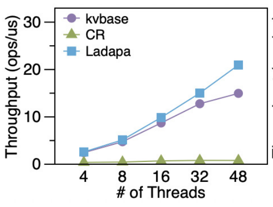
Experimental results of CR and kvbase under read-only and write-only workloads demonstrate the trade-off between read and write performance in NUMA-aware techniques. In contrast, Ladapa's optimal performance both in read and write operations disrupts this trade-off, thereby validating its efficacy. Further analysis of Ladapa's comprehensive performance under mixed workloads is presented in Fig. 9. Vertically, as write operations increase, CR's performance declines due to the cost of synchronous replication. Ladapa, however, always benefits from its adaptive asynchronous replication and the fast cache access facilitated by its multi-level metadata management, achieving peak performance improvements of up to 2.67× over kvbase and 26.19× over CR. Horizontally, the performance of all approaches under Zipfian distribution parallels that under a uniform distribution, diverging from the results in read-only scenarios. This disparity arises from the increased lock contention on the index structure when both read and write operations are executed concurrently, potentially neutralizing the performance benefits of workload locality. As thread counts escalate, lock contention's impact intensifies, potentially leading to performance degradation. Nonetheless, Ladapa's independence from the underlying index structure allows it to mitigate this issue through the application of advanced concurrent indexing techniques [5], [18], [22].


(a)


(b)


(c)


(d)


(e)

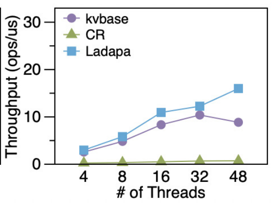

(f)

Fig. 9. Throughput under read-intensive, write-intensive and write-read balance workload with uniform (left column) and Zipfian (right column) distributions.

## E. Sensitivity Analysis

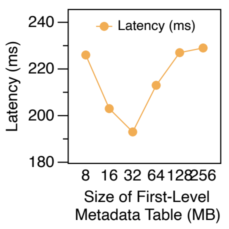


Fig. 10. Sensitivity analysis for the size of the first-level metadata table.

Fig. 10 shows how the size of the first-level metadata table affects performance. Our experiments, conducted under the write-read balance workload with 48 threads, scaled the table size from 8MB to 256MB. Optimal performance is observed with a 32MB table, which can accommodate enough metadata entries without frequent cache replacement. Performance declines with exceeded small/large due to either insufficient cacheable metadata entries or excessive table sizes relative to cache capacity, leading to increased replacement.

## V. Conclusion

This paper introduces Ladapa, a scalable and high-performance locality-aware data placement strategy for NUMA architectures. By decoupling data into metadata and data layers, Ladapa enables independent management with adaptive asynchronous replication to reduce overhead. Its multi-level metadata management utilizes fast caches to further improve data access efficiency. Experiments show Ladapa achieves up to 27.37× better write performance and 1.63× better read performance compared to typical replication methods.

## Acknowledgment

This work was supported in part by the National Key Research and Development Program of China under Grant No.2022YFB4501100, and by the NSFC under Grants No.62372197, No. U22A2071, and No.62102155.

## References

[1] Ardsher Ahmed, Pat Conway, Bill Hughes, and Fred Weber. Amd opteron shared memory mp systems. In *Proceedings of the 14th HotChips Symposium*, 2002.

[2] Sergey Blagodurov, Sergey Zhuravlev, Alexandra Fedorova, and Ali Kamali. A case for numa-aware contention management on multicore systems. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 557–558, 2010.

[3] Geoffrey Blake, Ronald G Dreslinski, and Trevor Mudge. A survey of multicore processors. *IEEE Signal Processing Magazine*, 26(6):26–37, 2009.

[4] Timothy Brecht. On the importance of parallel application placement in numa multiprocessors. In *Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV)*, pages 1–18, 1993.

[5] Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K Aguilera. Black-box concurrent data structures for numa architectures. *ACM SIGPLAN Notices*, 52(4):207–221, 2017.

[6] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[7] Andreia Correia, Pedro Ramalhete, and Pascal Felber. A wait-free universal construction for large objects. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 102–116, 2020.

[8] Lixiao Cui, Kedi Yang, Yusen Li, Gang Wang, and Xiaoguang Liu. Difflex: A high-performance, memory-efficient and numa-aware learned index using differentiated management. In *Proceedings of the 52nd International Conference on Parallel Processing*, pages 62–71, 2023.

[9] Henry Daly, Ahmed Hassan, Michael F Spear, and Roberto Palmieri. Numask: high performance scalable skip list for numa. In *32nd International Symposium on Distributed Computing (DISC 2018)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2018.

[10] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth. Traffic management: a holistic approach to memory placement on numa systems. *ACM SIGPLAN Notices*, 48(4):381–394, 2013.

[11] Fabien Gaud, Baptiste Lepers, Justin Funston, Mohammad Dashti, Alexandra Fedorova, Vivien Quéma, Renaud Lachaize, and Mark Roth. Challenges of memory management on modern numa systems. *Communications of the ACM*, 58(12):59–66, 2015.

[12] Daniel Hackenberg, Daniel Molka, and Wolfgang E Nagel. Comparing cache architectures and coherency protocols on x86-64 multicore smp systems. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on microarchitecture*, pages 413–422, 2009.

[13] Onur Kocberber, Boris Grot, Javier Picorel, Babak Falsafi, Kevin Lim, and Parthasarathy Ranganathan. Meet the walkers: Accelerating index traversals for in-memory databases. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 468–479, 2013.

[14] Renaud Lachaize, Baptiste Lepers, and Vivien Quéma. {MemProf}: A memory {Profiler} for {NUMA} multicore systems. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 53–64, 2012.

[15] Baptiste Lepers, Vivien Quéma, and Alexandra Fedorova. Thread and memory placement on {NUMA} systems: Asymmetry matters. In *2015 USENIX annual technical conference (USENIX ATC 15)*, pages 277–289, 2015.

[16] Zoltan Majo and Thomas R Gross. Memory management in numa multicore systems: trapped between cache contention and interconnect overhead. In *Proceedings of the international symposium on Memory management*, pages 11–20, 2011.

[17] Zoltan Majo and Thomas R Gross. (mis) understanding the numa memory system performance of multithreaded workloads. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pages 11–22. IEEE, 2013.

[18] Ajit Mathew and Changwoo Min. Hydralist: A scalable in-memory index using asynchronous updates and partial replication. *Proceedings of the VLDB Endowment*, 13(9):1332–1345, 2020.

[19] Asymmetry Matters. Thread and memory placement on numa systems. *Operating Systems and Sysadmin*, page 15.

[20] Ronak Singhal. Inside intel next generation nehalem microarchitecture. In *Hot Chips*, volume 20, page 15, 2008.

[21] Qing Wang, Youyou Lu, Junru Li, and Jiwu Shu. Nap: A {Black-Box} approach to {NUMA-Aware} persistent memory indexes. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 93–111, 2021.

[22] Zhengming Yi, Yiping Yao, and Kai Chen. A universal construction to implement concurrent data structure for numa-muticore. In *Proceedings of the 50th International Conference on Parallel Processing*, pages 1–11, 2021.

[23] Junsung Yook and Bernhard Egger. Selective data migration between locality groups in numa systems. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 143–147. Springer, 2022.

[24] Dimitrios Ziakas, Allen Baum, Robert A Maddox, and Robert J Safranek. Intel® quickpath interconnect architectural features supporting scalable system architectures. In *2010 18th IEEE Symposium on High Performance Interconnects*, pages 1–6. IEEE, 2010.