DiLOS: Adding Performance to Paging-based Memory Disaggregation

Wonsup Yoon
KAIST
wsyoon@kaist.ac.kr

Jinyoung Oh
KAIST
jinyoungoh@kaist.ac.kr

Jisu Ok
KAIST
jisu.ok@kaist.ac.kr

Sue Moon
KAIST
sbmoon@kaist.edu

Youngjin Kwon
KAIST
yjkwon@kaist.ac.kr

ABSTRACT

Memory disaggregation places computing and memory in physically separate nodes and achieves improved memory utilization in datacenters. Kernel-based approaches for memory disaggregation offer transparent virtual memory by using paging schemes but suffer from expensive page fault handling. As an alternative, library-based approaches incorporate application semantics to memory disaggregation and can even eliminate page fault handling on its data path. However, its lack of compatibility harms generality and obstruct wide adoption.

This paper revisits the paging-based approaches and challenges their performance. We posit that the page fault overhead is not a fundamental limitation. We propose DiLOS, a new memory disaggregating unikernel, that delivers both performance and generality. The key insight of DiLOS to overcome performance drawbacks while maintaining generality lies in the design of a fast, lightweight page fault handler on top of the unikernel's simple execution model. Since each unikernel serves a single application, it also opens room for extra optimization via app-aware prefetching. DiLOS outperforms a recent library-based system (AIFM) by 1.52× and 1.31× when the cache size is 12.5% and 100% of the total working set, respectively. Compared to the state-of-the-art paging-based system (Fastswap), DiLOS with a general-purpose prefetcher achieves up to 2.2× higher performance in real-world workload. An app-aware prefetcher further improves the throughput of Redis in-memory database up to 27%.

APSys '21, August 24–25, 2021, Hong Kong, China

ACM ISBN 978-1-4503-8698-2/21/08…$15.00
https://doi.org/10.1145/3476886.3477507

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**.

KEYWORDS

memory disaggregation, disaggregated data center, unikernel

ACM Reference Format:
Wonsup Yoon, Jinyoung Oh, Jisu Ok, Sue Moon, and Youngjin Kwon. 2021. DiLOS: Adding Performance to Paging-based Memory Disaggregation. In *12th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '21), August 24–25, 2021, Hong Kong, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3476886.3477507

1 INTRODUCTION

Resource disaggregation is a new hardware and system paradigm to split computation, memory, and storage into individual resource pools. Compared with the traditional single-machine design, resource disaggregation allows independent scaling and flexible resource provisioning beyond the boundary of a single machine. The benefits are a key driver to solve the chronic resource under-utilization problem in datacenters [13, 43, 46]. Cloud service providers are building disaggregated datacenters (DDC) to take full advantage of resource disaggregation [25].

Memory disaggregation aims to split computing and memory in the DDC, placing computing and memory in physically separate nodes. A computing node has a large number of computational units, while a memory node provides a large amount of memory with few or no computational units. Memory disaggregation enables building a large memory pool shared across compute nodes and overcomes the memory wall in the cluster of traditional server nodes [2, 12, 32, 43]. Fast networking technologies such as RDMA and userspace networking have reduced the remote access latency [45] and accelerated the trend towards memory disaggregation.

Early efforts to adopt memory disaggregation have mostly built it as a kernel feature [2, 12, 25, 32, 43]. These kernel-based approaches recast the existing kernels to migrate pages between the computing node and the memory node. Computing nodes evict pages to memory nodes under memory pressure, unmap them in their page table, and fetch them back on page faults. Since paging changes the address mapping behind the scene, applications use disaggregated memory without modification.

The compatibility of the kernel-based approaches, however, comes with a cost. Frequent kernel-user switching incurs non-negligible performance overhead, and semantic gaps between pages and actual units of access remain as opportunities for performance improvement. To overcome these limitations, library-based approaches incorporate application semantics to memory disaggregation. AIFM [40] puts user-level libraries in charge of remote memory management to avoid page faults. Semeru [47] reduces the number of page faults and network bandwidths by offloading garbage collection to its memory node. Yet library-based approaches trade compatibility for performance. AIFM's programming model requires annotations for remote objects and mandates custom C++ API. Semeru only supports JVM-related programming languages.

We look back at the burden of the kernel-based model: Are the costly switching overheads and lack of application knowledge fundamental limitations of paging? We rehash existing attempts to relax the mode switching overhead and the semantic gap, built on top of kernel abstraction. Dune [7] runs user processes in non-root ring-0 mode while the underlying OS runs in the root mode, allowing the user processes to manage paging directly. Unikernels [28, 29] break the boundary between a kernel and a process and integrate them together so that all software stack runs in ring-0 mode. Both approaches eliminate the kernel-user mode switching in paging, for processes themselves and page fault handling all run in ring-0 mode. Nevertheless, unikernels have a narrower attack surface and provide more robust isolation than Dune-like approaches, thus more suitable for datacenter environments. Moreover, unikernels open doors for additional improvements by leveraging application semantics; tailoring a kernel to an application includes optimization on paging techniques such as app-aware prefetching. In this work, we propose a unikernel-based memory disaggregation system that offers both performance and compatibility: DiLOS (Disaggregated Library Operating System). It is paging-based, but its page fault handler is void of mode-switching overhead. It incorporates known techniques for performance optimization, such as prefetching, background write-back, and fast RDMA communication. Our system supports POSIX and compatibility follows. In addition, DiLOS exploits application semantics in the form of a prefetching guide. The prefetchers take the guide as a cue for further performance improvement over general-purpose prefetchers.

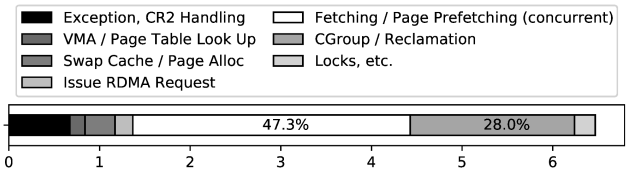

Figure 1: The x-axis is in μs. The performance breakdown shows page fetching takes up 47.3% of latency.

To evaluate our claim, we implement DiLOS on an open-source unikernel, OSv [17], and compare it with the library-based system, AIFM, and the kernel-based system, Fastswap. We run the same workload from AIFM and compare the performance. When a compute node has 12.5% and 100% of the total working set, DiLOS performs 52% and 31% better than AIFM, respectively, even without modifying applications. Against Fastswap, DiLOS demonstrates up to 2.2× superior performance in real-world workload. In addition, DiLOS with app-aware prefetcher achieves up to 27% higher throughput in Redis than DiLOS with a general-purpose prefetcher proposed in recent work [32]. These results demonstrate that with the simple abstraction and careful engineering on a lightweight platform, DiLOS offers a competitive and viable solution for memory disaggregation.

2 MOTIVATION

Existing kernel-based systems take a *reactive* approach in remote memory access: when a page not in local memory raises an explicit page fault, the OS (or hypervisor) fetches the page in response to the page fault. On the contrary, AIFM [40] takes a *proactive* approach: it checks whether an object is in local or remote memory when dereferencing its pointer. This approach eliminates page faults in the data path and promises good performance. However, AIFM forces programmers to use their APIs and annotate objects. To justify their design choices, AIFM points out that the kernel-based model has two fundamental drawbacks: cost of page faults and lack of application semantics.

Cost of page faults. To understand the cost in kernel-based systems, we conduct a performance breakdown (Figure 1). The latency coming from hardware, such as raising and returning page fault exceptions, takes up only 10.2% (0.67 μseconds) out of total page fault handling latency. The dominant fraction of latency comes from software cost originating from Linux: large room for performance improvement.

Unikernels assume a single process environment with a simple physical memory layout. Compared to Linux, unikernels do not need a complicated virtual address and physical memory management, inter-process resource controls, and namespace-based confinement. All page faults occur within a process, and the page fault handler does not require inter-process security checks and locks. These optimizations reduce the handling latency close to the physical hardware limit.

Lack of application semantics. AIFM uses intrusive semantic hints to optimize policies for fetching and evicting remote pages, such as fine-grained hotness tracking, prefetching, and detecting non-temporal accesses. Though unikernels are not as rich in features as the intrusive approach, zero cost in application-kernel switching and the highly customizable kernel (library OS) allows applications to leverage their domain knowledge in customizing prefetchers. Our DiLOS provides APIs to customize a prefetcher.

This work aims to demonstrate that it is feasible to have very low-cost page faults and at the same time utilize application semantics without any application modification.

3 DESIGN AND KEY COMPONENTS

DiLOS is a paging-based memory disaggregation system built on a unikernel. DiLOS provides fast remote memory access without modifying existing applications. In this section, we present the design and key components of DiLOS. We first describe the design overview of DiLOS (§3.1) and elaborate on key components: the page fault handler (§3.2), the prefetcher (§3.3), the page manager (§3.4), the communication module (§3.5), and the memory server (§3.6). Lastly, we discuss app-aware prefetching (§3.7) and compatibility of DiLOS (§3.8).

3.1 Design Overview

DiLOS consists of the four key components (fast page fault handler, prefetcher, page manager, and communication module) running on a computing node and a memory server on a memory node. On a computing node, local physical memory works as cache for remote memory similar to many modern disaggregated systems [2, 12, 25, 32, 43]. In the memory node, a memory server process reserves memory and handles memory requests from computing nodes. Figure 2 shows the system overview of DiLOS's computing node. The kernel is a library operating system containing the application in the same address space. The application interacts with the kernel over POSIX system calls. Thus DiLOS supports conventional application binaries compiled from any language. An application contains an app-aware prefetching guide. The guide uses the application's domain knowledge to provide semantic hints for prefetching. When a customized guide is absent, DiLOS uses a recently proposed general-purpose prefetcher [32]. The guide is in the form of a third-party library, which does not require any modification to the application's main code.

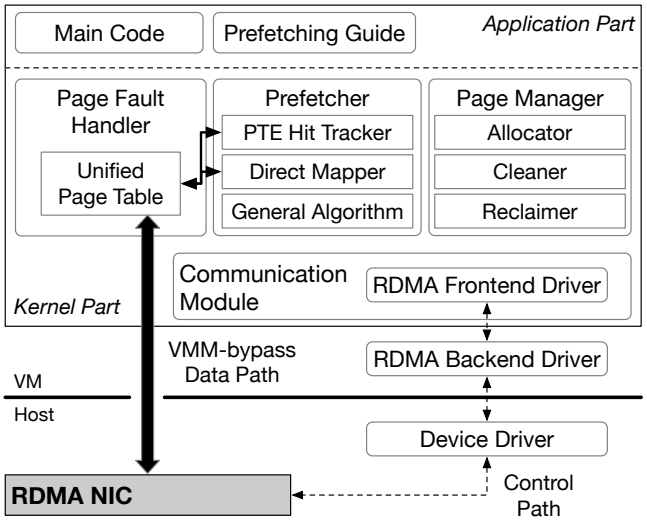

Figure 2: DiLOS system overview

3.2 Fast Page Fault Handler

The key idea behind reducing the page fault latency is to overlap page fault handling and asynchronous network requests. We move the expensive parts in page fault handling, such as permission checks and page reclaims, after the network request so that they are handled while the network request is being served.

We combine the local and remote page tables to a *unified page table* and store all the information (*e.g.*, remote address) needed to fetch pages from a memory node. It translates an application's virtual address to local physical memory or a remote address according to the data location. For local cache, DiLOS follows the Intel page table structure for efficient memory access. For other cases, DiLOS marks the present bit in the page table entry (PTE) to zero. DiLOS also marks the writable bit for the remote address and the user bit for protection to distinguish an unallocated case. DiLOS uses the remaining bits to embed the remote address and protection information. Later, when the application accesses a virtual address associated with the PTE, a page fault occurs. Then, DiLOS walks the unified page table and reads the embedded information. If the PTE is a remote address entry, DiLOS fetches its page from the memory node. For protected PTEs, which means there is an outstanding fetch request, the page fault handler waits for its completion.

3.3 Prefetcher

Prefetching is a common yet integral mechanism to hide inevitable hardware latency and network roundtrip time. DiLOS's prefetcher reduces overheads in prefetching stack by bypassing the *swap cache*. The swap cache is a core data structure in Linux's swap system. It stores all prefetched pages and provides statistical information, such as the hit ratio and the access history. Modern prefetchers (*e.g.*, readahead [14] and Leap [32]) use the statistics to determine the prefetching window size. However, the swap cache design incurs a large number of minor page faults, limiting overall performance. When a prefetching page arrives, Linux stores it in the swap cache. When the application accesses the page, minor page fault occurs, and the page fault handler maps the page in the swap cache. Also, it tracks the minor page faults to gather statistical information.

DiLOS cuts down on page faults during prefetching via skipping the swap cache. Instead of storing prefetched pages into the swap cache, it directly maps them to the page table. This design reduces the number of page faults and saves the swap cache lookup latency.

However, bypassing the swap cache also skips gathering statistical information. DiLOS has another way to track the statistics: a *hit tracker*. Upon prefetching, the hit tracker directly reads accessed bits in the page table entries and stores accessed addresses in access history. Then, it uses the information to induce hit ratio.

3.4 Page Manager

When local memory has no room to allocate new pages, DiLOS evicts pages in the local DRAM to the memory node. The design goal of DiLOS's page manager is cooperative execution of eviction path and fault handler. It hides its eviction latency to the window of fetching a page. Page eviction happens by two modules in a background thread: a *cleaner* and a *reclaimer*.

When the page fault handler requests pages for local DRAM, the page manager allocates free pages. The page manager inserts the pages into the LRU list and marks them evictable. The cleaner periodically scans the LRU list from its tail. If it finds dirty pages (dirty bits in their page table entries are set), it writes them back (no eviction) to the memory node and clears the dirty bits. When the system is under memory pressure, the reclaimer evicts the least recently cleaned and not-accessed pages first because clean pages are simply discarded. If there exists no cleaned page, the reclaimer evicts pages according to the clock algorithm. We observe that the cooperative execution of the page fault handler, the cleaner, and the reclaimer enables the latency of performing page management to be completely hidden within the time window of fetching a page.

3.5 Communication over RDMA

RDMA is the state-of-the-art communication channel for memory disaggregation. However, to use RDMA in unikernels, we should port an RDMA driver. We have considered PVRDMA [37] and HyV [36], but using them requires a non-trivial effort. Instead, we have built our own RDMA driver borrowing VMM-bypass [27]'s concept: reusing VMM's control path and bypassing VMM in data path.

RDMA has two paths: a control path for managing RDMA resources and a data path for transferring data. DiLOS and other disaggregated systems use the control path only at the initialization stage for establishing connections between computing nodes and memory nodes. Thereby, we conclude that the control path does not have to be fast and reuse an RDMA driver in the hypervisor (Linux). If the RDMA frontend driver receives RDMA control requests, it delivers them to the RDMA backend driver on the host side via virtualized device (virtIO). Then, the backend driver translates addresses in the requests to map an MMIO region. After the mapping, DiLOS issues requests to the RDMA NIC (RNIC) using the MMIO region without the driver's intervention.

3.6 Memory Server

Our memory node runs a memory server as a process. It reserves memory using 1GB huge TLB pages and registers them as RDMA memory regions. Huge pages reduce cache misses in the RDMA NIC; thus, they reduce the number of address translations in the NIC [49].

3.7 App-aware Prefetching Guide

To improve performance in memory disaggregation, unikernels can incorporate an application's semantics to customize the kernel's operations. Traditional OS has non-negligible costs at the time scale of handling page faults and exchanging pages over network. For example, a Linux upcall latency to invoke a user-level handler takes 2-3 μseconds [4, 10], whereas a single page fault takes 3-4 μseconds. In unikernels, on the other hand, the costs of system calls and upcalls are the same as function calls so that an application can specify user-level policies requiring frequent communication between the application and the kernel.

DiLOS provides upcall/downcall interfaces for prefetching guide, and the guide hooks application's domain knowledge to customize its prefetching algorithm via the interfaces. If a page fault happens, the page fault handler issues a fetch request to the communication module. During fetching, the handler issues an event with the faulting address and statistics to the guide. Upon receiving the event, the guide uses application semantic to inform the prefetcher what data to prefetch.

However, the application is agnostic to the notion of the page. From the application's standpoint, data represents a virtual address and a size. To fill the semantic gap, DiLOS supports *subpage* prefetching. Subpage prefetching is used when an app-aware guide quickly needs a small amount of data (not whole page) in a memory node, accelerating prefetch decisions.

Moreover, the prefetching guide is a shared object module, it does not require application modification, and DiLOS links it with the application on runtime if needed.

3.8 Compatibility

To preserve compatibility, DiLOS uses Linux ABI and POSIX-compliant interfaces. Moreover, to support unmodified off-the-shelf application binaries, DiLOS patches the symbol table of the binary to use the DiLOS memory allocator. DiLOS has `ddc_malloc` and a custom ELF loader. The `ddc_malloc` is a drop-in replacement of default memory allocator to let DiLOS identify memory pages that can be evicted to a remote memory server. During loading an application binary, DiLOS ELF loader links `malloc` in PLT and GOT sections to the `ddc_malloc`, making application use disaggregated memory.

4 IMPLEMENTATION

We build DiLOS on top of OSv [17]. DiLOS is written in 5,085 lines of C/C++ code[1]. We also modify 576 lines in OSv to link with DiLOS and 880 lines in QEMU to implement the RDMA backend driver. The memory node consists of 457 lines of code. DiLOS has two general-purpose prefetchers: Linux's VMA-Readahead algorithm [14] and Leap's majority trend-based algorithm [32]. To overcome the semantic gap, DiLOS also provides an app-driven prefetcher for Redis, described in §5.2.

We also optimize RDMA configuration. DiLOS posts requests using WQE-by-MMIO, which reduces latency with the cost of bandwidth [15]. To support the WQE-by-MMIO, we fix OSv to use a write combining buffer.

Limitations. Since DiLOS is based on OSv unikernel, it inherits limitations on the unikernel. Though OSv uses Linux ABI and runs unmodified Linux binaries [18], it lacks supports for applications using multi-process APIs such as `fork()`, `vfork()`, and `clone()`. Like Fastswap and AIFM, the current implementation does not include a fault tolerance mechanism when a memory node fails. To provide fault tolerance, it is possible to employ Infiniswap's mechanism [12], which persists evicted pages to local storage asynchronously.

5 EVALUATION

In this section, we evaluate DiLOS over the state-of-the-art implementations. As a result, DiLOS outperforms AIFM up to 1.52× and Fastswap up to 2.2×. Moreover, app-aware prefetcher improves in-memory key-value store performance 27% further.

5.1 Comparison with the State-of-the-Arts

To show the performance enhancement of our design, we compare DiLOS with the state-of-the-art memory disaggregation implementations, both library-based (AIFM [40]) and kernel-based (Fastswap [2]).

Since AIFM and its benchmarking suites require application modifications and a Shenango runtime [35], which does not run on unikernels, it is difficult to compare DiLOS with AIFM directly. Therefore, for a fair comparison with AIFM, we resort to the only real-world workload already instrumented in the evaluation of AIFM, DataFrame [33]. We leave it as future work to port the entire suite of AIFM's benchmark into the DiLOS environment and conduct a more rigorous analysis.

Testbed. Our experimental testbed consists of a computing node and a memory node. Each node has two Intel E5-2670 v3, DDR4 RAM (110G for computing node and 440G for memory node), and single Mellanox CX556A EDR/100GbE card. A 100GbE cable connects the two nodes. We use OSv 0.55, QEMU 4.1.1, and Mellanox OFED 5.0 on top of Debian 10 (Linux 4.19) for DiLOS. Due to compatibility issues, Fastswap runs on Ubuntu 16.04 (Linux 4.11) and Mellanox OFED 4.3, and AIFM runs on Ubuntu 18.04 (Linux 5.0) and Mellanox OFED 4.6. All AIFM's compute offloading features are disabled. To limit available local memory size, we use LXC container for Fastswap, `kCacheGBs` constant for AIFM, and `m` parameter of QEMU for DiLOS. Our remote memory server uses 1GB huge pages. Thus, we modify Fastswap's remote memory server to use 1GB huge pages for a fair comparison. All implementations except for AIFM use RoCE RDMA. We tried to port AIFM to use RDMA rather than using TCP, but it worsens performance. We suspect there is a scalability problem of RDMA NIC since AIFM spawns hundreds of connections for parallel requests. Therefore, we use the original AIFM using TCP for experiments.

Data analytic application. We evaluate the end-to-end performance of DiLOS compared to AIFM and Fastswap. We run a data analytic application, DataFrame [33], on each system and use the New York City taxi trip analysis workload [16] as AIFM does. In our testbed configuration, this workload requires about 40GB of peak memory usage, and thereby we limit the size of local cache to 20GB, 10GB, and

[1]We use SLOCCount (for new) and git (for modification) to measure lines of code

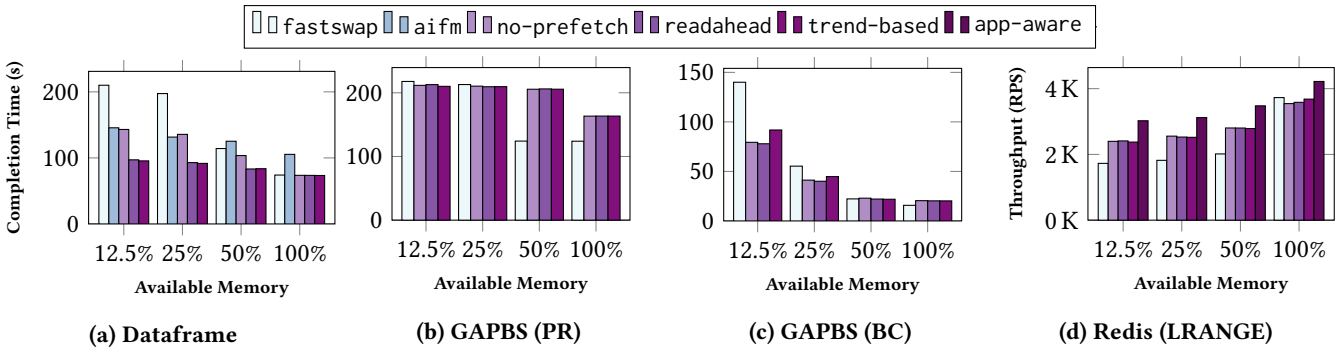

Figure 3: Performance comparison among Fastswap [2], AIFM [40], DiLOS without prefetcher, DiLOS with Readahead prefetcher [14], DiLOS with majority trend-based prefetcher [32], and DiLOS with app-aware prefetcher.

5GB. We used the DataFrame source code provided and used by AIFM, publicly available in [39].

Figure 3a shows the total completion time of the DataFrame application on each system. When the local cache size is large enough (*i.e.*, there is no need to use remote memory), DiLOS, as well as Fastswap, is 30% faster than AIFM. This gap stems from preserving memory abstraction in the kernel while AIFM proactively checks every pointer before dereferences regardless of the use of remote memory. As the local cache size decreases, the performance of Fastswap drops dramatically (184% slowdown) due to frequent page faults. AIFM reduces the slowdown by avoiding page faults, and it shows 1.44× higher performance than Fastswap when there is 12.5% of required memory in local. DiLOS, on the other hand, obtains comparable completion time to AIFM even without any prefetching. With general-purpose prefetching, DiLOS outperforms AIFM by 33%.

The results demonstrate that DiLOS achieves even better performance than AIFM for particular workloads. We conjecture that three factors contribute to this. First, since DiLOS reduces page fault overheads as much as possible, it rivals AIFM, which has zero page fault cost. Second, R/W amplification does not happen in DataFrame. DataFrmae's core data structure, `DataFrame`, is just a collection of vectors and thus has high spatial localities. Lastly, AIFM's communication between computing node and memory node is based on TCP. Even if we use the same hardware and layer-2 protocol (Ethernet), we find that AIFM's TCP stack throttles the eviction and fetching performance compared to RoCE.

Graph processing application. We use GAPBS (GAP Benchmark Suite) [6] to evaluate our implementations on multi-threaded application workload. We use two representative algorithms, page rank (PR) and betweenness centrality (BC), with the Twitter dataset [22]. These workloads have about 17GB of the total working set. We use GAPBS 1.3, which is the lastly released version. For all experiments, we limit the number of threads to 4 using `OMP_NUM_THREADS` and local cache to 14.0GB (100%), 7.0GB (50%), 3.5GB (25%), and 1.75GB (12.5%).

Figure 3b and Figure 3c show the completion time of each implementation. Unlike other experiments, DiLOS has a 22-25% performance drawback under 100% (14GB) local cache condition. It is because OSv has a scalability problem of synchronization methods. However, under 12.5% (1.75GB) local cache, DiLOS has a little higher performance on PR and up to 1.80× faster performance on BC.

In-memory key-value data store application. We conduct a performance evaluation using Redis, a popular in-memory key-value store [41]. In applications such as Redis, the memory access pattern is highly unpredictable due to their underlying pointer-based data structures. We use Redis 5.0.7, the latest stable version at the time of the experiments. We turn off Redis's disk-related features such as AOF and RDB to isolate memory-related performance from disk-related tasks. We also turn off Redis's eviction to overcommit memory and induce eviction to remote. We evaluate the performance of DiLOS with `20GB` (100%), `10GB` (50%), `5GB` (25%), and `2.5GB` (12.5%) local cache.

We use `LRANGE` workload [24] for the benchmark. It uses a quick list data structure, which stores strings in a linked-list [44]. It is heavily used to deal with sequential data such as thread conversation [9, 42]. We evaluate `LRANGE_100` performance with `redis-benchmark` [23], which retrieves the front 100 elements from a list. Since vanilla `redis-benchmark` uses only one list, which is not realistic in modern datacenters, we modified it to use 100 thousand lists. To populate the lists, we pushed 20 million elements (about 20GB) to the lists randomly. Then we send `LRANGE` operation 100K times.

Figure 3d shows performance improvements in Redis. For all experiments, DiLOS outperforms Fastswap. DiLOS, even without any prefetcher, has up to 1.39× higher throughput than Fastswap. However, general-purpose prefetchers do not

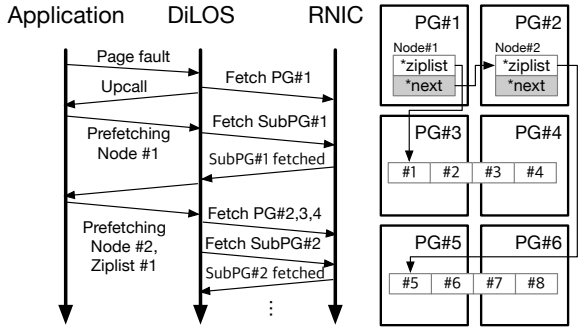

Figure 4: Quick List Prefetcher. PG and SubPG mean page and subpage, respectively.

improve `LRANGE` performance where the access pattern is irregular. `readahead` enhances throughput only 0.5% and `trend-based` somewhat worse performance 1%.

5.2 App-aware Prefetcher

To show how DiLOS uses application semantic, we add an `LRANGE` prefetching guide to enhance Redis's performance further. The prefetching guide decides pages to retrieve using hints from Redis's data structures.

Redis's `LRANGE` returns a set of objects from a list. For example, `LRANGE list 0 100` returns objects from index 0 to 100 in `list`. The range parameter is a good hint for which pages should be prefetched. When a range query request arrives, the prefetching guide is invoked on a page fault, and it traverses and prefetches list nodes of the specified range. Because list nodes are not always contiguously allocated, sequential prefetcher does not predict them correctly. On the other hand, the guide collects required objects to be fetched and informs DiLOS to prefetch the objects precisely.

Figure 4 shows app-aware prefetching on range query. When a page fault raises, the prefetcher fetches its list node (subpage #1) along with page #1. After the subpage fetching is complete, the prefetcher fetches pages associated with `ziplist` (page #3,4) and the next entry (page #2). The prefetcher does this iteratively until the last item requested in the query.

Figure 3d shows overall performance improvements of the app-aware prefetcher. While general-purpose prefetchers do not have any performance improvement compared to `no-prefetch`, the app-aware prefetcher improves DiLOS's performance up to 26%.

6 RELATED WORK

Memory disaggregation systems. Many memory disaggregation systems have been proposed. Kernel-based systems using swap device [1, 12], frontswap [2, 25, 32], and split kernel [43] have been proposed to run existing applications. Library-based systems have been introduced recently to exploit application semantic such as GC structure [47] and access pattern [40]. DiLOS uses a unikernel approach, which runs unmodified applications without huge overheads and uses application semantic to tune prefetcher.

Unikernels. Since unikernel was first introduced [28], researchers have proposed two kinds of unikernels: POSIX-based and language-based. The first one aims to support Unix applications on a unikernel. For example, OSv [17] and HermiTux [34] build new unikernels to support Linux binaries, Lupine [21] and UKL [38] convert Linux kernel to unikernel, and Unikraft [19] generates specialized POSIX-compatible unikernels automatically. Language-based unikernels [3, 8, 11, 28] offer a language-specific interface to build a unikernel instance. Researchers also have introduced various systems using unikernel for NFV [20, 31], instant booting [30, 48], enclave [5], and HPC [26]. We choose OSv unikernel that is mature enough and supports unmodified Linux binaries.

7 CONCLUSION

This paper claims unikernels are a promising platform for memory disaggregation. Without compromising compatibility, DiLOS demonstrates superior performance than the user-level approaches (52%) and other kernel-level systems (120%).

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT). (2019M3F2A1072211 and 2019R1A2C2008439)

REFERENCES

[1] Mellanox Accelio. nbdX. https://github.com/accelio/NBDX.
[2] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. Can Far Memory Improve Job Throughput? In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.
[3] Fran. J. Ballesteros. Clive. https://lsub.org/clive/.
[4] Mike Battersby and Michael Kerrisk. sigaction. http://man7.org/linux/man-pages/man2/sigaction.2.html.
[5] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding applications from an untrusted cloud with Haven. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 267–283, Broomfield, CO, October 2014. USENIX Association.
[6] Scott Beamer, Krste Asanović, and David Patterson. The GAP Benchmark Suite, 2017.
[7] Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazières, and Christos Kozyrakis. Dune: Safe User-level Access to Privileged CPU Features. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 335–348, Hollywood, CA, October 2012. USENIX Association.

[8] A. Bratterud, A. Walla, H. Haugerud, P. E. Engelstad, and K. Begnum. IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 250–257, 2015.

[9] 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*, SoCC '10, page 143–154, New York, NY, USA, 2010. Association for Computing Machinery.

[10] IBM Corporation. and Michael Kerrisk. userfaultfd. http://man7.org/linux/man-pages/man2/userfaultfd.2.html.

[11] Galois, Inc. The Haskell Lightweight Virtual Machine. https://github.com/GaloisInc/HaLVM.

[12] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient Memory Disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, Boston, MA, March 2017. USENIX Association.

[13] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who Limits the Resource Efficiency of My Datacenter: An Analysis of Alibaba Datacenter Traces. In *Proceedings of the International Symposium on Quality of Service*, IWQoS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Ying Huang. mm, swap: VMA based swap readahead. https://lwn.net/Articles/716296/.

[15] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design Guidelines for High Performance RDMA Systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 437–450, Denver, CO, June 2016. USENIX Association.

[16] Kartik Kannapur. NYC Taxi Trips - Exploratory Data Analysis. https://www.kaggle.com/kartikkannapur/nyc-taxi-trips-exploratory-data-analysis/notebook.

[17] Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov. OSv—Optimizing the Operating System for Virtual Machines. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 61–72, Philadelphia, PA, June 2014. USENIX Association.

[18] Waldemar Kozaczuk. OSv Linux ABI Compatibility. https://github.com/cloudius-systems/osv/wiki/OSv-Linux-ABI-Compatibility.

[19] Simon Kuenzer, Vlad-Andrei Bădoiu, Hugo Lefeuvre, Sharan Santhanam, Alexander Jung, Gaulthier Gain, Cyril Soldani, Costin Lupu, Ştefan Teodorescu, Costi Răducanu, Cristian Banu, Laurent Mathy, Răzvan Deaconescu, Costin Raiciu, and Felipe Huici. Unikraft: Fast, Specialized Unikernels the Easy Way. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, page 376–394, New York, NY, USA, 2021. Association for Computing Machinery.

[20] Simon Kuenzer, Anton Ivanov, Filipe Manco, Jose Mendes, Yuri Volchkov, Florian Schmidt, Kenichi Yasukata, Michio Honda, and Felipe Huici. Unikernels Everywhere: The Case for Elastic CDNs. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '17, page 15–29, New York, NY, USA, 2017. Association for Computing Machinery.

[21] Hsuan-Chi Kuo, Dan Williams, Ricardo Koller, and Sibin Mohan. A Linux in Unikernel Clothing. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[22] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a Social Network or a News Media? In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 591–600, New York, NY, USA, 2010. Association for Computing Machinery.

[23] Redis Labs. How fast is Redis? https://redis.io/topics/benchmarks.

[24] Redis Labs. LRANGE - Redis. https://redis.io/commands/lrange.

[25] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, and et al. Software-Defined Far Memory in Warehouse-Scale Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 317–330, New York, NY, USA, 2019. Association for Computing Machinery.

[26] Stefan Lankes, Simon Pickartz, and Jens Breitbart. HermitCore: A Unikernel for Extreme Scale Computing. In *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '16, New York, NY, USA, 2016. Association for Computing Machinery.

[27] Jiuxing Liu, Wei Huang, Bülent Abali, and Dhabaleswar K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. In Atul Adya and Erich M. Nahum, editors, *Proceedings of the 2006 USENIX Annual Technical Conference, Boston, MA, USA, May 30 - June 3, 2006*, pages 29–42. USENIX, 2006.

[28] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library Operating Systems for the Cloud. *SIGARCH Comput. Archit. News*, 41(1):461–472, March 2013.

[29] Anil Madhavapeddy and David J. Scott. Unikernels: Rise of the Virtual Library Operating System. *Queue*, 11(11):30–44, December 2013.

[30] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. My VM is Lighter (and Safer) than Your Container. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 218–233, New York, NY, USA, 2017. Association for Computing Machinery.

[31] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. ClickOS and the Art of Network Function Virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 459–473, Seattle, WA, April 2014. USENIX Association.

[32] Hasan Al Maruf and Mosharaf Chowdhury. Effectively Prefetching Remote Memory with Leap. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 843–857. USENIX Association, July 2020.

[33] Hossein Moein. DataFrame. https://github.com/hosseinmoein/DataFrame.

[34] Pierre Olivier, Daniel Chiba, Stefan Lankes, Changwoo Min, and Binoy Ravindran. A Binary-Compatible Unikernel. In *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE 2019, page 59–73, New York, NY, USA, 2019. Association for Computing Machinery.

[35] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving high cpu efficiency for latency-sensitive datacenter workloads. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, NSDI'19, page 361–377, USA, 2019. USENIX Association.

[36] Jonas Pfefferle, Patrick Stuedi, Animesh Trivedi, Bernard Metzler, Ionnis Koltsidas, and Thomas R. Gross. A Hybrid I/O Virtualization Framework for RDMA-Capable Network Interfaces. *SIGPLAN Not.*, 50(7):17–30, March 2015.

[37] QEMU. Paravirtualized RDMA Device (PVRDMA). https://github.com/qemu/qemu/blob/master/docs/pvrdma.txt.

[38] Ali Raza, Parul Sohal, James Cadden, Jonathan Appavoo, Ulrich Drepper, Richard Jones, Orran Krieger, Renato Mancuso, and Larry Woodman. Unikernels: The Next Stage of Linux's Dominance. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '19, page 7–13, New York, NY, USA, 2019. Association for Computing Machinery.

[39] Zhenyuan Ruan. AIFM. https://github.com/aifm-sys/aifm.

[40] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. AIFM: High-Performance, Application-Integrated Far Memory. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 315–332. USENIX Association, November 2020.

[41] Salvatore Sanfilippo. Redis. https://redis.io.

[42] ScaleGrid. Top Redis Use Cases by Core Data Structure Types. https://scalegrid.io/blog/top-redis-use-cases-by-core-data-structure-types/.

[43] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 69–87, Carlsbad, CA, October 2018. USENIX Association.

[44] Matt Stancliff. Redis Quicklist - From a More Civilized Age. https://matt.sh/redis-quicklist.

[45] Shelby Thomas, Geoffrey M. Voelker, and George Porter. CacheCloud: Towards Speed-of-light Datacenter Communication. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, Boston, MA, July 2018. USENIX Association.

[46] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The next Generation. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[47] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. Semeru: A Memory-Disaggregated Managed Runtime. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 261–280. USENIX Association, November 2020.

[48] Dan Williams and Ricardo Koller. Unikernel Monitors: Extending Minimalism Outside of the Box. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, Denver, CO, June 2016. USENIX Association.

[49] Jian Yang, Joseph Izraelevitz, and Steven Swanson. FileMR: Rethinking RDMA Networking for Scalable Persistent Memory. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 111–125, Santa Clara, CA, February 2020. USENIX Association.