Boosting Data Center Performance via Intelligently Managed Multi-backend Disaggregated Memory

1st Jing Wang
Shanghai Jiao Tong University
jing618@sjtu.edu.cn

2nd Hanzhang Yang
Shanghai Jiao Tong University
linqinluli@sjtu.edu.cn

3rd Chao Li ✉
Shanghai Jiao Tong University
lichao@cs.sjtu.edu.cn

4th Yiming Zhuansun
Shanghai Jiao Tong University
zsym2019@sjtu.edu.cn

5th Wang Yuan
Shanghai Jiao Tong University
wangyuan05@sjtu.edu.cn

6th Cheng Xu
Shanghai Jiao Tong University
jerryxu@sjtu.edu.cn

7th Xiaofeng Hou
Shanghai Jiao Tong University
hou-xf@cs.sjtu.edu.cn

8th Minyi Guo
Shanghai Jiao Tong University
guo-my@cs.sjtu.edu.cn

9th Yang Hu
Tsinghua University
hu_yang@tsinghua.edu.cn

10th Yaqian Zhao
IEIT SYSTEMS Co., Ltd.
zhaoyaqian@ieisystem.com

***Abstract*—Existing disaggregated memory (DM) systems face a problem of underutilized far memory bandwidth, which greatly limits the data throughput when processing data-intensive applications. Specifically, prior works all target runtime design for a single PCIe-based secondary memory device (i.e., single-backend far memory) with low data bandwidth and high system overhead.**

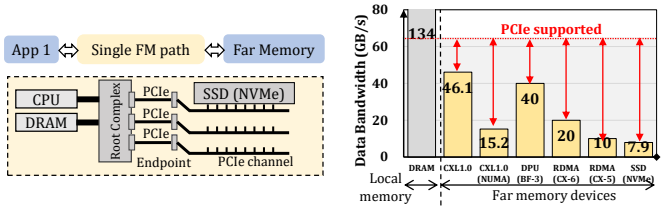
In this work, we take the first step to realize a well-crafted, multi-backend DM system with scale-out far memory paths. We propose xDM, a novel DM management scheme that can dynamically build and implicitly select appropriate far memory access paths. As part of xDM, we devise a smart far memory configuration strategy that can further optimize bandwidth usage effectiveness by tuning a wide set of key parameters based on synthesized information of application page data. Our design shows up to $3.9\times$ data swap performance speedup, $2.8\times$ data throughput increase, and $5.1\times$ data center task throughput improvement compared with state-of-the-art works.

***Index Terms*—multi-backend, far memory, multi-path, swap**

I. Introduction

The proportion of data-intensive applications (e.g., graph processing, data mining, AI training, and AI agent) in today's high-performance data center is experiencing rapid growth [1]–[8]. Big data centers in the cloud [9]–[12] and micro data centers near the edge [13], [14] all necessitate large memory capacity and efficient data management. It is quite important to provide flexible and powerful memory services for the widely used cross-layer services in the cloud-native environments [15]–[19]. In recent years, disaggregated memory (DM) architecture stands out as a way to enhance data center capabilities by offering highly flexible memory expansion [20]–[25]. In this case, a memory-hungry monolithic server can access a PCI Express (PCIe) based secondary memory device (i.e., far memory) with low data access latency.

While the addition of far memory (FM) could relieve a server's memory pressure, it unfortunately cannot meet the needs of high data/task throughput in today's data center. As shown in Figure 1-(a), prior works mostly limit their designs on a single FM device, which is often plugged into the server's PCIe Add-in-Card (AIC) interface as a fast backing store [26]–[31]. Figure 1-(b) compares a variety of commercial FM technologies. There is a wide gap between the data transfer bandwidth that a single FM device could support (from 7.9 to 46 GB/s) [25], [32]–[35] and the maximum available bandwidth that modern PCIe protocol could provide (64 GB/s on PCIe 4.0×16) [36]. This means that the single far memory device could become a crucial bottleneck for scale-out applications.

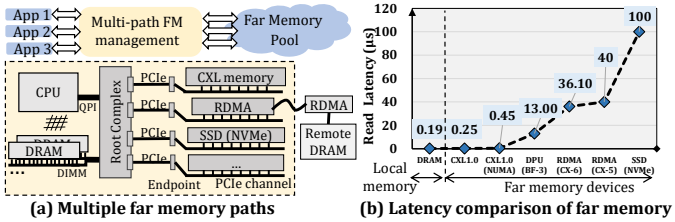


(a) Single far memory path **(b) Bandwidth comparison of far memory**

Fig. 1: (a) Traditional single-backend far memory. (b) Bandwidth comparison of various far memory technologies, including CXL 1.0, DPU card of BlueField 3, ConnectX-5/ConnectX-6 RDMA card, and NVMe-based SSD.

Incorporating multiple FM devices and oversubscribing the PCIe subsystem allows one to push the limit of server data throughput, as shown in Figure 2-(a). This approach, which we call *multi-backend disaggregated memory*, unfortunately cannot be implemented solely based on current data transfer protocols and device drivers. The key reason is that the virtual machines (VMs) still use a hierarchical data swap mechanism with the host operating system (OS) involved, which does not allow pages to be swapped to/from multiple swap backends (each associated with a different backing store). In other words, existing FM management schemes are blind to the

SC24, November 17-22, 2024, Atlanta, Georgia, USA
979-8-3503-5291-7/24/$31.00 ©2024 IEEE



(a) Multiple far memory paths (b) Latency comparison of far memory

Fig. 2: (a) System with multi-backend far memory. (b) Access latency of different far memory backends. We transfer 64MB data with page granularities (4KB) and test the latency on each far memory backend.

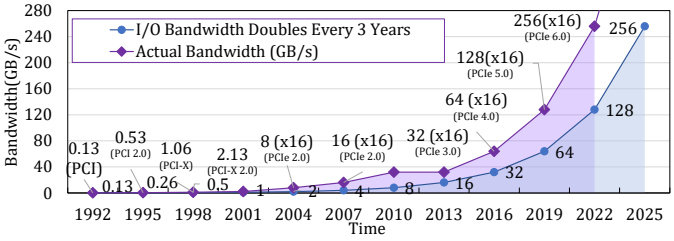
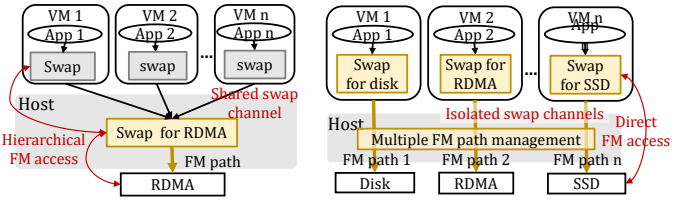


Fig. 3: I/O bandwidth trend doubles every 3 years [36].



(a) Single far memory access path (b) Multiple far memory access paths

Fig. 4: (a) Prior works can only access a single FM path with a shared swap channel based on traditional hierarchical data swap. (b) An attractive alternative is to provide multiple directed-connected FM paths with isolated swap channels.

possible multiple physical FM channels: they logically only support one data exchange path to FM devices.

We introduce *xDM*, a novel FM management system engineered to support high-performance data analytic workloads running on the new multi-backend disaggregated memory architecture. To realize this, a non-trivial *transition* from single-path to multi-path memory management is necessary. The use of multiple simultaneous FM access paths enables a server to access FM devices in parallel, thereby substantially improving the overall data throughput. We carefully tailor the VM OS to support multi-path FM management. xDM features a low-overhead switchable data swap module that allows memory-hungry machines to customize their FM access paths. As a result, different users are able to dynamically choose appropriate FM backends based on their specific workload characteristics and resource pressure, eliminating the need to stall tasks while awaiting the availability of machine bandwidth.

Importantly, the creation of multiple FM access paths brings new performance optimization opportunities as well. For example, Figure 2-(b) shows that FM devices exhibit a wide range of latency. Previous single-backend far memory systems only allocate local/FM resources to tasks based on a simple analysis of their sensitivity to memory access latency [27], [37], [38]. The introduction of multiple FM management paths actually offers more knobs for fine-tuning FM data access. We decompose far memory configuration down to a multi-dimensional parameter tuning problem. We overhaul xDM's far memory configuration strategy so that it can adapt to applications according to a rich synthesis of page trace information. As a result, it can greatly boost bandwidth usage effectiveness as well as save memory, I/O, and network bandwidth, thus improving the overall performance of the data center.

This paper makes markable research contributions and practical engineering contributions, listed as follows.

- We examine the possibilities and opportunities of a new multi-backend DM architecture. We propose and design xDM, the first multi-path FM management system that not only allows *simultaneous* access of multiple FM devices but also provides *dynamic and implicit* adjustment of FM paths based on program behaviors.
- We devise an intelligent configuration console which makes xDM a truly *versatile and smart* system. Our method can synthesize multi-dimensional page information and configure FM devices from different aspects. This guarantees highly effective bandwidth usage that no prior FM system could satisfactorily provide.
- We implement xDM as a prototype and evaluate it on a wide range of popular applications. Our system shows up to 3.9× swap performance speed up, 2.8× data throughput increase, and 5.1× data center task throughput improvement compared with state-of-the-art baselines with Quality of Service (QoS) guarantee.

The remainder of this paper is organized as follows. Section II provides more details of the background. Section III introduces key design considerations and gives a system overview. Section IV proposes multi-path FM management. Section V presents evaluation methodology and results. Section VI discusses related works and Section VII concludes the paper.

II. Background and Motivation

This section presents more details about the background and challenges that motivate this work. Ideally, a well-designed FM system needs to achieve two goals: high PCIe bandwidth utilization and high bandwidth usage effectiveness.

A. Far Memory Usage Bottleneck

PCIe has served as the de facto interconnect solution for most of the FM devices. The PCI Special Interest Group predicts that speeds will double approximately every three years, as shown in Figure 3. The growing demand for data has driven I/O bandwidth to increase faster than expected. To meet the data bandwidth requirements of data-intensive markets in the era of cloud-native deployment and AI-driven applications, this trend will continue. Currently, PCIe 5.0 protocols can offer a bandwidth of 128 GB/s, providing a very large bandwidth that any single FM device will not saturate. This demonstrates the importance of efficiently utilizing the

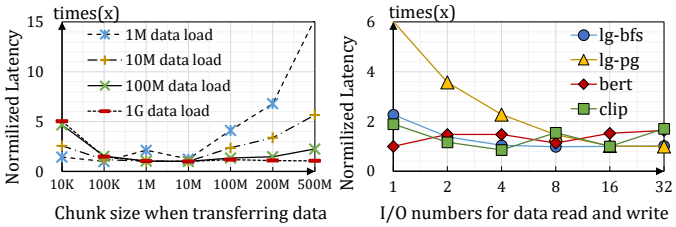


Fig. 5: Impacts of data granularity and I/O width.

growing I/O bandwidth. Looking ahead, if we stick to the single-backend DM architecture, it will cause under-utilized computation power and highly limited serving throughput. This observation mandates the design and optimization of multi-backend DM architecture.

Nevertheless, the requirement of multiple parallel FM access paths actually goes beyond the capabilities of the current host operating system. Previous far memory system designs all rely on a single-path swapping mechanism, where the original disk-based swap backend is replaced with RDMA [27], [28]. A straightforward idea is to add more swap backends, but this runs into complications as adapting different backends requires unique modifications to the single swap frontend [39]. In general, it is impractical to directly support multi-path far memory management in the OS.

An unexplored yet viable option is to create and manage FM access paths within virtual machine (VM). While today's virtualization technology can deliver performance that is comparable to bare-metal servers, it unfortunately cannot manage multi-backend disaggregated memory architecture very well. We show the normalized data transfer latency the Y-axis of Figure 4, and the "times(x)" indicates the multiple of speedup. As shown in Figure 4-(a), a naive VM-based FM system has two key limitations. First, existing hosted VMs generally employ a single shared swap channel, and therefore one cannot dynamically assign different FM swap backends for different VMs. Second, current designs all use a hierarchical swap architecture [40], [41], which incurs considerable overhead due to two data swapping processes (swapping data between the VM's swap space and the host operating system's swap space, plus swapping between the host OS and the far memory).

In sum, multi-path far memory access shows great promise. Removing FM usage bottleneck requires expanding data swap channels and tapping into light-weight swapping, as depicted in Figure 4-(b). It is desirable to setup FM path directly within guest OS to support multiple different FM paths. Meanwhile, a non-hierarchical swapping approach can bypass the host and reduce overhead. The key challenges here include how to build isolated FM data swap channels and how to enable individual configuration of each FM swap backend.

B. Far Memory Usage Effectiveness

Increasing the system's PCIe bandwidth utilization alone does not necessarily mean high performance. At the application level, it is important to also optimize the effectiveness

Related works	to Block Device	to RDMA	Hybrid	Multi-path
Linux zswap [42]	✓	×	×	×
Fastswap [27]	×	✓	×	×
TMO [37]	✓	×	✓	×
XMemPod [40]	✓	✓	✓	×
Pond [31]	✓	×	×	×
xDM (Ours)	✓	✓	✓	✓

TABLE I: Single-path vs. multi-path far memory systems.

Related works	Data Ratio on FM	Data Ratio on NUMA	Data Granularity	I/O Width
Linux zswap [42]	✓	×	×	×
Fastswap [27]	✓	×	×	×
TMO [37]	✓	×	×	×
XMemPod [40]	✓	×	×	×
Pond [31]	✓	✓	×	×
xDM (Ours)	✓	✓	✓	✓

TABLE II: Comparison of key tuning knobs of far memory configuration used in related works.

of FM data access, namely, improved task performance under given bandwidth consumption.

Most of the prior works follow a simple idea: by offloading part of data to far memory based on workload behaviors, the local DRAM can retain more latency-sensitive tasks [27], [37], [42]. They have proposed low-overhead FM management software [43], [44] and different schemes for data distribution control between local DRAM and far memory [27], [37], [42].

Determining the most suitable FM configuration is crucial for highly effective FM access, requiring a deep understanding of tasks' memory access behaviors. For regular data processing applications such as AI model computing, tasks involve continuous memory operations including data copying and matrix calculations [5]–[7], [45], [46]. On the other hand, irregular data processing applications, like graph processing [1], [2], [47], [48], entail random memory access operations such as data searching and graph structure traversal.

Applications often exhibit more complex performance variation, which requires a detailed analysis of page behaviors. Specifically, we evaluate the impact of data access granularity and data transfer channel width in Figure 5. We test the end-to-end latency of data loading from RDMA and Figure 5-(a) shows the changing trends on different data unit sizes. We analyze the varying patterns of latency changes that occur with the addition of disk I/O access widths for graph processing *(lg-bfs, lg-pg)* and AI inference *(bert, clip) workloads, details of which are listed in Table V.* Results aredepicted in Figure 5-(b). Previous works unfortunately overlook the importance of fine-tuning the above key system parameters, which could significantly influence the application performance on multi-backend disaggregated memory.

III. Design Overview

Before getting into technical details, we first describe our key design considerations with respect to multi-path FM management and give a brief overview of the overall architecture.

A. Design Philosophy

Realizing multi-path simultaneous FM access is non-trivial. In this work, we set two big yet attainable goals:

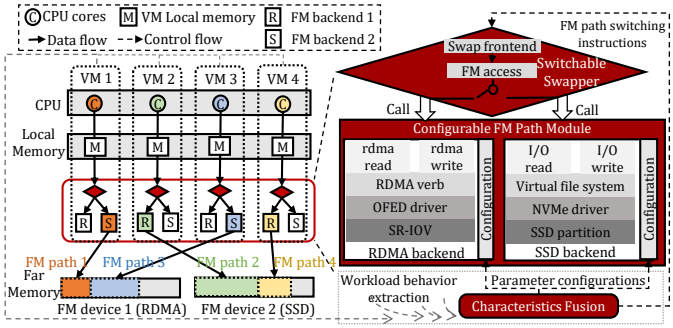


Fig. 6: The overview of xDM system design.

- *Make it dynamic and implicit*. One possible solution is to make the system static and implicit. It means that users have to specify a flavor when they launch their compute instances on FM-enabled machines. It actually binds an instance to a specific physical FM channel (e.g., RDMA or SSD). Co-locating different user instances allows the server node to simultaneously access different FM backends, thereby increasing bandwidth utilization. Since workload behaviors often change during runtime, such a design only yields sub-optimal performance. In this work, we aim to make the system dynamic: each instance can evaluate task preferences during runtime and implicitly select the optimal FM path without the need of user intervention. To our knowledge, previous works never implement a static multi-path FM system, not to mention a dynamic one. The differences between our work and related studies are shown in Table I.
- *Make it versatile and smart*. Instead of simply manipulating the distribution of data across local DRAM and far memory devices, xDM aims to be a truly versatile system. To improve FM bandwidth usage effectiveness and boost data center performance, one needs to take additional configurable parameters into account. It is important to leverage a rich set of application page data and adjust system settings based on multi-dimensional system information including data distribution, data granularity, as well as I/O characteristics. This allows us to make informed decisions and enable xDM to smartly configure each FM path. To the best of our knowledge, previous works all use simple and straightforward data management strategies. The differences between our work and related studies are summarized in Table II.

B. Overview of xDM

In this work, we scale out the physical far memory channels and build a multi-backend far memory system xDM, as Figure 6 shows. We aim to make the best use of the available FM resources for lower task latency and high data throughput.

The basic idea is to decouple the management of far memory backends from the traditional swap mechanism to swap data at high bandwidth. We find that the traditional hierarchy-based memory access pattern is inefficient due to the single data swap path on very deep memory hierarchies. A flat

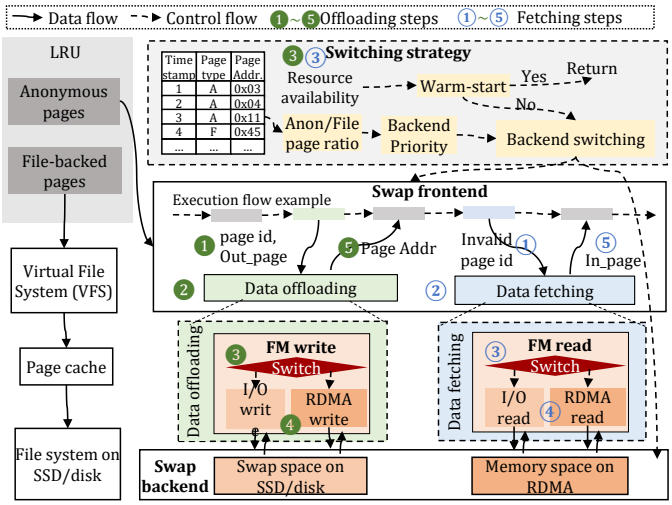


Fig. 7: The proposed dynamic switching scheme of far memory paths based on switchable far memory swapper.

data management method for disaggregated memory systems is required, providing flexibility for each application to choose and use the proper far memory path directly and quickly. To achieve this, we devise a switchable FM swapper module and multiple FM path configuration modules. The FM swapper module's job is to select an appropriate FM path and assign reasonable far memory space for applications. The FM path configuration module provides low-overhead configuration of various tunable parameters of the FM paths. We further extract important workload behaviors from three dimensions and use a rich synthesis of system information to intelligently switch and configure the FM access paths. In this way, xDM allows customized high-performance far memory path configurations to utilize architectural design optimizations and skip certain data fetching overheads on far memory paths.

xDM considers both scalability and extensibility. This practical and portable design can easily migrate to other far memory environments and real-world applications. xDM analyzes page data at the system level, which is all transparent to applications. Its FM swapper module is designed to seamlessly scale with the addition of new FM devices and access paths. Meanwhile, its path configuration module is ready to incorporate additional system attributes to better accommodate new application scenarios.

IV. Multi-path Far Memory Management

In this section, we discuss the key system and software support for multi-path far memory management. We first introduce xDM's dynamic FM switching mechanism (Sec. IV-A). We then describe xDM's intelligent configuration strategy for high resource usage effectiveness (Sec. IV-B).

A. Dynamic FM Switching Mechanism

The foundational component of xDM is a dynamically switchable FM swapper module. As Figure 7 shows, it has two main features: 1) a low-overhead, switchable FM swapper, and 2) an efficient, implicit FM switching strategy.

1) Switchable FM Swapper: It is basically a modified swap frontend plus a variety of adaptive FM swap backends. Leveraging the data swap interfaces provided by Frontswap [39], we redirect page swapping to our customized FM read and write functions, as shown in Figure 7.

The modified swap frontend allows the use of flexible FM access APIs to support multiple and heterogeneous far memory paths. As a bridge between the page management process and the swap backends, it needs to cooperate well with multiple FM backends. As shown in Figure 7, it continuously performs data offloading by receiving swapped-out page entries from the running processes and data fetching by refining the required swapped-in pages back to the related processes (① and ⑤). Since only anonymous pages drawn from LRU lists are involved in the page fault operations, the frontend skips file-backed page operations directly. The swap mechanism for data offloading and data fetching internally calls the FM access interfaces (②), which are adeptly integrated into the existing page fault handling workflow. The interface can switch the usage of far memory access channels (③ and ④) with specified parameters *switch_to_SSD* and *switch_to_RDMA*.

We prepare a set of pre-configured FM backend modules to serve as swapper backends. They handle the swap-out data offloading and swap-in data fetching, including signal and command transferring, data writing and indexing, etc. Two primary backend modules are switchable RDMA backend and switchable SSD backend. The former uses SR-IOV (Single Root I/O Virtualization) to generate virtualized RDMA card for each VM, and the latter creates separate swap spaces by mounting different swap files on isolated SSD partitions. The system can support more far memory types by injecting corresponding far memory access APIs into the swap frontend. Each FM backend module functions as a supplementary patch to the original swap kernel. Implementing these patches into the OS entails kernel recompiling overhead. To streamline this process and minimize compilation time, we proactively assemble FM backend modules as backups for low-overhead switching. We use a listening queue in the frontend to handle page data synchronization between page cache and FM backends.

2) Implicit FM Switching Strategy: There is not a universal FM path setting that can meet the needs for all. We examine the proportion of anonymous pages and file-backed pages from the page trace table. A key observation is that the ratio of anonymous to file-backed pages provides a good indication of the preferred far memory usage, as shown in Figure 8. For example, FM switching from SSD to RDMA can greatly boost application performance, as demonstrated by the improved performance of graph workload *lg-bc* and array sorting workload *sort* in Figure 8. Nevertheless, graph traversal workload *gg-bfs* and floating computing workload *lpk* exhibit similar execution latency on both SSD and RDMA backends. In this case, SSD is preferred compared with the expensive RDMA-connected memory devices.

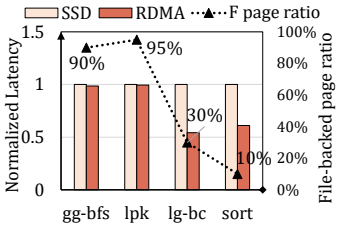


Fig. 8: Workloads with more file-backed (anonymous) pages prefer SSD (RDMA) backends.

We design a far memory switching strategy to choose FM backends with the highest effectiveness based on the above analysis. Both page distribution statistics and application's sensitivity to different FM paths are useful when making switching decisions. We use a new metric *memory effectiveness improvement (MEI)*, defined as the quotient of runtime performance improvement divided by the far memory device cost. We label the backend priority of different workloads by ordering the obtained *MEI* value. We maintain a list of available backend that represents each backend's availability.

We use warm-start optimizations to reduce the switching overhead of each compute instance. The execution flow of backend switching consists of three steps: (1) We initialize multiple virtualized far memory devices when starting each VM, which allows backend switching without shutting down VMs. (2) We then prioritize placing new tasks in VMs already equipped with the required far memory backends. (3) If no suitable VM is available, the task is assigned to an active VM, which then switches to the preferred far memory backend.

B. Smart FM Configuration Console

After FM backend switching, the next question is how to configure the far memory access path to make the best use of the I/O bandwidth. The abstraction and tuning of far-memory parameters are non-trivial and complex. Each parameter has a wide range of available candidates, which can cause different memory occupation and execution latency. In real-world environments, the states of the VM, memory, and network affect the path choices and configurations of far memory. To handle the complexity, we devise a versatile FM configuration console that can smartly fuse data characteristics and set up multi-dimensional parameters for each far memory data access path, as illustrated in Figure 9.

1) Data Characteristic Fusion: By analyzing page data, we find that data granularity, I/O width, and data distribution significantly impact the performance and resource usage.

First, when accessing far memory, the *data granularity* of transferred data has an obvious influence on application performance. For example, the end-to-end latency of data loading from RDMA shows the changing trends on different data unit sizes, as shown in Figure 5-(a). The reason behind this can be that the data fragment distribution affects the ratio of available data when transferring each data unit. Thus, a better configuration on data granularity is critical. We analyze data segments formed from contiguous memory addresses and their distribution, as shown in Figure

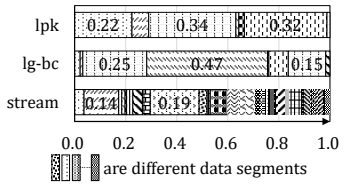
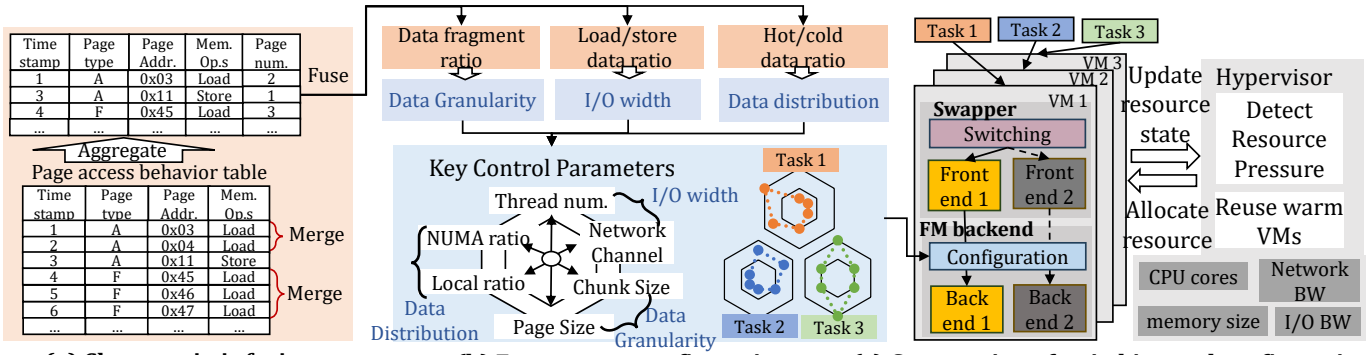


Fig. 10: Data segments with different data fragment ratios.



(a) Characteristic fusion **(b) Far memory configuration** **(c) Cooperation of switching and configuration**

Fig. 9: xDM's adaptive FM parameter configuration. The characteristic fusion module provides a synthesis of information from page access traces. xDM can configure and fine-tune multi-backend FM from various aspects.

10. This guides us to predict and choose the optimal data granularity by identifying page fragment ratio of each application.

Second, the allocated *I/O width* of each far memory path is essential to resource efficiency. We analyze the latency changes that occur with different I/O width allocations on disks for graph processing and AI inference workloads. Some tasks achieve lower end-to-end latency when adding I/O width assignments, while others do not, as depicted in Figure 5-(b). The reason is that the task maintains different distributions of sequential and random I/O access requests. As indicated in Figure 11, applications with larger maximum sizes of sequentially accessed data generally benefit from larger I/O bandwidth, while applications with predominantly random-access data segments may experience performance drops due to I/O amplification. Thus, we prioritize adding/reducing the bandwidth of applications with a more/less sequential data access ratio.

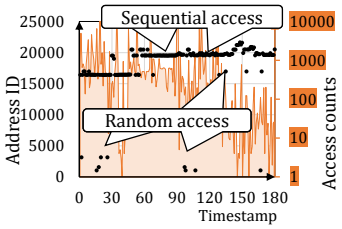


Fig. 11: Sequential and random accessed page behaviors.

Third, the *data distribution*, i.e. the proportion of frequently accessed data segments, in each program greatly affects overall application performance and memory efficiency. Usually, decreasing the proportion of local memory results in different performance degradation, which is relative to the far memory sensitiveness. Thus, keeping a proper rate of data in local memory can maintain the minimum acceptable latency. We estimate the minimum ratio of hot data by accumulating distinguished high-frequency data. In addition, local memory allocation strategies also consider the memory usage in the same socket and different sockets on NUMA architecture. In Figure 12, some tasks show little sensitivity to these strategies, while others are highly affected. NUMA memory nodes can be selected for insensitive applications when facing issues of local same-socket memory shortage.

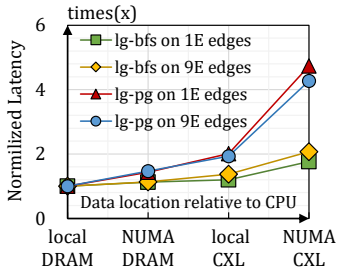


Fig. 12: Impacts of data distribution on NUMA architecture.

As mentioned above, analyzing various page data allows one to gain a deeper understanding of application features. We derive page access characteristics from a page trace table (shown in Figure 9-(a)). We collect data fragment ratio, page load/store ratio, and hot data segment ratio from the trace data. We then allocate and adjust the multi-dimensional parameters, including the data granularity, I/O width, and data distribution.

2) FM Parameter Adjustment: Parameter configuration shares similar data feature extraction ideas but different implementation methods on far memory backends, as shown in Figure 9-(b). We optimize the key tunable parameters on different far memory paths, as shown in Table III.

The *data granularity* can be flexibly modified by altering the size of data units transferred via RDMA (i.e. *chunk size*) or by amalgamating data blocks on SSD (i.e. *page size*). Typically, the page size is 4KB in common-used OS. The OS can also support larger page sizes, like 2MB huge pages controlled by transparent huge pages (THP). We adaptively turn on THP to large-granularity programs to achieve higher performance. There is a trade-off of configuring larger pages since huge pages can reduce the number of TLB misses but cause extra page reclaim overhead. We selectively enable THP by utilizing *khugepaged* to tailor page size and huge page allocation. In this work, the average page size can vary from 4KB to 2MB by controlling the amounts of to-be-allocated huge pages.

The *I/O width* is dynamically modifiable by assigning *CPU cores* related to I/O channels on SSD and *network channels* when transferring data through RDMA. We analyze the I/O width requirement by analyzing the ratio of data segments with continuous load operations. This information is obtained from the counts of load and store page operations. For storage-based FM, we adjust the I/O width by setting the block size or allocating multi-threaded I/O channels on SSDs. For network-based FM, we limit the network bandwidth by changing the number of bound event queues, such as adding multiple transfer queues on RDMA. We further enhance RDMA-based

Parameter	Offline Conf.	Online Conf.	Scale
Total CPU core	Yes	No	$\leq$ Total CPU cores
Local memory size	Yes	No	$\leq$ Server memory size
NUMA memory	Yes	No	Different NUMA nodes
Far memory ratio	Yes	Yes	$0 \sim 0.9$
Page size	Yes	Yes	4K $\sim$ 2M on average
Network channel	Yes	Yes	$\leq$ Total I/O channels

TABLE III: The tunable FM parameters in our system.

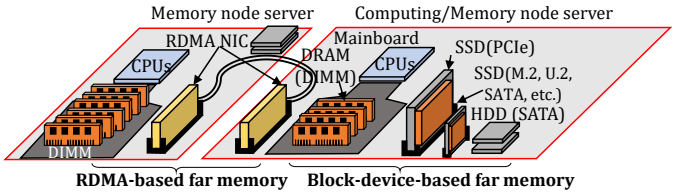


Fig. 13: Our physical testbed with RDMA-based far memory devices and block-device-based far memory devices.

far memory efficiency by enabling shared receive queues.

The *data distribution* is adaptively managed by setting *far memory ratio* and *NUMA memory nodes*. We limit the overall local memory usage to trigger page swap under certain data offloading ratios. We use the data segments ratio and the dataset size to estimate the corresponding minimum local memory size. Applications can be assigned a larger local memory size than the minimal size to meet specific Service Level Objectives (SLOs). Furthermore, we also consider different NUMA control strategies: We bind the CPU and memory on the same NUMA node to keep locality while on the different NUMA node for load balance.

Note that in recent works [25], [31], [49], the advanced CXL memory is considered mainly as a specific NUMA node, as the physical CXL memory device haven't occur yet. xDM can support the usage of new-generation cache-coherent memory devices and interconnects. As stated in Section IV.B., we utilize NUMA control strategies to manage the cache-coherent local memory allocations. Taking CXL as an example, the PCIe-based CXL memory can act as a local NUMA node with large memory space and no CPU, or one of the far memory backends.

V. Evaluation Methodology and Results

Our evaluation answers these questions:

- What is the overall performance and efficiency of our system on real-world workloads? (Section V-B)
- Does this system bring significant overhead to the original environment? (Section V-C)
- How scalable is xDM and what are the benefits of deploying it in a large cluster? (Section V-D)

A. Experimental Setup

1) System Testbed: We build a system prototype of xDM. As shown in Figure 13, it includes both SSD-based FM and RDMA-based FM systems on physical machines. Each server node is provided with two 10-core Xeon CPUs, (larger than) 64 GB of DRAM memory (134 GB/s), 1TB SSD (3.8 GB/s), 6 TB of HDD (0.4 GB/s), and Mellanox ConnectX-5 RDMA NICs supporting dual-port 10 GB/s bandwidth. The RDMA driver is version 5.4.0 of the OFED kernel with RoCE protocol. We adopt SR-IOV (Single Root I/O Virtualization) that provides direct virtualization of physical RDMA devices. We use QEMU [50] in KVM [51] to manage the created virtual far memory paths. We use *Cgroup* and *namespace* to control the CPU core, memory usage, network channel, and swap space for each process. Typically, the evaluated overall latency comprises kernel-level time (`sys time`) and user-level time (`cpu time`). We measure the overall task runtime using `time`, `maps`, etc., collect real-time memory usage using Intel *VTune*, test memory access latency using *PMU*, and record the page fault number with Linux *perf*.

Algorithm 1: Multi-backend FM System Workflow.

Input: Application set: A, online VM set: OVs, free VM set: FVs
Result: All applications have been efficiently dispatched

```
for a in A do
    f_a = page_feature_extraction(a)
    b_a = backend_selection(f_a, system_pressure)
    List p_a = parameter_optimization(f_a)
    for Online_VM in OVs do
        if Online_VM.backend = b_a AND Online_VM.accept(a)
          then
            dispatch a → Online_VM
            Online_VM.OptParameters(p_a)
            break
    if no available online VM then
        for Free_VM in FVs do
            if Free_VM.backend = b_a AND Free_VM.accept(a)
              then
                dispatch a → Free_VM
                Free_VM.OptParameters( p_a )
                break
    else if no available idle VM with b_a then
        Free_VM← SelectVM(FVs)
        Free_VM.SwitchBackend(b_a)
        Free_VM.OptParameters( p_a )
        dispatch a → Free_VM
    else if no available idle VM AND host resource is available
      then
        Free_VM← CreateVM(b_a, system_pressure)
        Free_VM.OptParameters(p_a)
        dispatch a → Free_VM
        add Free_VM→ Vs
```

2) System Workflow: For easy deployment of our system, we outline the workflow of xDM's core part,detailed in Algorithm 1. *i). Far memory initialization:* We configure the *memory.high* file in *Cgroup* to limit the usage of local memory and trigger data swap.We prepare the FM environment via automated configuration shells. For storage-based FM, we use swap files on different block devices and call for file read/write APIs. For network-based FM, we use RDMA event-driven queue to transfer pages with RDMA one-side read and write operations. *ii). Offline preparation:* We track the page behaviors of applications and prepare the offline fused information. Utilizing the offline information, we generate the FM path preference of each application and the parameter adjustment shells on corresponding far memory paths. *iii). VM allocation and warm start:* We set up virtual machines

Related works	Far memory	Max BW	FM size
Linux swap [42]	disk	2 GB/s	2T
TMO [37]	SSD	7.9 GB/s	1T
Fastswap [27]	RDMA	10 GB/s	256G
XMemPod [40]	DRAM or RDMA	10 GB/s	1T
xDM-SSD	multiple SSD	32 GB/s	1T
xDM-RDMA	multiple RDMA	32 GB/s	256G
xDM-Hetero	RDMA and SSD	32 GB/s	1.3T

TABLE IV: Baseline configurations.

Type	Abbr.	Algorithm Description	Max Mem.
HPC workloads	stream	Stream [52] for memory bandwidth	4G
	lpk	Linpack [53] for floating-point computing	4G
	kmeans	K-means clustering on sklearn [48]	4G
	sort	Quicksort [53] on c++ std	8G
	sp-pg	Page rank on Spark [2]	10G
Graph workloads	gg-pre	Graph preprocess on GridGraph [47]	16G
	gg-bfs	Breadth-first search on GridGraph [47]	16G
	lg-bfs	Breadth-first search on Ligra [1]	16G
	lg-bc	Betweenness centrality [1]	16G
	lg-comp	Connected components [1]	16G
	lg-mis	Multiple importance sampling [1]	16G
AI workloads	tf-incep	Resnet inception on Tensorflow [45]	1G
	tf-infer	Resnet inference on Tensorflow [45]	1G
	tf-tc	CNN inference on text classification [46]	10G
	bert	Inference on Bert [7]	1.5G
	clip	Inference on Clip [6]	1.7G
	chat-int	Inference on ChatGLM [5] (int4)	14G

TABLE V: Evaluated workloads.

(VMs) with appropriate CPU cores, memory size, storage, and network channels via the hypervisor. We prioritize using idle VMs with the required FM paths and online VMs to avoid a cold start. *iv). FM path selection and switching:* We get the priority list from page feature extraction and then select the appropriate FM path according to the list. If there is no available idle VM with the required FM path, we switch FM paths and adjust optimized parameters accordingly. *v). FM parameter configuring:* The parameter optimization always happens behind FM path selection and before each application starts. For each FM path, we use the optimal configurations including the page size, minimum CPU core, I/O channel number, local memory ratio, NUMA core binding, etc.

3) Baselines: We compare our system with state-of-the-art far memory systems including *TMO*, *Fastswap* and *Xmempod*. We deploy Linux swap and Fastswap as the basic far memory runtime emvironments. We also realize and test the far memory management strategy of these works on the required far memory devices. We list the configurations of the evaluated baselines in Table IV. Additionally, xDM system is tested on various multiple backends, including on multiple SSD devices (*xDM-SSD*), multiple RDMA devices (*xDM-RDMA*), and heterogeneous environments (*xDM-Hetero*) that integrate both SSD and RDMA devices.

4) Workloads: We use a range of real-world applications prevalent in today's data centers, as outlined in Table V. Except from regular computing applications from standard benchmarks, such as Linpack, Stream, and Spark, we also consider graph processing algorithms like graph traversal, page rank, subgraph searching, and simple sampling on popular high-performance graph frameworks including Ligra and GridGraph. In addition, we also test commonly used inference workloads on foundation and fine-tuned AI models including models on ResNet, Bert, Clip, Chatglm, etc. In our detailed performance analysis, we categorize the workloads into two types based on their runtime characteristics in relation to memory, computation, and I/O: *swap-sensitive*, and *swap-friendly*. This classification helps us precisely evaluate the impact of our system on different types of workloads.

B. Overall Benefits

Our system shows significant swap performance speedup and memory efficiency improvement on physical machines as testbeds and real-world applications.

1) System Swap Speedup: xDM can significantly improve the swap performance with minor overhead compared with baselines. The results are detailed in Table VI. We run workloads from table V at an appropriate local memory ratio. The swap performance improvement is then assessed using kernel-level `sys time`. The results show that our design achieves up to 2.43× speedup over Fastswap on the DRAM backend, a 2.16× speedup over Linux swap on the SSD backend, and a 3.89× speedup over Fastswap on the RDMA backend. In general, workloads experience higher speedup on the RDMA and DRAM backend, compared with the SSD backend. In a few cases of running swap-sensitive applications such as sort and clip, xDM on DRAM and SSD backends is suboptimal due to irregular memory access. In most cases, our work shows positive speedup except for some special conditions, like *kmeans* on SSD and *clip* on DRAM backend. The maximum average speedup on each workload is 2.32× on the *chat-int* workload, which inspires us to optimize the memory efficiency of AI inference tasks in our future work.

2) System Data Throughput: To assess data throughput enhancement, we measured the amount of data swapped per second for each workload. We use the results of TMO on a single SSD backend as the normalization basis and show the data throughput of our designs compared with baselines in Figure 14. Basically, the disk-based Linux swap acts much worse than SSD-based baselines due to the I/O operation difference. The *stream* and *means* workloads are memory-intensive with no extra I/O operations, thus they maintain nearly the same throughput on the disk-based and SSD-based FM paths. Our system consistently achieves higher data throughput than our baselines. It shows an improvement of up to 2.63× on multiple SSD-based FM paths, 2.82× on configurations with multiple RDMA-based FM paths, and 2.76× on heterogeneous FM paths, compared with TMO. For applications of *stream, tf-incep, and lg-comp*, xDM on heterogeneous FM paths (xDM-hetero) has better speed up compared with xDM on multiple RDMAs (xDM-RDMA), which infers heterogeneous devices can bring higher efficiency. The above findings underscore that incorporating multiple far memory paths leads to improved data throughput, thereby enhancing the bandwidth efficiency of far memory access.

Evalauted Workload	stream	lpk	kmeans	sort	s-pg	gg-pre	gg-bfs	lg-bfs	lg-bc	lg-comp	lg-mis	tf-infer	tf-incep	clip	tf-tc	chat-int	bert
Swap Feature	S	S	S	S	S	F	S	F	F	F	F	F	F	S	F	F	S
Sp. on DRAM	**1.32**×	1.18×	**1.64**×	1.05×	**1.44**×	**2.24**×	**1.29**×	2.00×	2.16×	**2.43**×	**2.17**×	1.88×	1.72×	0.82×	1.28×	1.15×	1.03×
Sp. on SSD	1.01×	**1.52**×	0.88×	0.86×	1.01×	1.02×	1.18×	1.40×	1.42×	1.52×	1.36×	1.51×	1.34×	0.91×	2.16×	1.92×	**1.75**×
Sp. on RDMA	1.25×	1.09×	1.40×	**1.40**×	1.37×	2.06×	1.19×	**2.24**×	**2.26**×	2.22×	2.07×	**2.70**×	**2.53**×	**2.46**×	**2.55**×	**3.89**×	1.10×
Average Speedup	1.19×	1.26×	1.31×	1.11×	1.28×	1.77×	1.22×	1.88×	1.95×	2.05×	1.86×	2.03×	1.86×	1.40×	2.00×	**2.32**×	1.29×

TABLE VI: The swap performance speedup (Sp.) of our xDM compared with baselines on the same backends. The baselines include Linux swap [42] on SSD backend, Fastswap [27] on RDMA and DRAM backends separately. We categorize application swap features into two types: swap-sensitive (S, average Sp.$\leq 1.5\times$) and swap-friendly (F, average Sp.$\geq 1.5\times$).

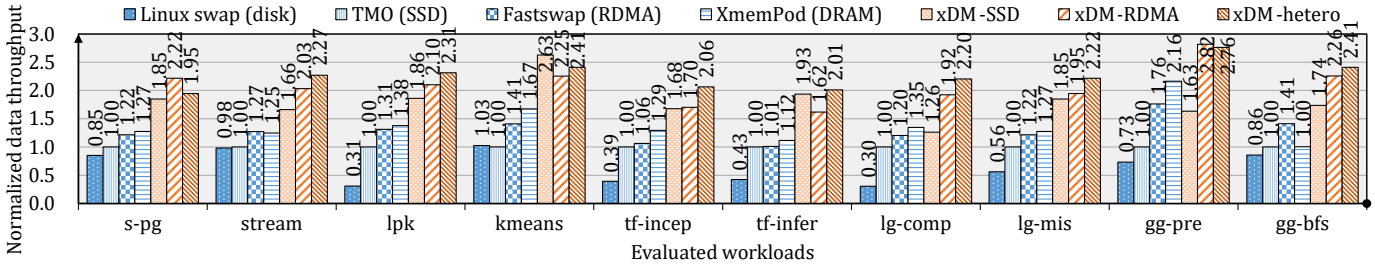


Fig. 14: Our design shows larger data throughput than baselines on evaluated workloads with different far memory backends.

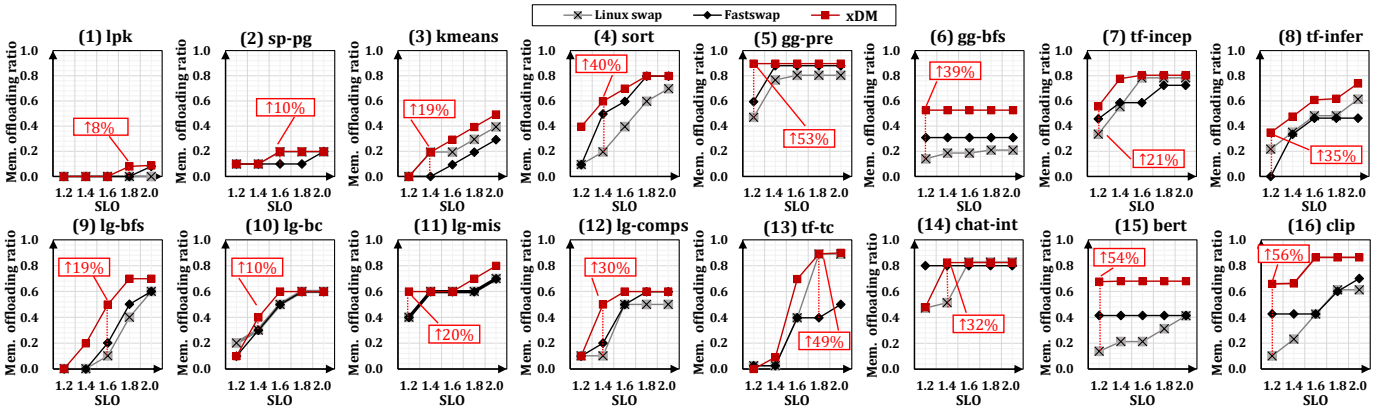


Fig. 15: Our design shows larger memory offloading ratios than baselines on evaluated workloads under different SLOs.

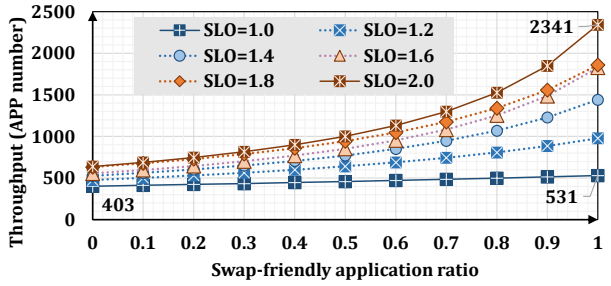


Fig. 16: The overall task throughput under different SLOs.

TABLE VII: PCIe bandwidth of xDM on different backends.

Devices in xDM	PCIe configuration	Device R/W bandwidth (Max, GB/s)	PCIe BW is full?
RDMA backend	Speed 8GT/s, Width x16	10.72 GB/s	Full
SSD backend	Speed 8GT/s, Width x8	8.95 GB/s	Full

Furthermore, we test PCIe bandwidth utilization with read and write bandwidth to prove that xDM can improve as much PCIe bandwidth as possible. The results show that xDM can achieve almost upper-bound bandwidth of each device and achieve full PCIe bandwidth when using RDMA and SSD backends, as shown in Table VII. This proves that the usage of multiple far memory backends in xDM can easily saturate the provoded PCIe bandwidth.

3) Memory Pressure Reduction: We examined the memory efficiency of our system.We investigated the local memory usage reduction and execution latency under various SLOs (permissible latency increase over the original workload latency). We then compare the results with the Linux swap and Fastswap baselines. As illustrated in Figure 15, a larger memory offloading ratio means better memory efficiency. Consistently, our system demonstrates superior memory offloading ratios compared to the baseline under identical SLO constraints. As the SLO rises, the memory offloading ratio increases, with up to 54% local memory pressure reduction. Notably, swap-friendly workloads such as *clip*, *gg-pre*, *tf-tc*, and *bert* benefit significantly from our approach. We also find that a larger SLO does not always bring memory efficiency

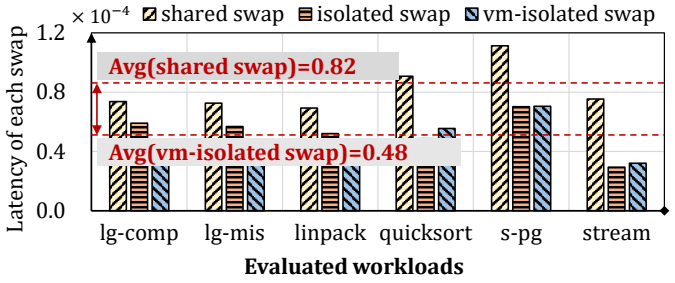


Fig. 17: The latency of each swap operation on different swap isolation methods (Our xDM uses *vm-isolated swap*).

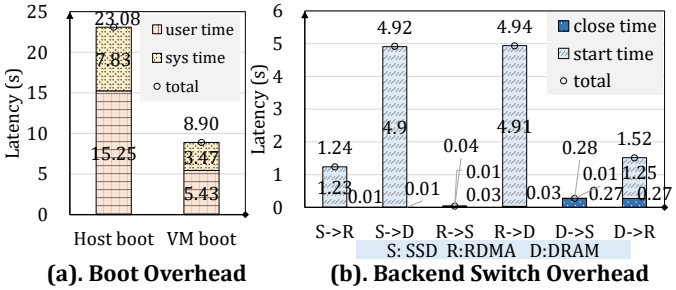


Fig. 18: Virtualization and backend switching overhead.

improvement. This is because performance degradation tends to be magnified when local memory falls below the hot data size, and in this case even increasing the SLO value will not trigger memory offloading. In most cases, appropriate SLO settings, like an SLO of 1.4 in *sort* and *tf-incep*, can offer a balance between enhanced user experience and increased memory efficiency.

4) System Task Throughput: System serving throughput matters in data centers. We analyzed task throughput across different program distributions, adhering to specified SLO limits. We varied the proportion of swap-friendly programs from 0 to 1 and observed the corresponding changes in task throughput. As illustrated in Figure 16, larger SLOs lead to more pronounced throughput improvements, reaching up to 5.6× compared to the baseline without far memory. A noteworthy finding is that the highest efficiency is not always achieved with the largest SLO. For instance, SLOs of 1.6 and 1.8 yield similar throughput, making an SLO of 1.6 a more optimal choice over 1.8. Furthermore, the presence of a larger proportion of swap-friendly applications correlates with increased system throughput, underscoring the impact of program type on overall system performance.

5) Swap Isolation: Swap isolation is important, as it reduces data contention among different workloads. We evaluate the effectiveness of swap isolation in our system compared with other isolation methods. We co-locate each workload with a specific task and measure the average latency of each swap operation on *shared swap*, *isolated swap* and *vm-isolated swap* methods, as shown in Figure 17. The evaluated *shared swap* method is the traditional shared-LRU swap mechanisms in Baseline *Linux swap* and *Fastswap*. It shows the worst swap latency performance since applications often face obvious data

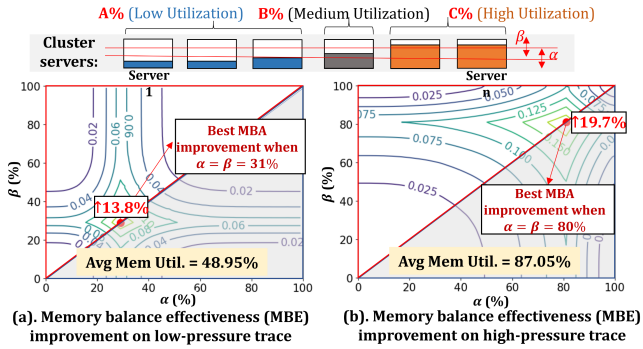


Fig. 19: MBE improvement on Alibaba 2017 and 2018 traces.

competition when sharing swap channels. The *isolated swap* uses separated isolated swap channels with no colocated tasks in Canvas [30], which has a significant swap performance increase. Our method utilizes VM to isolate the swap channels (*vm-isolated swap*), which shows close performance with the *isolated swap* method. Compared with *shared swap* method in the traditional far memory works, our method can gain an average 1.7× speedup on swap performance.

C. System Overhead

It is important to minimize design overhead. We compare the system backend switching overhead of the traditional method and xDM. Our design incurs significantly lower operating system boot overhead on VMs. In contrast, existing works [27], [40] typically require a physical shutdown and system reboot. The detailed latency at the user level and system level is as illustrated in Figure 18-(a). By utilizing VM reboot rather than host boot, xDM performs 2.6× faster than related works.

We also assess the specific backend switching overhead, as depicted in Figure 18-(b). We provide a detailed analysis of the overhead for each switching scenario between SSD, DRAM, and RDMA. The overhead primarily involves the time taken to shut down and start up swap backend modules. Notably, all backend switches are completed in less than 5 seconds, a duration that is generally acceptable for many long-running, data-intensive tasks. Specifically, backend switching on both SSD and RDMA FM paths shows a quite short duration, which benefits from our configuration of the prepared swap backends. The startup of the DRAM backend is the most time-intensive, primarily due to the delay associated with memory allocation on the host machine.

D. System Scalability

Finally, we evaluate the scalability of our design. We use public cluster traces of large-scale production systems [54] to evaluate the effectiveness of our system. We design a metric called memory balance effectiveness MBE), which reflects the increase in memory utilization on underutilized nodes and the corresponding decrease on nodes with high memory pressure. The MBE is calculated as $MBE = C\% \times (\bar{c} - \beta) - A\% \times (\bar{a} - \alpha)$. In clusters, we assume that $A\%, B\%, C\%$ is the percentage of servers with *low*, *medium*

and *high* memory utilization. When balancing memory, the $A\%$ servers can share memory resources with $C\%$ servers while the $B\%$ servers perform no memory adaption. We set variables α and β ($\beta \geq \alpha$) as the utilization threshold between *low*, *medium* and *high* utilization servers. $\bar{a}$ is the average utilization of $A\%$ servers and $\bar{c}$ is the average utilization of $C\%$ servers. In cluster traces, $A\%, B\%, C\%, \bar{a}, \bar{c}$ can be calculated given the α and β.

Utilizing multi-path far memory environments can balance resource pressure among busy and idle servers without adding new server nodes. Figure 19 shows the contour percentage of memory balance effectiveness improvement of Alibaba cloud trace in 2017 and 2018. Specifically, (a) is a low-pressure trace with 48.95% average memory utilization and (b) is a high-pressure trace with 87.05% average memory utilization. As Figure 19 shows, we can achieve a memory balance effectiveness improvement by up to 13.8% when $\alpha = \beta = 31\%$ if the memory pressure is low and up to 19.7% when $\alpha = \beta = 80\%$ on the trace of high memory pressure. The results show that we have a better effectiveness improvement on clusters with a high average memory usage rate. It demonstrates robust scalability under conditions of increased memory resource pressure in the data center.

VI. Related Work

A. Disaggregated Memory Runtime

Disaggregated architecture [20], [22], [55]–[57] has attached great attention in recent years. Existing works build large memory pools [58] with various memory devices including CXL memory devices [25], [31], [59]–[62], non-volatile persistent memory [24], [63]–[65] and solid-state drive (SSD) [37], [42] on local servers as *intra-node* FM. They also share memory resources among servers through network channels including RDMA network cards [27], [28], [66], OpenCAPI-based fabrics [22], [67], [68], smart-NICs [43], [69], and smart network switches [70], to access *inter-node* FM on remote servers. In the early years, works utilize user-defined remote memory access APIs to improve the performance of far memory access [71]–[75]. These works help to utilize and manage different FM resources in the single FM-path environment. Complementarily, our work focuses on the problem of managing a new multi-backend disaggregated memory architecture, enabling elastic and flexible expansion of far memory resources.

B. Task Management in Far Memory Environment

Existing disaggregated memory management systems typically allocate resources to applications by detecting current resource pressure [42], [76]. Based on the existing memory resource scheduling strategies [77]–[79], related studies [24], [37], [40] have introduced adaptive allocation strategies that utilize heterogeneous backends to accommodate varying resource pressures effectively. Previous works [27], [37], [38], [44] have examined application sensitivity on far memory platforms. Based on this analysis, they adjust the far memory ratio and memory size allocated to tasks to meet Quality of Service (QoS) requirements. Some works take the size of the transferred data chunk is considered to accelerate data transferring in recent works [43], [44], [80]. Canvas [30] builds separated isolated swap channels on RDMA far memory to isolate tasks. However, they lack detailed page information analysis to fully extract the application behavior. Our work outperforms previous studies by analyzing page data in a fine-grained way and fine-tuning various FM parameters for higher performance.

C. Data Placement on Hybrid Memory

In hybrid memory architecture, data on far memory or slow memory is often considered as an exclusive relationship with the data on local or fast memory, managed by data swapping [24], [81]–[84]. There are some works concentrated on swap mechanism design at different system levels. VSwapper [85] builds data swap channels across virtual machines. Hybridswap [86] optimizes the data swap between VMs and host disks. XMemPod [40] enables virtual machines to access RDMA-based remote memory on the host with a shared swap channel on the host. Hybrid^2 [87] handles near and far memory access between SRAM and DRAMs. In GPU-involved heterogeneous environments, related works utilize unified memory to place data into CPU memory and SSD [19], [84], [88]. However, these works swap data in a hierarchical way, with data copies on VM, host, and FM. Our work supports host-bypass FM access in each VM, reducing data duplication for enhanced efficiency.

VII. Conclusion

In this work, we design and implement xDM, a novel multi-backend far memory system with high bandwidth utilization and application performance. By turning the conventional swap mechanism into a switchable data swap module, we successfully realize simultaneous multi-path FM access. In addition, based on a rich synthesis of application page data, we tailor the far memory path configurations to the needs of various applications. Our work shows up to $3.9\times$ data swap performance speedup, $2.8\times$ data throughput increase, and $5.1\times$ data center task serving throughput improvement compared with state-of-the-art works. Our design provides a flexible solution to scale out far memory access paths and an efficient way to manage them on monolithic servers. We expect that our design can improve the execution performance and memory usage effectiveness of memory-hungry tasks in cloud and near-edge micro data centers.

Acknowledgement

We sincerely thank all the anonymous reviewers for their valuable comments. This work is supported by the National Key R&D Program of China (No. 2022YFB4501702), and the National Natural Science Foundation of China (No. 62122053). The corresponding author is Chao Li.

REFERENCES

[1] Julian Shun and Guy E Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*, 2013.

[2] Spark. https://spark.apache.org/.

[3] Pengyu Wang, Chao Li, et al. Skywalker: Efficient alias-method-based graph sampling and random walk on gpus. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2021.

[4] Junyi Mei, Li Chao Sun, Shixuan, Cheng Xu, Yibo Liu, Wang Jing, Xiaofeng Zhao Cheng, Hou, Minyi Guo, Bingsheng He, and Cong Xiaoliang. Flowwalker: A memory-efficient and high-performance gpu-based dynamic graph random walk framework. 2024.

[5] Chatglm from thudm. https://github.com/THUDM/ChatGLM-6B.

[6] Clip from openai. https://github.com/openai/CLIP.

[7] Bert from google. https://github.com/google-research/bert.

[8] Yifei Pu, Chi Wang, Xiaofeng Hou, Cheng Xu, Liu Jiacheng, Jing Wang, Minyi Guo, and Chao Li. M2sn: Adaptive and dynamic multi-modal shortcut network architecture for latency-aware applications. In *Proc. the IEEE International Conference on Multimedia and Expo (ICME)*, 2024.

[9] Amazon. Iaas. https://aws.amazon.com/cn/what-is/iaas/, 2024.

[10] Amazon. Amazon s3. https://aws.amazon.com/cn/s3/, 2019.

[11] Huawei Cloud. Huawei cloud object storage service (obs). https://www.huaweicloud.com/product/obs.html, 2019.

[12] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. When cloud storage meets rdma. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021.

[13] Nathaniel Bleier, Muhammad Husnain Mubarik, Gary R Swenson, and Rakesh Kumar. Space microdatacenters. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Association for Computing Machinery, 2023.

[14] Daliang Xu, Mengwei Xu, Chiheng Lou, Li Zhang, Gang Huang, Xin Jin, and Xuanzhe Liu. Socflow: Efficient and scalable dnn training on soc-clustered edge servers. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Association for Computing Machinery, 2024.

[15] Zijun Li, Linsong Guo, Jiagan Cheng, Quan Chen, Bingsheng He, and Minyi Guo. The serverless computing survey: A technical primer for design architecture. *ACM Computing Survey*, 2022.

[16] Lu Zhang, Weiqi Feng, et al. Tapping into nfv environment for opportunistic serverless edge function deployment. In *IEEE Transactions on Computers (TC)*, 2021.

[17] Lu Zhang, Weiqi Feng, Chao Li, Xiaofeng Hou, Pengyu Wang, Jing Wang, and Minyi Guo. Tapping into nfv environment for opportunistic serverless edge function deployment. *IEEE Transactions on Computers (TC)*, 2021.

[18] Du Liu, Jing Wang, Xinkai Wang, Chao Li, Zhang Lu, Xiaofeng Hou, Xiaoxiang Shi, and Minyi Guo. Cocg: Fine-grained cloud game co-location on heterogeneous platform. In *ACM/IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2024.

[19] Taolei Wang, Jing Wang, Chao Li, Cheng Xu, and Xiaofeng Hou. Cocg: Fine-grained cloud game co-location on heterogeneous platform. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2024.

[20] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. Legoos: A disseminated, distributed os for hardware resource disaggregation. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[21] Qizhen Zhang, Xinyi Chen, Sidharth Sankhe, Zhilei Zheng, Ke Zhong, Sebastian Angel, Ang Chen, Vincent Liu, and Boon Thau Loo. Optimizing data-intensive systems in disaggregated data centers with teleport. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*, 2022.

[22] Christian Pinto, Dimitris Syrivelis, Michele Gazzetti, Panos Koutsovasilis, Andrea Reale, Kostas Katrinis, and H Peter Hofstee. Thymesisflow: a software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.

[23] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter. In *European Conference on Computer Systems (EuroSys)*, 2018.

[24] Jacob Wahlgren, Gabin Schieffer, Maya Gokhale, and Ivy Peng. A quantitative approach for adopting disaggregated memory in hpc systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2023.

[25] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. Direct access, High-Performance memory disaggregation with DirectCXL. In *2022 USENIX Annual Technical Conference (USENIX ATC)*, 2022.

[26] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K Aguilera, and Adam Belay. Aifm: High-performance, application-integrated far memory. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[27] 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 *European Conference on Computer Systems (EuroSys)*, 2020.

[28] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with infiniswap. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[29] Irene Zhang, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson, Omar S Navarro Leija, Ashlie Martinez, Jing Liu, Anna Kornfeld Simpson, Sujay Jayakar, et al. The demikernel datapath os architecture for microsecond-scale datacenter systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (ASPLOS)*, 2021.

[30] Chenxi Wang, Yifan Qiao, Haoran Ma, Shi Liu, Wenguang Chen, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. Canvas: Isolated and adaptive swapping for Multi-Applications on remote memory. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

[31] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. Pond: Cxl-based memory pooling systems for cloud platforms. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[32] Xingda Wei, Rongxin Cheng, Yuhan Yang, Rong Chen, and Haibo Chen. Characterizing off-path SmartNIC for accelerating distributed systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023.

[33] Infiniband architecture. https://developer.nvidia.com/networking.

[34] Samsung solid state drives. https://www.samsung.com/us/computing/memory-storage/solid-state-drives.

[35] Compute express link. https://www.computeexpresslink.org/.

[36] Pcie 6.0 is coming. https://www.theverge.com/2022/1/12/22879732/pcie-6-0-final-specification-bandwidth-speeds.

[37] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, et al. Tmo: transparent memory offloading in datacenters. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[38] Jing Wang, Chao Li, Junyi Mei, Hao He, Taolei Wang, Pengyu Wang, Lu Zhang, Minyi Guo, Hanqing Wu, Dongbai Chen, and Xiangwen Liu. Hyfarm: Task orchestration on hybrid far memory for high performance per bit. In *International Conference on Computer Design (ICCD)*. IEEE, 2022.

[39] Frontswap. https://www.kernel.org/doc/html/latest/mm/frontswap.html.

[40] Wenqi Cao and Ling Liu. Hierarchical orchestration of disaggregated memory. *IEEE Transactions on Computers (TC)*, 2020.

[41] Stella Bitchebe and Alain Tchana. Out of hypervisor (ooh): efficient dirty page tracking in userspace using hardware virtualization features. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2022.

[42] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, et al. Software-defined far memory in warehouse-scale computers. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[43] Irina Calciu, M Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[44] Jing Wang, Li Chao, Taolei Wang, Lu Zhang, Pengyu Wang, Junyi Mei, and Minyi Guo. Excavating the potential of graph workload on rdma-based far memory architecture. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2022.

[45] Tensorflow: An open source machine learning framework for everyone. https://github.com/tensorflow.

[46] Text classification from gaussic. https://github.com/gaussic/text-classification-cnn-rnn.

[47] Xiaowei Zhu, Wentao Han, and Wenguang Chen. Gridgraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning. In *USENIX Annual Technical Conference (USENIX ATC)*, 2015.

[48] Scikit-learn library. https://scikit-learn.org/stable/index.html.

[49] Haifeng Liu, Long Zheng, Yu Huang, Jingyi Zhou, Chaoqiang Liu, Runze Wang, Xiaofei Liaot, Hai Jinf, and Jingling Xue. Enabling efficient large recommendation model training with near cxl memory processing. In *ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024.

[50] Qemu. https://www.qemu.org/.

[51] Kvm, kernel virtual machine. https://www.linux-kvm.org/page/Main_Page, 2016.

[52] Stream: Sustainable memory bandwidth in high performance computers. https://www.cs.virginia.edu/stream/.

[53] C++ standard library headers. https://cppreference.com/.

[54] Alibaba cluster trace. https://github.com/alibaba/clusterdata.

[55] The future of data infrastructure: Composable disaggregated infrastructure. https://www.datacenterknowledge.com/industry-perspectives/future-data-infrastructure.

[56] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K Reinhardt, and Thomas F Wenisch. Disaggregated memory for expansion and sharing in blade servers. *ACM SIGARCH computer architecture news (CAN)*, 2009.

[57] Zhihan Guo, Xinyu Zeng, Kan Wu, Wuh-Chwen Hwang, Ziwei Ren, Xiangyao Yu, Mahesh Balakrishnan, and Philip A. Bernstein. Cornus: Atomic commit for a cloud dbms with storage disaggregation. *Proc. VLDB Endow.*, 2022.

[58] Matheus Ogleari, Ye Yu, Chen Qian, Ethan Miller, and Jishen Zhao. String figure: A scalable and elastic memory network architecture. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[59] Wenqin Huangfu, Krishna T Malladi, Andrew Chang, and Yuan Xie. Beacon: Scalable near-data-processing accelerators for genome analysis near memory pool with the cxl support. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022.

[60] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. CXL-ANNS: Software-Hardware collaborative memory disaggregation and computation for Billion-Scale approximate nearest neighbor search. In *USENIX Annual Technical Conference (USENIX ATC)*, 2023.

[61] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. Tpp: Transparent page placement for cxl-enabled tiered-memory. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[62] Shao-Peng Yang, Minjae Kim, Sanghyun Nam, Juhyung Park, Jin yong Choi, Eyee Hyun Nam, Eunji Lee, Sungjin Lee, and Bryan S. Kim. Overcoming the memory wall with cxl-enabled ssds. In *USENIX Annual Technical Conference (USENIX ATC)*, 2023.

[63] Shin-Yeh Tsai, Yizhou Shan, and Yiying Zhang. Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated key-value stores. In *USENIX Annual Technical Conference (USENIX ATC)*, 2020.

[64] Sekwon Lee, Soujanya Ponnapalli, Sharad Singhal, Marcos K Aguilera, Kimberly Keeton, and Vijay Chidambaram. Dinomo: an elastic, scalable, high-performance key-value store for disaggregated persistent memory. *Proceedings of the VLDB Endowment*, 2022.

[65] Ruihong Wang, Jianguo Wang, Prishita Kadam, M Tamer Özsu, and Walid G Aref. dlsm: An lsm-based index for memory disaggregation. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023.

[66] Ruihong Wang, Jianguo Wang, Stratos Idreos, M. Tamer Özsu, and Walid G. Aref. The case for distributed shared-memory databases with rdma-enabled memory disaggregation. *Proceedings of the VLDB Endowment (VLDB)*, 2022.

[67] Opencapi specification. https://opencapi.org/.

[68] Esha Choukse, Michael B Sullivan, Mike O'Connor, Mattan Erez, Jeff Pool, David Nellans, and Stephen W Keckler. Buddy compression: Enabling larger memory for deep learning and hpc workloads on gpus. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020.

[69] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. Clio: A hardware-software co-designed disaggregated memory system. *nternational Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[70] Seung-seob Lee, Yanpeng Yu, Yupeng Tang, Anurag Khandelwal, Lin Zhong, and Abhishek Bhattacharjee. Mind: In-network memory management for disaggregated data centers. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP)*, 2021.

[71] Qing Wang, Youyou Lu, and Jiwu Shu. Sherman: A write-optimized distributed b+tree index on disaggregated memory. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*, 2022.

[72] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. Farm: Fast remote memory. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[73] Marcos K Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novakovic, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, et al. Remote regions: a simple abstraction for remote memory. In *USENIX Annual Technical Conference (USENIX ATC)*, 2018.

[74] Shin-Yeh Tsai and Yiying Zhang. Lite kernel rdma support for datacenter applications. In *Symposium on Operating Systems Principles (SOSP)*, 2017.

[75] Philipp Fent, Alexander van Renen, Andreas Kipf, Viktor Leis, Thomas Neumann, and Alfons Kemper. Low-latency communication for fast dbms using rdma and shared memory. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020.

[76] Anurag Khandelwal, Yupeng Tang, Rachit Agarwal, Aditya Akella, and Ion Stoica. Jiffy: Elastic far-memory for stateful serverless analytics. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys)*, 2022.

[77] Malte Schwarzkopf, Andy Konwinski, et al. Omega: flexible, scalable schedulers for large compute clusters. In *ACM European Conference on Computer Systems (EuroSys)*, 2013.

[78] Eric Boutin, Jaliya Ekanayake, et al. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[79] Xinkai Wang, Hao He, et al. Not all resources are visible: Exploiting fragmented shadow resources in shared-state scheduler architecture. In *ACM Symposium on Cloud Computing (SoCC)*, 2023.

[80] Jing Wang, Chao Li, Taolei Wang, Lu Zhang, Pengyu Wang, Junyi Mei, and Minyi Guo. Fargraph+: Excavating the parallelism of graph computing workload on rdma-based far memory system. *Journal of Parallel and Distributed Computing (JPDC)*, 2023.

[81] Bruce Jacob, David Wang, and Spencer Ng. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010.

[82] Yaohui Wang, Ben Luo, and Yibin Shen. Efficient memory overcommitment for I/O passthrough enabled VMs via fine-grained page meta-data management. In *2023 USENIX Annual Technical Conference (USENIX ATC)*, 2023.

[83] Subramanya R Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. Data tiering in heterogeneous memory systems. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys)*, 2016.

[84] Pengyu Wang, Jing Wang, Chao Li, Jianzong Wang, Haojin Zhu, and Minyi Guo. Grus: Toward unified-memory-efficient high-performance graph processing on gpu. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2021.

[85] Nadav Amit, Dan Tsafrir, and Assaf Schuster. Vswapper: A memory swapper for virtualized environments. *ACM Sigplan Notices*, 2014.

[86] Pengfei Zhang, Xi Li, Rui Chu, and Huaimin Wang. Hybridswap: A scalable and synthetic framework for guest swapping on virtualization platform. In *IEEE Conference on Computer Communications (INFOCOMM)*, 2015.

[87] Evangelos Vasilakis, Vassilis Papaefstathiou, Pedro Trancoso, and Ioannis Sourdis. Hybrid2: Combining caching and migration in hybrid memory systems. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[88] Chuanming Shao, Jinyang Guo, Pengyu Wang, Jing Wang, Chao Li, and Minyi Guo. Oversubscribing gpu unified virtual memory: Implications and suggestions. In *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering (ICPE*, 2022.