Wang CX, Shan YZ, Zuo PF *et al*. Reinvent cloud software stacks for resource disaggregation. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 38(5): 949-969 Sept. 2023. DOI: 10.1007/s11390-023-3272-0

Reinvent Cloud Software Stacks for Resource Disaggregation

Chen-Xi Wang[1, 2] (王晨曦), *Member, CCF, ACM, IEEE*, Yi-Zhou Shan[3] (单一舟)
Peng-Fei Zuo[3] (左鹏飞), *Member, CCF*, and Hui-Min Cui[1, 2, *] (崔慧敏), *Member, CCF, IEEE*

[1] *Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*
[2] *University of the Chinese Academy of Sciences, Beijing 101408, China*
[3] *Huawei Cloud, Shenzhen 518129, China*

E-mail: wangchenxi@ict.ac.cn; shanyizhou@huawei.com; pengfei.zuo@huawei.com; cuihm@ict.ac.cn

Received April 3, 2023; accepted September 1, 2023.

Abstract Due to the unprecedented development of low-latency interconnect technology, building large-scale disaggregated architecture is drawing more and more attention from both industry and academia. Resource disaggregation is a new way to organize the hardware resources of datacenters, and has the potential to overcome the limitations, e.g., low resource utilization and low reliability, of conventional datacenters. However, the emerging disaggregated architecture brings severe performance and latency problems to the existing cloud systems. In this paper, we take memory disaggregation as an example to demonstrate the unique challenges that the disaggregated datacenter poses to the existing cloud software stacks, e.g., programming interface, language runtime, and operating system, and further discuss the possible ways to reinvent the cloud systems.

Keywords cloud computing, resource disaggregation, datacenter, program semantics

1 Introduction

The urgent demand of cloud providers to improve the reliability, availability, and serviceability (RAS) of datacenters promotes the rapid development of resource disaggregation technologies in the fields of hardware, systems, and applications[1–3]. In a disaggregated datacenter, hardware resources (e.g., CPU, memory, storage, and accelerators) are grouped as resource pools by connecting the individual servers over emerging I/O interconnects[1], as Fig.1 shows. On the one hand, from the perspective of an application, it can acquire resources from the large-scale cluster without considering the limitations of an individual server, increasing the amount of CPU and memory that can be accessed by orders of magnitude. On the other hand, from the perspective of the cluster, a server with any type of available resources can be utilized by the application without considering the availability of other types of resources, which significantly improves the resource utilization of the datacenter.

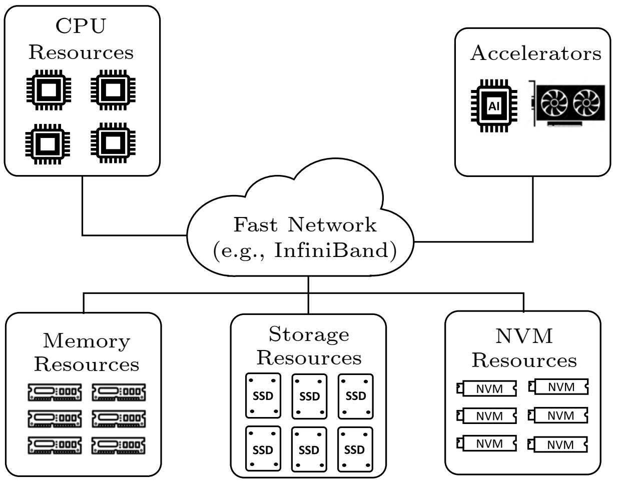

Fig.1. Overview of a disaggregated datacenter. All the servers are connected by the fast I/O interconnects and reorganized as different types of resource pools.

Due to the potentially huge benefits, both industry and academia devote much research to resource disaggregation. Due to the requirements of ultra-low communication latency, the CPU and memory resources

Survey

This work is supported by the National Key Research and Development Program of China under Grant No. 2022YFB4500400.

*Corresponding Author

are tightly coupled until recently. Hence, existing systems are all designed for conventional monolithic servers. As a result, the disaggregated architecture brings several severe challenges to the existing cloud software stacks due to its three characteristics— microsecond-scale latency, asymmetric performance, and costly synchronization. When running on disaggregated servers, applications can slow down 2x with 400x longer tail latency[4, 5].

New Challenges from Emerging I/O Interconnects. To keep pace with the scale of cloud applications, the I/O interconnect is progressing at an unprecedented speed to provide extremely high bandwidth and low latency. Recently, the I/O latency has been reduced to a microsecond-scale, the bandwidth can reach up to dozens of gigabytes per second[6], and the performance starts being comparable with the on-die interconnect. The fast network allows the decoupling of the tightly coupled CPU and memory resources. However, on the one hand, as Barroso *et al.* discussed in [7], such a new breed of microsecond-scale I/O devices brings various challenges to the existing cloud systems, which were originally designed to mitigate millisecond-scale events. As a result, the mismatch between the system design and the disaggregated architecture leads to significant performance overhead, and more details will be discussed in Subsection 3.1.

On the other hand, there is still a performance gap when comparing the I/O interconnects with the on-die interconnect, e.g., the peer-to-peer latency of Ultra Path Interconnect (UPI)[①] is only 40 ns, more than 10x shorter than using the most advanced I/O interconnects, InfiniBand[8, 9]. Hence, when applications run on a disaggregated datacenter connected by such emerging I/O interconnects, the hardware resources are heterogeneous with asymmetric performance. However, the conventional cloud software stacks, including data structure, language runtime, and operating system, are not aware of such a huge performance gap, leading to unexpected performance degradation and variation (Subsection 3.2).

At the same time, due to a lack of hardware support for cache coherency, inter-server synchronization (via I/O interconnects) is much slower than intra-server synchronization. For example, the atomic instructions, e.g., compare-and-exchange (CMPXCHG) and fetch-and-add (FAA), provided by the x86 CPU can finish around 6 nanoseconds–150 nanoseconds within a monolithic server[10]. But the synchronization between two disaggregated servers (connected by InfiniBand) can reach up to 4 microseconds by using a sequence of instructions[4], which is more than 20x slower. As a result, in order to get reasonable performance, the programmers have to partition the data carefully to limit the overhead of inter-server synchronization. The broader impacts of the costly synchronization will be further discussed in Subsection 3.3.

Semantics-Aware System for Disaggregation. The existing cloud system is built for monolithic servers under general-purpose design principles. The whole software stack, ranging from data structures, runtime to the operating system, is oblivious to the diversity of cloud applications and the characteristics of disaggregated architectures. As a result, the cloud system leads to a semantics gap between high-level applications and low-level hardware. There are two main reasons for this problem.

On the one hand, high-level programming languages (HHL) are widely used in the development of cloud applications, e.g., Hadoop, Spark, Cassandra, and Flink. The high-level languages provide a variety of runtime supports to make programming easier, letting the programmer focus on designing the program logic instead of elaborating on resource management. For example, most of the popular managed languages, e.g., Java, Python, and Scala, provide the data abstract of *Object* to enable an object-oriented programming model and automatically reclaim the dead space via the Garbage Collection (GC) mechanism to ease memory management and improve memory safety. However, applications need to pay performance tax for these runtime supports, and the program semantics are usually hidden from underlying systems. For example, the GC mechanism can lead to a 3x larger memory footprint[3]. As a result, when running on memory disaggregated servers, such cloud applications with large working sets tend to cause excessive accesses to the remote memory, degrading the performance significantly[3, 4, 9, 11, 12].

On the other hand, the operating system is an ill fit for managing disaggregated resources over the network. First, OS (operating system) components assume resources are local and communicate via shared memory or fast IPCs, regardless of kernel types. This mode, however, is infeasible over a slower network. Second, a traditional OS fate-shares with its hard-

[①]Interconnect that moves data faster and smarter. https://www.intel.com/content/www/us/en/silicon-innovations/6-pillars/interconnect.html, Sept. 2023.

ware. But in a disaggregated setting, failure is expected to be common. The OS must ensure a more flexible failure model. Finally, many OS mechanisms and policies are designed for slow devices, and one prominent example is that OS's decade-old paging subsystem is tailored for millisecond-scale disks. However, these mechanisms work poorly for microsecond devices such as RDMA.

We will discuss the new challenges in Section 3, and existing solutions in Section 4. Finally, we will discuss the potential research trends of disaggregated cloud systems in Section 5. To conclude, it is necessary to reinvent the cloud software stack to bridge the semantics-gap between cloud applications and disaggregated architectures.

2 Rise of Resource Disaggregation

In order to meet the ever-increasing parallelism of cloud applications, cloud providers proposed to build the warehouse-scale datacenter. At the same time, system developers build a series of distributed frameworks, e.g., MapReduce[13], to achieve very high scaling-out parallelism on the large-scale cluster. On the other hand, a new concept of datacenter, i.e., the computational Grid, is proposed to enable applications to scale up across multiple servers. The benefits are twofold: 1) the application can dynamically scale and acquire heterogeneous resources without considering the limitations of a single machine; 2) cloud providers can build dedicated resource nodes (servers) to reduce the complexity and cost of hardware.

2.1 Computational Grid

The computational Grid is a distributed framework proposed in the mid-1990s, developed for resource sharing during scientific collaborations[14]. The servers and scientific instruments are connected by networks, and all the hardware resources are organized as different kinds of resource pools, e.g., computing pool, memory pool, and storage pool. An application can be divided into a bunch of subtasks and scheduled to the appropriate servers. Different kinds of subtasks benefit from either the large-scale resource or the heterogeneous hardware, e.g., accelerators. Right now, the Grid② is still widely deployed in different areas, e.g., financial service, entertainment, and engineering.

Based on the computational Grid, Fan *et al.* proposed a reconfigurable architecture, DSAG (Dynamic Self-Organized Computer Architecture Based on Grid-components)[15, 16] in the early 2000s to fulfill the large-scale resource requirements of high-performance computing. The key idea is to disaggregate the hardware resources from an integrated monolithic server and organize the same type of hardware resources into a physical resource cluster (pool), as Fig.2 shows. When deploying an application, the underlying architecture will be dynamically reconfigured according to the program's characteristics. DSAG brings a new trend to building large-scale datacenters. First, the manufacturer can build dedicated architectures and systems for each kind of resource clusters to achieve a larger scale and higher management efficiency. For

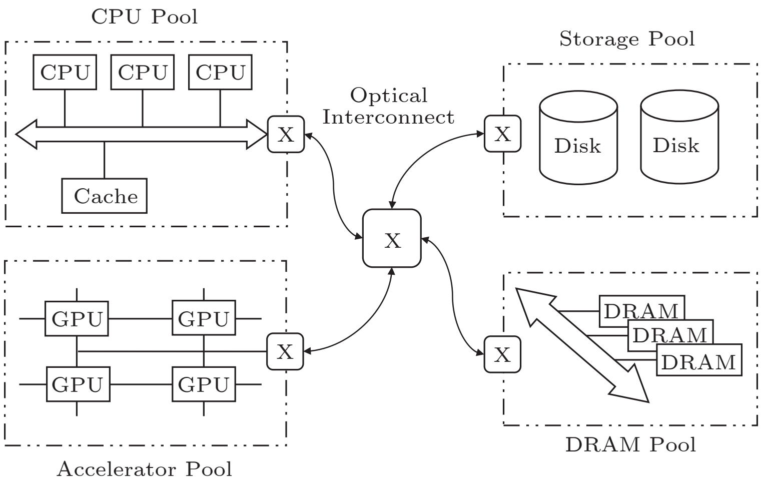

Fig.2. Dynamic Self-Organized Computer Architecture Based on Grid-Components (DSAG)[15, 16]. Each type of hardware is physically organized as a resource pool. A high-performance computer can be dynamically built from the resource pools according to the requirements of applications. The symbol X represents the optical switch.

②What is grid computing? https://aws.amazon.com/what-is/grid-computing, Sept. 2023.

example, the memory cluster is equipped with dedicated CPUs with fewer computing units and a larger memory management unit to break the Memory Wall[17]. Second, DSAG can provide specialized architectures for diverse applications with fine-grained resource provision via the ability of architecture reconfiguration. In addition, DSAG provides an exclusive and isolated hardware execution environment for applications to avoid the resource racing and conflicts of the legacy multi-tenant usage scenarios. DSAG establishes the trend of hardware resource disaggregation.

2.2 Resource Disaggregation

Fig.3 demonstrates the bandwidth development of different types of interconnects. With the unprecedented development of network technologies, the latency and bandwidth of I/O interconnects are narrowing the performance gap with the on-die interconnect. As a result, the emerging I/O interconnects allow us to disaggregate the tightly coupled CPU and memory resources and build the general-purpose disaggregated datacenter to support ubiquitous cloud computing workloads[1]. For example, the industry has built several prototypes, such as HP's The Machine③, Intel's Rack-Scale Design④ and the Firebox[18] from Berkeley. However, in order to make the disaggregated server more practical, some requirements should be satisfied, from hardware features to the programming model and OS design.

2.2.1 *Architecture Features*

Hardware components are reorganized and virtualized as resource pools. The disaggregated datacenter is built from separated monolithic servers, which

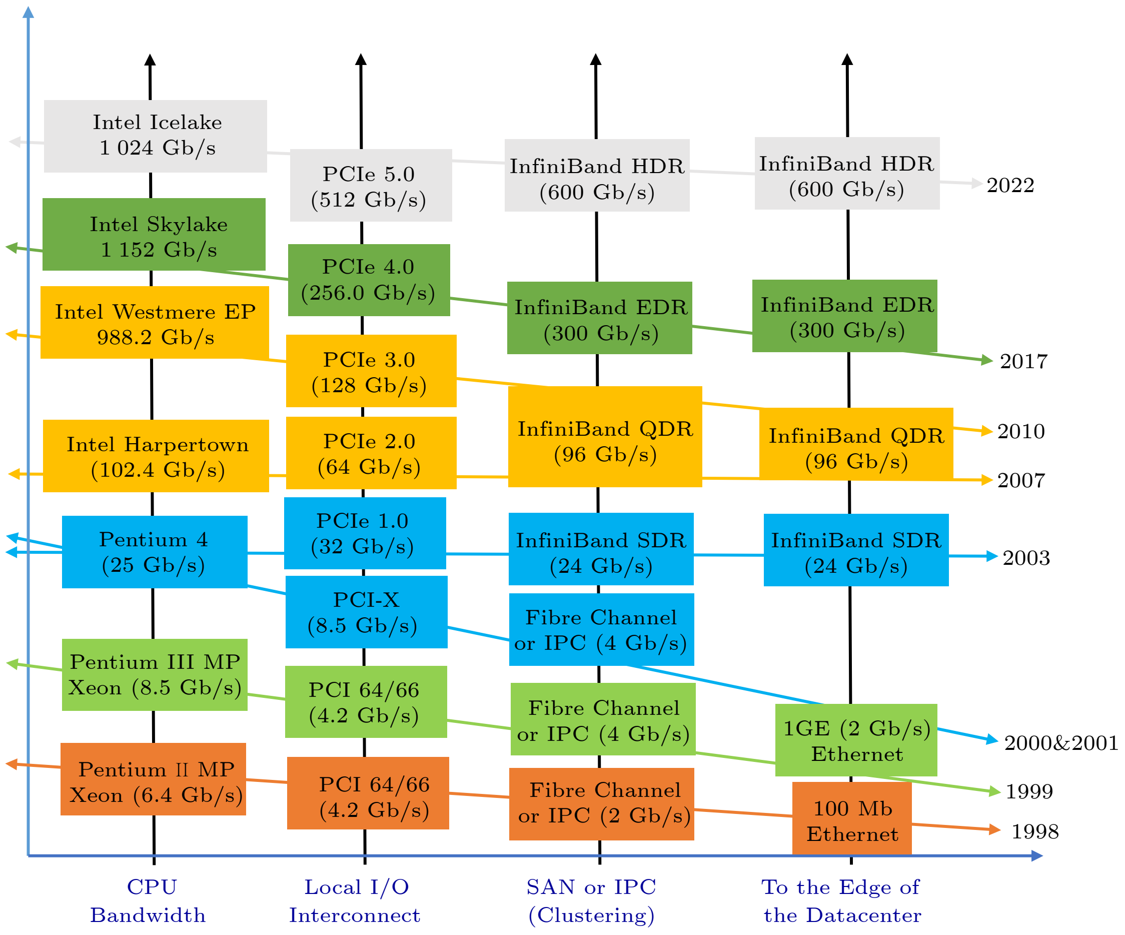

Fig.3. Development trend of I/O interconnects and on-die interconnects[19]. The data is collected and extended based on the white paper of Mellanox Technologies[20].

③The machine: A new kind of computer. https://www.hpl.hp.com/research/systems-research/themachine, Sept. 2023.

④Intel rack scale design. https://www.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design-overview.html, Sept. 2023.

are connected by the PCIe-based emerging I/O interconnects, as shown in Fig.4. For example, an application can request CPU resources from the first server and memory resources from three different servers. During the execution, the application can also offload the compute to where the data resides, e.g., the second and third servers in Fig.4. The benefits are twofold. First, PCIe has emerged as the dominant interconnect technology and is widely adopted in the server design. Hence, all the legacy servers supporting the compatible PCIe standards can be integrated into the disaggregated datacenter, saving the cost of the total redesign. Second, the PCIe technology can provide good scalability with ultra-low latency. For example, both CXL and InfiniBand are built atop the PCIe bus. CXL can provide sub-microsecond-scale latency in a rack-scale server[8], and InfiniBand supports around a few thousand connected servers with microsecond-scale communication latency[21]. In addition, the I/O interconnects can add new features to the transport layer, such as reliable connection and hardware-support traffic congestion control. As a result, a single application can be equipped with thousands of cores and petabyte-level memory, which is far beyond the assumptions and expectations of the existing resource management mechanisms and programming models[22]. For example, the widely deployed 4-level paging only supports up to 256 TB of virtual address space and 64 TB of physical address space. It is also impractical to expect programmers to manage such huge memory resources in fine-grained. Hence, how to manage such a large-scale server remains a challenging problem.

Resource disaggregation simplifies the maintenance of datacenter hardware and improves resource management efficiency by virtualizing hardware components as resource pools. First, the new hardware components can be dynamically added to the resource pool without interrupting the execution of deployed applications. Second, resource disaggregation can cope well with the increasing hardware heterogeneity— all the domain-specific accelerators are aggregated as different kinds of pools and are available to the whole datacenter without considering their deployed locations. However, resource disaggregation complicates resource management by introducing asymmetric performance to homogeneous hardware resources. For example, each application can acquire CPU, memory and accelerator resources from different nodes and reorganize them as a virtualized server,

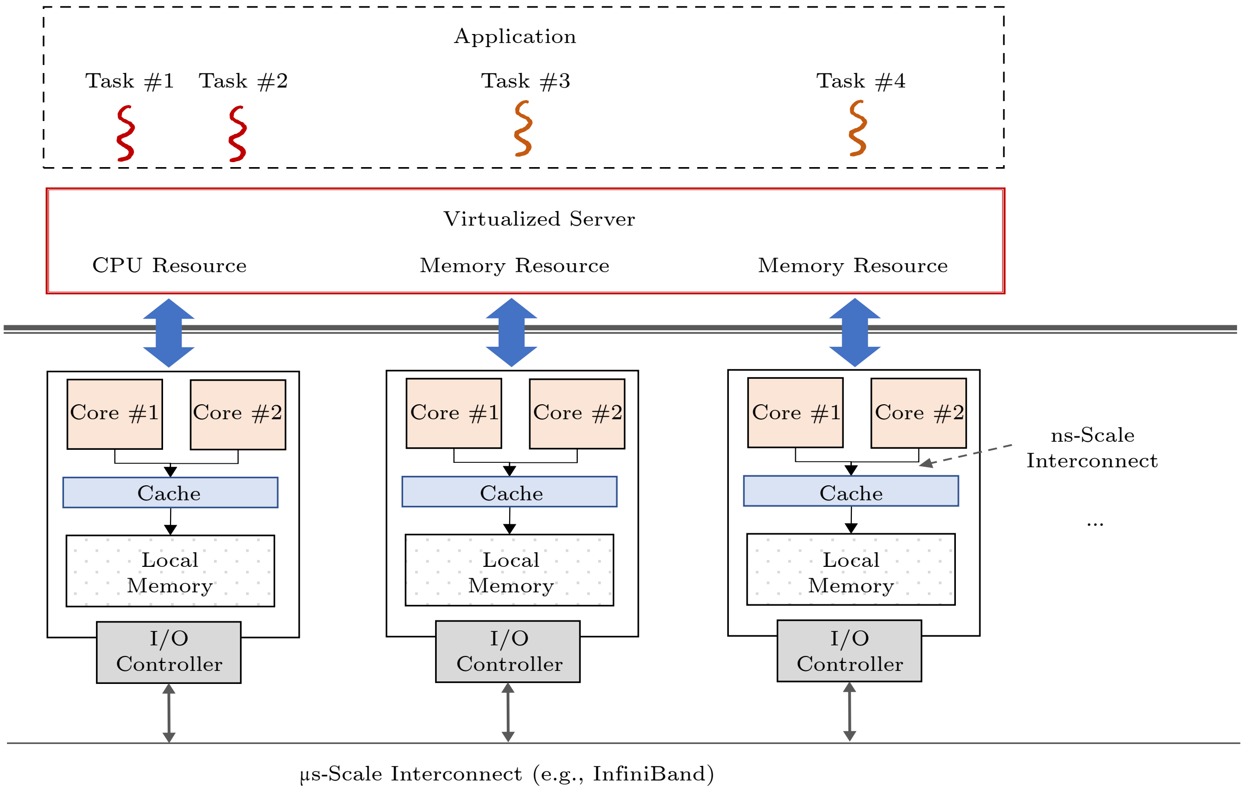

Fig.4. Rack-scale server built from monolithic servers which are connected via the ultra-low latency I/O interconnects. An application can request resources from different servers and treat the acquired resources as a virtualized server under the management of a single system image.

as shown in Fig.4. As a result, the memory resource of an application may come from several different servers and the performance, e.g., latency and bandwidth, may vary significantly according to interconnect topology. These new challenges may cause severe performance degradation and extensive QoS violations to the existing cloud applications[23]. Hence, it is necessary to extend the existing VM management framework, e.g., Kubernetes (k8s), to be aware of the performance variation of CPU and memory resources during the allocation and migration. In other words, resource disaggregation tends to shift the burden of resource management from developers to cloud systems to exchange the scalability and efficiency of hardware development. We will further discuss these new challenges in Section 3.

2.2.2 *Operating System (OS)*

OS is the new firmware. The core functionality of the disaggregated OS is to aggregate and virtualize the disaggregated hardware components as centralized resource pools and provide standard POSIX interfaces to upper-layer applications. In addition, the disaggregated OS is also expected to provide strong performance isolation and fault-tolerance to support the multi-tenant nature of the disaggregated datacenter.

By supporting the standard POSIX interfaces, the legacy applications can run on a disaggregated datacenter unchanged while transparently reaping the benefits such as higher resource utilization, independent failure domains and so on. The key challenge to building such a disaggregated OS is to seamlessly group distributed resources used to be co-located in close proximity. Among all, the most challenging part is disaggregating memory from CPU with good performance and management efficiency. Despite its simple abstraction and backward compatibility, standard SSI (single-system image) disaggregated OS has non-trivial overheads due to its CPU-centric design[2]. Consequently, a series of OS prototypes are proposed to improve the disaggregated memory resource management efficiency from the data-centric[24] and the resource-centric[25] to the fabrics-centric design[26].

The datacenter is designed to be shared by diverse cloud applications. The disaggregated OS should be responsible for providing performance isolation for the co-running applications. For example, a bunch of dedicated memory-disaggregation data planes are proposed to mitigate the microsecond-scale latency via prefetching[27], data migration[28] and compute offloading[4, 12]. All these techniques rely on application semantics monitoring and recognition. However, all these co-running applications share the same I/O interconnect, and the mixed memory access patterns significantly reduce the aforementioned memory-disaggregation management efficiency[3, 23]. Hence, it is necessary to provide strong isolation on the memory-disaggregation data plane and further enable the adaptive management policies of the co-running applications. Besides, due to the far more complicated resource-sharing scenarios in a disaggregated datacenter, the disaggregated OS is also expected to provide robust execution environments[29].

2.2.3 *Runtime System*

Runtime is the key to bridging the semantics gap between diverse applications and disaggregated architectures. As we mentioned above, resource disaggregation brings hardware resource heterogeneity to the datacenter, and the disaggregated OS is only responsible for providing basic resource management mechanisms. Due to lacking specialized optimizations for disaggregated architectures, the system-level optimizations are usually under-performed. Hence, on the one hand, the system builders increasingly rely on the application developers to pass more program semantics down to the low-level systems to improve management efficiency, e.g., data prefetching and job scheduling. Hence, a series of new programming abstracts, e.g., user-level threads[30] and logical processes[31], are proposed to allow programmers to explicitly elaborate the program for disaggregated hardware in fine-grained.

However, on the other hand, application developers tend to use high-level programming languages, e.g., Java, Python and Scala, which hide the complicated resource management details from developers and let them focus on designing the program logic. Hence, it is not practical to expose these extensive hardware features to the application developers and shift all the optimization burden to them. In order to fix the gap, a new trend is to design domain-specific (language) runtime via the newly proposed programming abstracts and mask out the hardware heterogeneity by only providing easy-to-use interfaces, e.g., single address space and automatic data reclamation and migration, to the application developers. By following this philosophy, a series of resource-disaggregation-friendly data structures[32–35] and semantics-aware cross-layer systems[3, 4, 9, 12, 36] are proposed.

2.2.4 Summary

The emerging network technologies bring opportunities to build large-scale clusters and also introduce challenges to the existing cloud software stacks.

3 New Challenges from Resource Disaggregation

The existing cloud software stacks, e.g., operating system, language runtime and programming abstracts, are designed for the CPU-centric monolithic server. On the one hand, the system-level optimizations for the conventional hardware have been well exploited, e.g., the kernel blocking layer relies on batching and thread-level parallelism to mitigate the storage accessing latency and the compiler can reorder the instructions to take full advantage of the CPU's out-of-order execution. On the other hand, the hardware is also optimized to accelerate the execution of the program, e.g., adding the TLB (Translation Lookaside Buffer) to speed up the virtual address translation and atomic instructions to simplify parallel programming. That is to say, the existing cloud software stack is tightly coupled with the characteristics of the conventional architectures of the monolithic server.

However, the server paradigm-shifting to resource disaggregation results in a mismatch between the existing software stack and the hardware features, leading to several new challenges to the cloud—significant performance degradation due to the microsecond-scale latency, severe QoS violations due to the heterogeneity of disaggregated datacenter, and the costly inter-server synchronization. In the following of this section, we will take memory disaggregation as an example to demonstrate the new challenges introduced to cloud systems.

3.1 Lack of Microsecond-Scale Event Mitigation Techniques

The emerging I/O interconnects introduce a new type of latency, microsecond-scale, to the cloud, invalidating many existing system-level mechanisms and optimizations. Taking memory disaggregation as an example, when an application runs on a disaggregated server, its memory resources may come from separated blades connected by microsecond-scale I/O interconnects. The memory resources can be divided into two types with huge latency differences—the nanosecond-scale local memory (residing with CPU) and the microsecond-scale remote memory. The performance gap is so huge, reaching up to approximately 10x to 100x[8, 9, 26], that none of the existing latency mitigation techniques can be directly applied to bridge the gap.

The existing hardware latency-mitigation techniques lack parallelism to hide the microsecond-scale latency. For example, the modern out-of-order CPU utilizes the non-blocking cache[37–39] to issue multiple outstanding memory requests to hide the nanosecond-scale CPU stall caused by memory access. However, due to the hardware limitations, e.g., chip area, cost, and design complexity[40], the memory level parallelism is limited to dozens and still a few orders of magnitude less to hide the microsecond-scale latency.

The existing software latency-mitigation techniques are all designed for storage with millisecond-scale latency. Applying them to the disaggregated datacenter can even worsen the performance. Taking the multi-threading as an example, the context-switch overhead of the POSIX thread reaches up to hundreds of nanoseconds[30] and can be longer if the thread is scheduled to a new core, which can further lead to data movement overhead in the cache hierarchy. Besides, inappropriate batching techniques can only waste more CPU time, and state-of-the-art remote memory data planes[23, 41] tend to utilize the `frontswap` interface⑤ to bypass the unnecessary block layer optimizations. However, the software overhead still accounts for approximately 60% of remote memory accessing latency[5].

3.2 Severe QoS Violations

Resource disaggregation further contributes to the hardware heterogeneity of datacenters by introducing asymmetric performance to the homogeneous resource. For example, memory performance and intercore communication overhead within an application may vary significantly according to the I/O interconnect topology[21]. As reported in previous research[42, 43],

⑤Transcendent memory in a nutshell. https://lwn.net/Articles/454795, Sept. 2023.

the hardware heterogeneity in conventional monolithic datacenters can slow down applications by up to 2x and even lead to workload crashes due to resource exhaustion. The situation can be much worse in a disaggregated cluster. For example, when running on memory disaggregated servers, applications' performance variation can reach up to 7x, with 400x longer 99th percentile latency[5, 23]. The reasons are twofold. 1) Each type of hardware resources is going more heterogeneous, exhibiting a huge performance gap. For example, the bandwidth and the latency of local memory are more than an order of magnitude better than remotely attached memory. As a result, applications' performance varies significantly when acquiring different kinds of memory resources. 2) Interference within a disaggregated cluster can be more complicated. Each application spans multiple resource servers, and each resource server needs to serve many applications. For example, an application can acquire hardware resources from a single CPU server and several memory servers. As a result, the application will race resources with all the other applications co-running on the related servers, leading to an unpredictable performance variation[23, 44].

3.3 Costly Inter-Server Synchronization

In order to satisfy the increasing parallelism of applications, the scale of computers keeps growing, e.g., from the symmetric multiprocessor (SMP) server to the Non-Uniform Memory Access (NUMA) system, and further shifting to the rack-scale and warehouse-scale clusters. As a result, each application crosses multiple cores with complicated interconnection topology, and the data synchronization overhead leads to non-trivial performance impact[45]. For traditional monolithic servers, manufacturers tend to provide hardware support for the data synchronization between different cores, such as the atomic instructions and the hardware-support snooping for cache coherency. Hence, the data synchronization overhead of SMP and NUMA systems is within the nanosecond scale. However, it is very challenging to provide cache coherency in a rack-scale cluster with thousands of cores. Although previous research, e.g., Multicube[46], FLASH[47], has explored the possibility of building a cache-coherent large-scale server, there is still a lack of commodity products today. Hence, we assume that the software is responsible for maintaining the data consistency between disaggregated blades[2, 4]. As a result, due to hardware limitations, the inter-server communication overhead is orders of magnitude higher than the intra-server communication.

As shown in Fig.5, an application with three parallel tasks executes on a monolithic NUMA machine with hardware-support cache coherency, e.g., MESIF[48]. All three tasks are trying to modify the same variable, e.g., application threads and concurrent GC threads race for the same object. For task #1 and task #2, residing in the same SMP processor and sharing the same L3 cache, the synchronization overhead is around 20 nanoseconds to 30 nanosec-

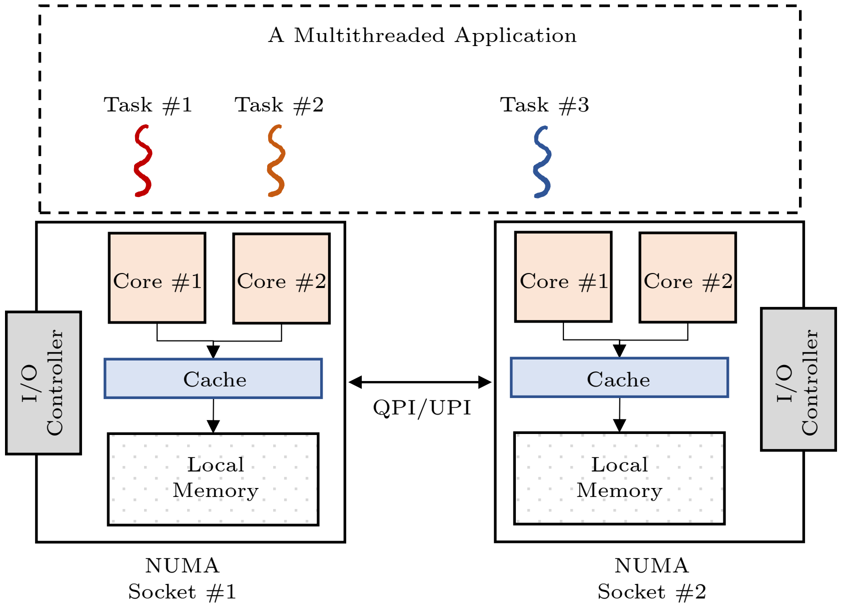

Fig.5. Multithreaded application executing on a NUMA server equipped with Intel Xeon processors. The Xeon processor provides atomic instructions and the snooping cache to ease the parallel programming.

onds under the implementation with atomic instructions. And for the two tasks (e.g., task #1 and task #3) residing in different NUMA processors, the data synchronization overhead can also be still limited within the nanosecond scale, around 100 nanoseconds to 150 nanoseconds[10]. However, if an application runs on a disaggregated cluster, its two threads execute on two separated resource servers connected by the emerging I/O interconnects (e.g., InfiniBand), as shown in Fig.6. Because of the lack of hardware-support cache coherency, inter-server communication is usually implemented via message passing or RPC (Remote Procedure Call), reaching up to 2.3 μs[24, 49], more than 10x slower than the intra-server data synchronization.

4 Semantics-Aware Software Stack

There are two major trends in cloud computing. First, cloud applications are becoming increasingly diverse. Up to 81% of businesses are already using cloud technology in one capacity or another[50]. The applications span a large number of domains, including machine learning, the Internet of Things, big data processing, databases, etc., with dramatically different compute behaviors and memory usage patterns. Second, the hardware heterogeneity starts dominating the architecture development due to the largely ending of Moore's Law[51, 52]. Resource disaggregation further promotes the trend by introducing resource pooling and complicated I/O topology, as discussed in Section 2. As a result, a growing semantics gap begins between the diverse cloud applications and the heterogeneous datacenter hardware, leading to severe performance degradation and frequent Service-Level Agreement (SLA) violations. Hence, it is necessary to build cross-layer software stacks to align the computing behaviors of high-level applications and the characteristics of emerging hardware with program semantics.

In this section, we will first talk about the major reasons leading to the program semantics gap (Subsection 4.1), and then demonstrate the architecture of the semantics-aware software stack (Subsection 4.2) and the represented methods to bridge the gap. Both the industry and academia have made numerous proposals to bridge the semantics gap by building new software stacks from operating systems and abstractions to programming models. We categorize the exploration canvas into three layers: operating system (Subsection 4.3), runtime (Subsection 4.4) and programming interface (Subsection 4.5). This section is by no means a complete survey of all related work. We aim to describe the existing research landscape, how it evolved, and where it is heading.

4.1 Semantics Gap

The high-level programming languages (HHL) provide a variety of runtime supports, e.g., object-oriented programming (OOP) model, Garbage Collection (GC), Just-In-Time (JIT) compiler, to increase

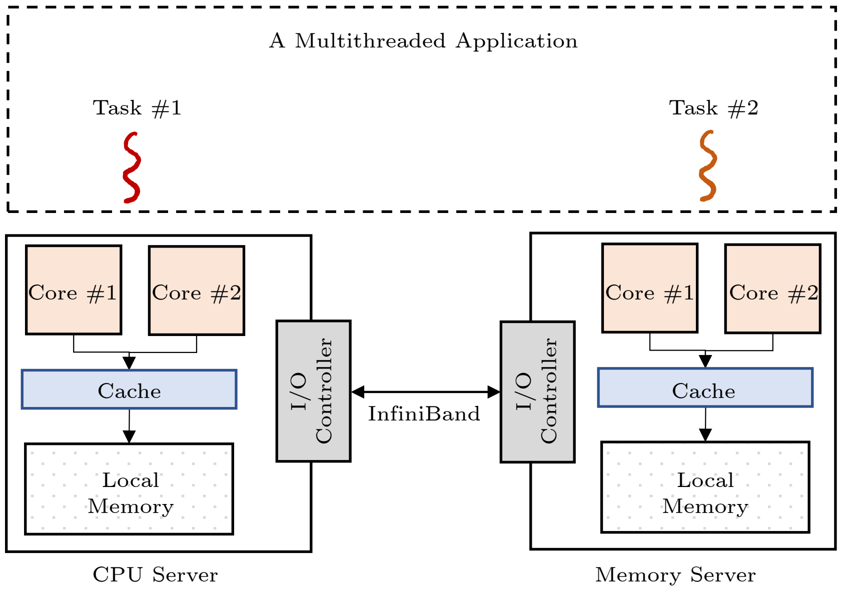

Fig.6. Multithreaded application executing on a disaggregated server connected via InfiniBand. Task #1 and task #2 are racing for the same data and the software is responsible for maintaining the inter-server data consistency.

the development productivity. As a result, developers focus on designing the algorithms instead of managing the resource usage or adapting the program to low-level hardware. Hence, applications rely on the software stack, e.g., runtime and OS, to monitor and recognize the application's computing behaviors and optimize the performance, e.g., data layout adjustment and task scheduling, for the underlying hardware. However, on the one hand, the existing cloud software stacks fail to recognize applications' diverse computing behaviors due to the hierarchical design, resulting in the inefficiency of the optimization strategies. On the other hand, the existing software stacks are designed for the legacy hardware, e.g., slow disks and Ethernet, leading to a mismatch with the emerging I/O interconnects. As a result, the disaggregated hardware components are under-performed. The details of the major reasons are elaborated below.

Data Abstraction Mismatch Between Software Stack Layers. First, each stack layer has unique mechanisms and data abstractions for specialized functionality. The lack of cross-layer co-design prevents application semantics from passing down to the underlying hardware. For example, as Fig.7 shows, there are at least three stack layers between the Apache Spark application and hardware—the distributed framework (Apache Spark[53]), Runtime (OpenJDK⑥) and Operating System (OS). Spark proposes a distributed data abstraction, Resilient Distributed Dataset (RDD), to ease parallel programming and improve the fault tolerance of distributed computing. Runtime relies on the object-oriented programming model to enable automatic memory management, e.g., Garbage Collection. At the same time, OS manages data via the virtual address mechanism to provide isolation, portability and the ability of defragmentation. As a result, even programmers interact with RDDs via simple operations, e.g., word-count (scanning the data in sequence). The simple and clear memory patterns cannot be passed down to the OS layer from the Spark framework layer. This is because of the data abstraction mismatch between stack layers—each RDD contains thousands of objects distributed in discrete OS pages. From the perspective of OS, the program shows random memory patterns resulting in a series of problems, e.g., poor spatial locality and inefficient prefetching, resulting in significant performance degradation and variation[11, 23, 54].

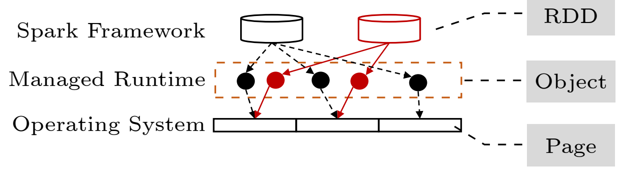

Fig.7. Data abstraction mismatch between different cloud software stacks.

Interference Between Semantics-Agnostic Tasks. Second, the applications written in high-level languages run with various concurrent service threads (tasks), e.g., GC threads, JIT compilation threads, with different compute behaviors. Hence, when these semantics-agnostic threads co-run on the same server, all the different semantics are mixed and cannot be distinguished by the underlying systems, leading to inefficient management policies. As Fig.8 demonstrates, even if application threads have a clear sequential memory access pattern, it cannot be recognized by the underlying OS because of the interference with the co-running GC threads. Compared with disabling the GC threads, the prefetching effi-

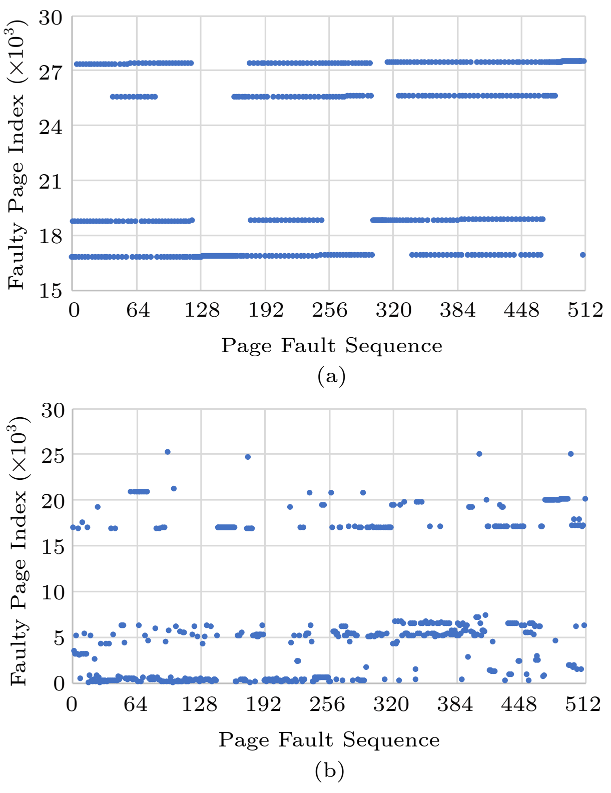

Fig.8. The memory access patterns of the application cannot be passed down to the underlying OS due to the interference between application threads and GC threads[3]. (a) Page fault trace of Spark application. (b) Page fault trace of Spark application with concurrent GC.

⑥An open-source implementation of the Java platform. https://openjdk.org, Sept. 2023.

ciency is reduced by 40%. Because these service threads are crucial to providing vital functions, it is not practical to disable them directly[3].

Lack of Software Support for the Emerging I/O Interconnects. The existing resource management mechanisms and optimizations of the disaggregated datacenter are usually under-performed and even lead to SLA violations and correctness problems, e.g., data inconsistency problems, due to the aforementioned new system-level challenges introduced by the emerging I/O interconnects (Section 3). For example, the existing OS cannot be aware of the I/O topology of the disaggregated resources and usually fails to schedule the compute-intensive and memory-intensive tasks to the proper hardware components. That is to say, the application semantics cannot be fully utilized even if they are successfully passed down to the low-level system. Hence, a redesigned OS equipped with the new programming abstractions, e.g., disaggregated process, and disaggregated resource managements are the basis of the cloud software stacks.

4.2 Design Overview

In order to bridge the semantics gap, we propose the semantics-aware software stack that conveys the diverse application information to the heterogeneous hardware to improve management efficiency and reduce interference by aligning application behaviors. As Fig.9 demonstrates, the semantics-aware software stack contains three basic layers—programming interface, runtime, and operating system. Developers define the behaviors of the runtime layer by providing explicit program semantics via programming interfaces. The runtime layer requests resources from the operating system and is in charge of task scheduling and data migration. The operating system provides hardware virtualization, basic resource management, and necessary fault-tolerance mechanisms.

The programming interface aims to balance the programming productivity with necessary semantics provision. For example, providing a disaggregated type system to let system developers explicitly decide the location of fine-grained variables among the disaggregated memory resources with asymmetric performance can significantly improve the data locality and concurrency safety (Subsection 5.2). However, it is also essential to provide predefined disaggregated data structures, tailored for disaggregated memory, to improve the productivity of application developers with providing fair performance (Subsection 4.5).

Runtime is responsible for providing good performance. Through utilizing the acquired program semantics, runtime needs to adjust the data layout to

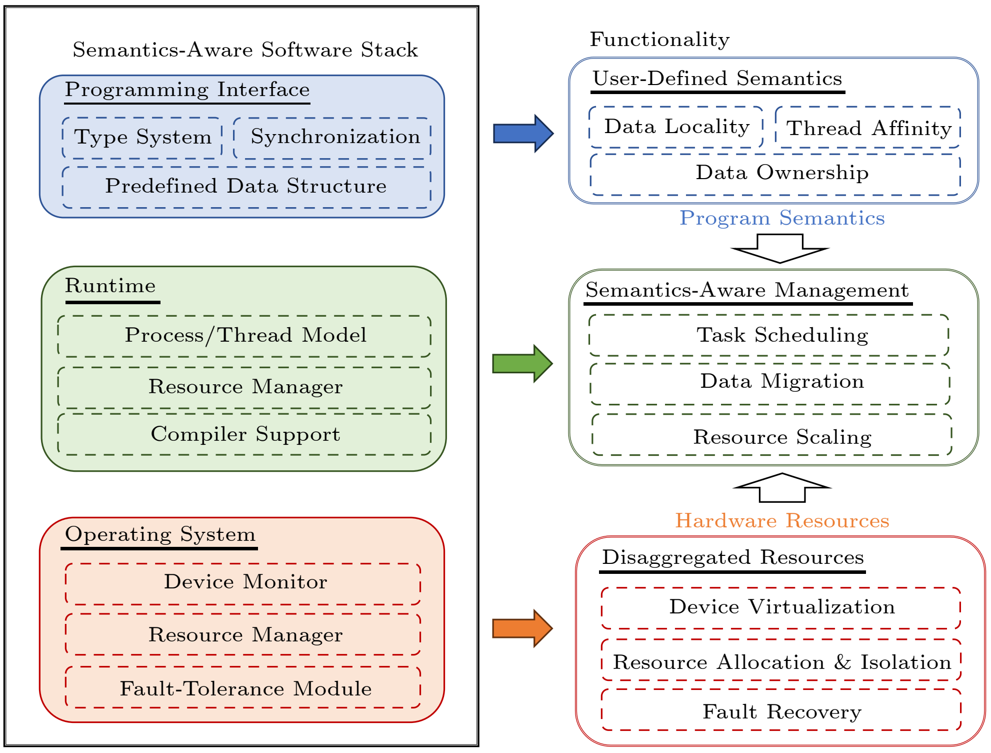

Fig.9. Architecture of the semantics-aware software stack.

mitigate the microsecond-scale latency and schedule different tasks to appropriate disaggregated servers to improve the CPU and memory efficiency. One step further, the disaggregated runtime also proposes a series of process, thread and data models to improve the efficiency of disaggregated hardware (Subsection 4.4), such as the user-level thread, Shenango[30], which can provide ultra-high parallelism with fast context switch, and the new process model, Nu[31], which supports microsecond-scale migration. Runtime is the key to bridging the semantics gap between upper-level diverse applications and low-level heterogeneous architectures.

The operating system provides basic management modules to cope with the paradigm-shifting of the datacenter hardware. As the hardware components are disaggregated, the kernel is also split to manage the resources and provides communication primitives between disaggregated nodes[2, 55]. However, the operating system should largely leave the fine-grained resource management to the runtime and only needs to provide different types and quantities of resources according to runtime needs. In addition, due to the multi-tenant nature of cloud computing, the operating system also needs to provide device virtualization and performance isolation among co-running applications (refer to Subsection 4.3 for more details).

4.3 Disaggregated Operating System

An OS designed for the disaggregated datacenter will manage a sea of disaggregated hardware resources and expose a single-system image (SSI) to user-level programs. Crucially, it would maintain backward-compatible APIs for user-level programs to reap the benefits of resource disaggregation with no or minor changes.

The disaggregated OS approach is a unique design point within the exploration canvas for using disaggregated resources. It was among the initial systematic studies on leveraging disaggregated resources since Lim *et al.*[22] first proposed memory disaggregation. Two generations of the disaggregated OS have emerged: a CPU-centric one called LegoOS[2] and a data-centric one called FractOS[24]. This evolution happens in a short 5-year span, reflecting the design philosophy of a disaggregated OS changes as the workloads it hosts shift: the best cost-efficiency is achieved when we co-design the OS and the workloads at the sacrifice of flexibility and generality. We will now briefly discuss two pioneer studies in this space.

CPU-Centric Disaggregated OS. LegoOS[2] is the first-generation, CPU-centric disaggregated OS. It offers a set of Linux-compatible system calls and can run unmodified Linux binaries (such as TensorFlow) over a disaggregated infrastructure. LegoOS embraces the CPU-centric model. All the orchestration of data must go through the CPU. In order to expose an SSI abstraction, LegoOS proposes to run a monitor on each disaggregated device, and each monitor is highly customized to the device it runs on. For example, the monitor on a CPU device is only responsible for task execution, while the monitor on a memory device is responsible for virtual memory handling. OS functionalities are cleanly distributed among heterogeneous devices. LegoOS proposes a two-level resource management scheme in which a global layer oversees cluster-wide resource allocations, and each monitor handles device-local allocations. Despite its simple and backward-compatible APIs, LegoOS incurs non-trivial performance overheads (25% for a typical application) mainly because 1) both the memory access latency and bandwidth are worse in the disaggregated setting compared to the monolithic setting, and 2) the CPU-centric model leads to unnecessary data copies among CPU devices and other devices. In order to mitigate these issues, the second generation was proposed.

Data-Centric Disaggregated OS. FractOS[24], as the second-generation disaggregated OS, eschews the CPU-centric model and embraces a data-centric one. Instead of exposing a traditional POSIX abstraction, FractOS expects a program written in a DAG fashion, with each vertex representing a particular operator running on a specific disaggregated device and each edge representing how the data flows. Similar to LegoOS, FractOS runs a monitor on each device. FractOS will execute the DAG based on its description. A unique challenge in building FractOS is ensuring security and isolation. FractOS leverages distributed capability to solve this issue. However, the downside of using FractOS is that programmers need to rewrite legacy applications and adopt FractOS's DAG-based programming model.

Fundamentally, the design philosophy of a disaggregated OS is not different than that of a classical single-node OS. There is a constant tension between achieving the best performance and offering good generality. LegoOS aims for universal generality with non-trivial overheads, in contrast to FractOS, which co-designs with data systems for the best performance. Going forward, we expect more customized solutions will emerge for the best cost-efficiency, and we believe it is likely a co-design of applications, OS, and the underlying hardware.

4.4 Disaggregated Runtime

The major philosophy of runtime solutions is to improve the system efficiency by exploiting the program semantics and hide the heterogeneity of disaggregated hardware from application developers[9, 11, 12]. Hence, first, the disaggregated runtime should be co-designed with the underlying disaggregated OS to enable the program semantics recognition and passing down to the low-level software stacks to improve the efficiency of system-level optimizations; second, the runtime also needs to provide disaggregated programming abstracts to ease the high-level cloud application developments.

4.4.1 Disaggregated Abstractions Targeted at the Killer Microseconds

In order to fully take advantage of the emerging hardware, the researchers from Berkeley[56], MIT[9, 30, 31] and VMWare[12] proposed several new programming abstractions, e.g., Shenango (user-level thread)[30], Nu (disaggregated process)[31], AIFM (disaggregated APIs)[9] and Kona (hardware-support disaggregated primitives)[12], to let developers build disaggregated applications from scratch. All these system abstractions target the new challenges discussed in Section 3, e.g., mitigating microsecond-scale events and providing microsecond-scale task migration between different servers. For example, Shenango is a new type of user-level thread, similar to the green thread and coroutine, optimized for fast context switch. Shenango supports CPU reallocation every 5 µs, orders of magnitude faster than conventional POSIX thread (PThread). Hence, compared with PThread-based applications, Shenango-based applications have much higher thread-level parallelism. On the one hand, when triggering a microsecond-scale remote memory access, the lightweight Shenango thread can yield the CPU resources to other ready Shenango threads to improve CPU efficiency. On the other hand, Shenango-based applications can issue many more in-the-fly remote memory access requests to saturate the available RDMA bandwidth to improve the memory-level parallelism further.

Other abstractions also provide novel functions based on the emerging I/O interconnects: Nu[31] proposes a new disaggregated process model, which divides the process into dozens of prolects, and allows the prolect to be migrated between different nodes of a rack-scale server within 100 µs; Kona[12] allows the user to fetch data from memory servers in fine-grained granularity, i.e., cache line, to eliminate the read/write amplification caused by the existing swap system.

Developers can rebuild the applications for the disaggregated datacenter with these newly proposed hardware-specific abstractions to enjoy the benefits of resource disaggregation. However, such clean-slate solutions cannot support the existing applications. Developers also need to make great efforts to elaborate their programs to achieve reasonable performance. Although rebuilding every program is infeasible for both industrial and academic developers relying on countless open-source libraries and software, we believe that these new prototypes and abstractions have the potential to reshape cloud systems in the future.

4.4.2 Semantics-Aware Runtime Improving the Efficiency of System Management

The disaggregated runtime is responsible for monitoring program semantics and guiding the behaviors of low-level system policies. Hence, semantics awareness started becoming the design principle of modern disaggregated runtime—horizontally, the runtime should recognize and isolate the semantics of different tasks to avoid interference; vertically, the cross-layer design enables the semantics propagation from user layers to the underlying kernel and virtualization layer.

Horizontally, Semeru[4] separates the application threads and GC threads, and schedules them to the CPU servers with powerful computing resources and the memory servers where the data resides correspondingly. The benefits are twofold. First, Semeru reduces the interference of semantics-agnostic threads. Second, the GC threads can run concurrently and continuously in the memory servers without interrupting and racing resources with application threads. And then they proposed a new concurrent GC algorithm, Mako[36], to further exploit the concurrency of GC threads running on the memory servers. For the tasks running on the CPU server, MemLiner[3] is proposed as a way to reconcile the memory footprint of application threads and service threads, e.g., object tracing. After the above optimizations, the semantics of applications can be passed down to the underlying system stacks, which significantly improves the efficiency of existing management policies, e.g., the coverage and accuracy of OS prefetching[27] are improved by 50% and 70%, respectively. Besides, these optimization can reduce the memory footprint of cloud applications, which leads to 56% less costly remote memory access.

Vertically, application-integrated far memory (AIFM)[9] designs semantics-aware prefetcher, evacuator and frequently used data structures, e.g., hash table, based on the Shenango threads. With the help of program semantics, the efficiency of system-level mechanisms is significantly improved. For example, AIFM prefetcher can accurately fetch data at fine-grained granularity, i.e., object, which eliminates the read/write amplification of paging-based far-memory data path, e.g., Fastswap, and improves throughput via extremely high thread-level parallelism of user-level threads.

Similarly, Panthera[11] matches the distributed data structures with the runtime objects and the low-level OS pages to improve the data layout on hybrid memory. Panthera allows users or compilers to tag properties, e.g., hot, cold or fault-tolerant, on different data structures. And then Panthera utilizes the GC to automatically and transparently propagate the semantics to the related objects and then compact the discrete objects into contiguous pages according to the preference of users. As a result, the computing behaviors and memory access patterns on the distributed data structures can be clearly recognized by the underlying OS and hardware, which significantly improves the efficiency of low-level optimizations, such as data prefetching and data layout adjustment. In order to bridge the semantics gap, a series of new hardware prototypes are also proposed, such as the Programmable Architecture for Resourcing-on-Demand (PARD)[57] and HoPP[28].

In addition, the disaggregated runtime provides the same programming interfaces with the monolithic programming model, e.g., single address space and object-oriented programming. As a result, the burden shifts from application developers to system developers and all the existing managed applications can benefit from these systems transparently.

4.5 Disaggregated Programming Interface

Data structures are fundamental building blocks of many cloud applications, e.g., databases, key-value stores, and file systems, used for organizing, storing, and retrieving data. Conventional data structures are originally designed for and used in local memory, which, however, become inefficient in disaggregated memory (DM) systems. This is mainly due to the challenges discussed in Section 3, such as the big performance gap between the local memory with nanosecond-scale latency and the disaggregated memory with microsecond-scale latency. Researchers from both industry and academia started reinventing data structures that are aware of the characteristics of disaggregated memory. These DM-aware data structures[32–35, 58–61] have much better spatial locality and support task offloading to mitigate the aforementioned microsecond-scale events. In this subsection, we present the general design guidelines via several critical data structures, e.g., hash table, tree, and learned index, which are widely used in cloud applications.

4.5.1 Hash Table

Hash tables are popular data structures that are widely used to develop latency-critical applications and provide fast lookup services in distributed memory systems, such as the Memcached[⑦] and Redis[⑧]. However, due to a lack of consideration of the characteristics of the disaggregated resources, e.g., the asymmetric performance of the memory resource (Subsection 3.3), the applications exhibit significant performance degradation and variation, causing excessive QoS violations (Subsection 3.2).

Zuo *et al.*[35] proposed RACE, a disaggregated hash table with three new techniques to address the challenges. First, RACE introduces a concurrent lock-free remote access mechanism to hide the microsecond-scale latency by increasing the memory parallelism. Second, in order to reduce the QoS violations, RACE presents a one-side RDMA-conscious hash table structure that achieves constant worst-case RDMA access times for all operations, including search, insert, delete, and update. Finally, RACE leverages a client-side directory cache to reduce the remote access to the directory of the hash table and designs three rules for detecting and resolving the cache in-

[⑦]A memory object caching system. http://memcached.org, Sept. 2023.
[⑧]An in-memory data store. https://redis.io, Sept. 2023.

consistency cases. Based on RACE[35], Shen *et al.*[32] designed a fully memory-disaggregated key-value store called FUSEE. FUSEE guarantees the data consistency of multiple hash table replicas stored in the disaggregated memory and is able to tolerate both client and server failures.

4.5.2 *Tree*

Computing on tree data structures tends to exhibit pointer-chasing style irregular memory access patterns that rely on the accessed data to determine the following memory accesses. Compared with other applications, tree-based applications are more memory latency sensitive and perform much worse when running on the memory disaggregated servers. Based on these semantics, it is necessary to design DM-aware tree data structures to achieve reasonable performance.

Ziegler *et al.*[58] proposed the first tree index FG that purely employs one-sided RDMA verbs. FG is built on a B-link tree structure (a variant of the B+-tree) to support disaggregated memory. Wang *et al.*[59] proposed a write-optimized B+tree Sherman to achieve higher performance on disaggregated memory. Sherman combines the local lock table and global on-chip lock table to reduce the overhead of concurrent accesses and presents a two-level version mechanism to reduce the remote write amplification. Luo *et al.*[34] proposed that the Radix tree is a more suitable tree index for disaggregated memory than the B+ tree since it has less remote read and write amplifications. They presented the first Radix tree designed for disaggregated memory called SMART. SMART leverages lock-free internal nodes and lock-based leaf nodes to reduce the lock overhead, a read-delegation and write-combining mechanism to improve the overall throughput, and a reverse check method to validate the cache consistency.

4.5.3 *Learned Index*

Learned indexes[60, 61] are a new type of indexes that leverage machine learning methods to learn the cumulative distribution function of the sorted keys. Due to the small model size, learned indexes have much lower memory space overhead and higher performance than tree indexes. Several scalable learned indexes are proposed for the disaggregated memory system, e.g., XStore[60], ROLEX[33]. However, machine learning models are only good at static workloads, such as read and scan. The dynamical workloads, e.g., insert and delete, keep changing the data structure, which leads to the need to retrain the ML models. In order to address this challenge, XStore proposes a hybrid path that leverages the tree index for dynamical workloads and the learned index for static workloads. ROLEX proposes to decouple the retraining of learned models from the dynamic operations. By doing so, model retraining can be pushed down to the memory pool and executed asynchronously. The insertion operations are executed in the compute nodes via pure one-sided RDMA verbs, delivering high performance and scalability.

5 Open Problems

The high demand for memory capacity and the urgent need to improve resource utilization motivate the explosive development of resource disaggregation technologies. As new I/O interconnects are being introduced to the market[8], system developers need to pay more attention to the new features and reinvent the system-level mechanisms and optimizations. In this section, we will discuss the new research trends in the cloud system.

5.1 Hardware for Disaggregation

Hardware evolution is the driving force behind the redesign of cloud software stacks. Crucially, the fast datacenter network plays a key role in making disaggregation a reality since resources used to be accessed within a single chassis via high-speed PCIe interconnect are now accessed through emerging datacenter networking with comparable performance. Akin to the OS design philosophy shift, monolithic servers no longer fit as the building blocks of a disaggregated datacenter. In response, customized devices tailored for disaggregated datacenters have been proposed. In this subsection, we will review notable studies related to networks and disaggregated devices.

5.1.1 *Datacenter Networking*

The Compute Express Link (CXL)[6] is an emerging interconnect technology for memory disaggregation. It offers sub-microsecond-level access latency for rack-scale memory, akin to accessing memory on a neighboring NUMA node. Accessing disaggregated memory via CXL is much faster than canonical networking technologies such as Ethernet. Essentially, CXL shortens the software path from where applications access memory via load/store instructions (or explicit APIs) to where the network requests are transmitted. CXL achieves so by claiming a segment of PCIe bus addresses and hardening the translation from local application semantics (e.g., load/store or explicit APIs) to network requests (e.g., PCIe packets)[6].

Aquila[62] is a recent work that revisits the decade-old datacenter networking infrastructure and calls for a tightly-coupled one tailored for resource disaggregation. Aquila proposes to break the strict boundaries among layered network protocols and co-design the transport layer and link layer for the best tail latency. It also proposes to use a Dragonfly topology to group heterogeneous disaggregated resources in close proximity instead of using a traditional multi-tier Clos topology. Aquila offers many insights on how a disaggregated datacenter can be deployed at a large scale.

A key challenge in using these emerging I/O interconnects is that the asymmetric performance of the memory resource complicates the system design (Subsection 3.2). Take memory disaggregation as an example. The CXL-attached memory exhibits approximately 3x to 5x larger latency than the local memory in a rack-scale server[8] which limits the scalability of building larger-scale clusters due to the increasingly complicated networking topology. System developers need to carefully partition and migrate data among the disaggregated servers to achieve reasonable performance. The complicated design of the existing cloud software stacks keeps weakening the predictability, stability and security of the cloud[63]. Hence, how to reduce the disorders and improve the efficiency of the cloud via software-hardware co-design stays a challenging problem.

5.1.2 *Disaggregated Devices*

Since the monolithic server is an ill-fit for building disaggregated datacenters, many researchers have proposed customized devices. Notably, Clio[64], Farview[65], and StRoM[66] are FPGA-based disaggregated memory devices. By design, they have no beefy CPUs attached and only adopt a wimpy CPU for control and management. Their data path is customized for remote data access. They usually support offloading user-defined operators onto the devices to achieve near-data processing. For instance, StRoM[66] has a remote pointer-chasing API, Clio offers remote KV semantics, and Farview can run database operators. As the disaggregation architecture matures, we expect more customized devices to emerge and flourish in areas like disaggregated storage and computing devices.

These heterogeneous devices are highly specialized and can only speed up limited domain-specific tasks. If the underlying systems cannot correctly recognize the behaviors of different computing tasks and schedule them to proper devices, these applications will be under-performed and pay heterogeneity tax. However, the ability of software-based program analytics is limited, and the system developers expect emerging devices to provide more hardware support for program instrumentation and monitoring.

5.2 Programming Language for Disaggregation

Designing disaggregated programming models with proper runtime support and letting the system developers build semantic-aware systems from scratch can reduce the performance overhead and potentially eliminate the data inconsistency introduced by the disaggregated architecture. As we discussed in Section 3, when running on a memory disaggregated datacenter, the performance overhead mainly comes from the inter-server data movement and thread-level synchronization. Hence, we think that the disaggregated programming language should have the ability to express the correlations between 1) data and data, 2) thread and data, 3) thread and thread, which can help the runtime system precisely and efficiently recognize the application's compute and data accessing behaviors. And then, this information can be utilized to improve the effectiveness of system-level management mechanisms, e.g., thread scheduling and data migration.

5.2.1 *Data Management*

Although there are a series of data management research, e.g., prefetching techniques[9, 27, 67], data eviction mechanisms[5, 41] and data layout adjustment optimizations[11, 68, 69], how to mitigate the microsecond-scale inter-server data accessing latency stays one of the most challenging problems in the disaggregated datacenter[7]. For example, in order to address the inaccuracy and data amplification problem of the paging-based remote memory management[20, 41], AIFM[9] launches dozens of threads to periodically scan and categorize the data in fine-grained object granularity. These online profiling threads consume and race significant CPU resources with application threads, which in turn degrades the application's performance.

Hence, such a solution only shifts the space racing problem, e.g., hot data vs cold data, to another CPU resource racing problem, i.e., application threads vs profiling threads. There is no panacea for the challenge of data management in a heterogeneous memory system.

However, with the guidance of programming hints, the relationship between different objects can be quickly identified with low overhead. The benefits are twofold: first, the correlated objects can be moved between disaggregated servers in batches, reducing the data migration overhead; second, the data access patterns can be recognized via the static program analysis, e.g., during the compilation, which can be used to precisely prefetch the useful data to hide the microsecond-scale latency. Hence, we believe it is reasonable and profitable to extend the type system or annotation system of the programming language to the disaggregated architectures.

5.2.2 *Thread Synchronization*

As we discussed in Subsection 4.1, in order to fulfill the increasing requirements of users and developers, the cloud applications are equipped with more and more threads (tasks). For example, a database application, Cassandra[9], contains over 100 separate tasks, including persistent tasks, networking service tasks, garbage collection (GC) tasks, caching tasks, etc. These concurrent tasks are usually semantics-agnostic and tend to cause various performance bottlenecks due to resource racing and lock contention. Making things more challenging, the existing trend is to increase the thread-level parallelism to hide the microsecond-scale latency that widely exists in the disaggregated cluster[9, 30]. Hence, letting developers define the thread synchronization via message passing to improve the task scheduling efficiency is becoming necessary, although such a design may incur non-negligible programming efforts to developers and communication overhead between threads.

In addition, we can utilize the ownership[10] to explicitly define the correlation between data and threads during the execution and help multi-threaded applications to guarantee data consistency when running on disaggregated servers. There are some pioneering studies targeting reducing the thread locking and data racing by carefully orchestrating the parallel threads, such as Singularity[70] and MemLiner[3]. However, balancing the expressiveness and programming efforts of a language remains a challenging and open problem.

5.3 Other Important Problems

Fault-Tolerance. Although memory disaggregation improves resource utilization by allowing applications to request resources across multiple servers, this paradigm weakens the robustness of the memory system. Compared to the monolithic cluster, each memory server in the disaggregated cluster can be shared by more applications. Hence, a memory server failure can potentially have a much wider impact on the cloud. Some fault-tolerance mechanisms have been proposed recently to help the applications recover from a memory sever failure, e.g., Carbink[29] and Hydra[71]. However, these erasure-coding based solutions lead to nontrivial performance and space overhead. For example, limited by the erasure-coding calculation, Carbink only supports data swapping in and out at span (several pages) granularity, resulting in significantly read/write amplification[12]. Besides, the space overhead can reach up to 35%, which almost cancels out the benefits of memory utilization improvement. At the same time, the conventional ECC (Error Correction Code) technology cannot be directly applied to the remotely attached memory due to the lack of hardware support. Hence, how to build a fault-tolerant memory system stays a challenging and important research topic for the disaggregated datacenter.

Quality of Service (*QoS*). Although the QoS has been well exploited from networking to CPU scheduling and memory systems in the past decades[72–76], as we mentioned in Section 3, it remains a severe problem in the disaggregated cloud, and is becoming even more challenging due to the complicated resource-sharing situations.

Storage and Accelerator Disaggregation. Unlike CPUs and memory, the storage and accelerator resources are already I/O attached. Hence, the fast I/O interconnects bring more opportunities than challenges to the existing storage and accelerator systems, e.g., cache layer, block layer, file system, accelerator runtime and programming models. For example, blk-switch[77] proposes a redesigned kernel block layer to achieve µs-scale tail latency for applications running on clusters with disaggregated storage. In addition, the emerging interconnect, e.g., NVLink-C2C, allows

[9] A NoSQL distributed database. https://cassandra.apache.org, Sept. 2023.
[10] The rust programming language. https://doc.rust-lang.org, Sept. 2023.

NVIDIA to build the rack-scale GPU cluster, DGX GH200[11]. The NVLink-C2C provides hardware-support cache coherence to the connected CPU (Grace) and GPU (Hopper), which significantly reduces the programming overhead for managing the data movement and maintaining the data consistency between GPU and CPU.

Resource disaggregation presents so many active challenges and opportunities for the existing cloud systems. It is time to reinvent the software stacks for this emerging architecture.

6 Conclusions

The resource disaggregation pushes the scalability, availability, and serviceability of cloud computing to the Next Era. At this moment, we should rethink the design of cloud software stacks, from programming interfaces and runtime to the operating systems, to fully take advantage of the benefits and cope with the new challenges introduced by the disaggregated architectures.

Acknowledgements We thank the anonymous reviewers for their valuable and thorough comments. We are grateful to our shepherd Zhi-Wei Xu for his feedback.

Conflict of Interest The authors declare that they have no conflict of interest.

References

[1] Gao P X, Narayan A, Karandikar S, Carreira J, Han S, Agarwal R, Ratnasamy S, Shenker S. Network requirements for resource disaggregation. In *Proc. the 12th USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2016, pp.249–264.

[2] Shan Y Z, Huang Y T, Chen Y L, Zhang Y Y. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *Proc. the 13th USENIX Conference on Operating Systems Design and Implementation*, Oct. 2018, pp.69–87.

[3] Wang C X, Ma H R, Liu S, Qiao Y F, Eyolfson J, Navasca C, Lu S, Xu G H. MemLiner: Lining up tracing and application for a far-memory-friendly runtime. In *Proc. the 16th USENIX Symposium on Operating Systems Design and Implementation*, July 2022, pp.35–53.

[4] Wang C X, Ma H R, Liu S, Li Y Q, Ruan Z Y, Nguyen K, Bond M D, Netravali R, Kim M, Xu G H. Semeru: A memory-disaggregated managed runtime. In *Proc. the 14th USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2020, pp.261–280.

[5] Qiao Y F, Wang C X, Ruan Z Y, Belay A, Lu Q D, Zhang Y Y, Kim M, Xu G H. Hermit: Low-latency, high-throughput, and transparent remote memory via feedback-directed asynchrony. In *Proc. the 20th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2023, pp.181–198.

[6] Gouk D, Lee S, Kwon M, Jung M. Direct access, high-performance memory disaggregation with DirectCXL. In *Proc. the 2022 USENIX Annual Technical Conference*, July 2022, pp.287–294.

[7] Barroso L, Marty M, Patterson D, Ranganathan P. Attack of the killer microseconds. *Communications of the ACM*, 2017, 60(4): 48–54. DOI: 10.1145/3015146.

[8] Li H C, Berger D S, Hsu L, Ernst D, Zardoshti P, Novakovic S, Shah M, Rajadnya S, Lee S, Agarwal I, Hill M D, Fontoura M, Bianchini R. Pond: CXL-based memory pooling systems for cloud platforms. In *Proc. the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Jan. 2023, pp.574–587. DOI: 10.1145/3575693.3578835.

[9] Ruan Z Y, Schwarzkopf M, Aguilera M K, Belay A. AIFM: High-performance, application-integrated far memory. In *Proc. the 14th USENIX Conference on Operating Systems Design and Implementation*, Nov. 2020, Article No. 18.

[10] Schweizer H, Besta M, Hoefler T. Evaluating the cost of atomic operations on modern architectures. In *Proc. the 2015 International Conference on Parallel Architecture and Compilation*, Oct. 2015, pp.445–456. DOI: 10.1109/PACT.2015.24.

[11] Wang C X, Cui H M, Cao T, Zigman J, Volos H, Mutlu O, Lv F, Feng X B, Xu G H. Panthera: Holistic memory management for big data processing over hybrid memories. In *Proc. the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Jun. 2019, pp.347–362. DOI: 10.1145/3314221.3314650.

[12] Calciu I, Imran M T, Puddu I, Kashyap S, Al Maruf H, Mutlu O, Kolli A. Rethinking software runtimes for disaggregated memory. In *Proc. the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 2021, pp.79–92. DOI: 10.1145/3445814.3446713.

[13] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. In *Proc. the 6th Symposium on Operating System Design and Implementation*, Dec. 2004.

[14] Foster I, Kesselman C. The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers Inc., 2003.

[15] Fan J P, Chen M Y. Dynamic self-organized computer architecture based on grid-components (DSAG). *Journal of Computer Research and Development*, 2003, 40(12): 1737–1742. (in Chinese)

[16] Li L, Cao Z, Chen M Y, Fan J P. A reconfigurable optical interconnect system for DSAG. In *Proc. the 6th International Conference on Parallel and Distributed Comput-*

[11]NVIDIA DGX GH200 AI Supercomputer, AI Supercomputer for the Generative AI Era. 2023. https://resources.nvidia.com/en-us-dgx-gh200/technical-white-paper, Sept. 2023.

ing Applications and Technologies, Dec. 2005, pp.31–34. DOI: 10.1109/PDCAT.2005.40.

[17] Asanović K, Bodik R, Catanzaro B C, Gebis J J, Husbands P, Keutzer K, Patterson D A, Plishker W L, Shalf J, Williams S W, Yelick K A. The landscape of parallel computing research: A view from Berkeley. Technical Report, No. UCB/EECS-2006-183, EECS Department, University of California, 2006. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html, Sept. 2023.

[18] Asanović K. FireBox: A hardware building block for 2020 warehouse-scale computers. In *Proc. the 12th USENIX Conference on File and Storage Technologies*, Feb. 2014.

[19] Li S. High throughput remote memory data path for cloud application [Bachelor's Thesis]. University of Chinese Academy of Sciences, 2023. (in Chinese)

[20] Mellanox Technologies Inc. Introduction to InfiniBand. White Paper. https://network.nvidia.com/pdf/whitepapers/IB_Intro_WP_190.pdf, Sept. 2023.

[21] Subramoni H, Potluri S, Kandalla K, Barth B, Vienne J, Keasler J, Tomko K, Schulz K, Moody A, Panda D K. Design of a scalable InfiniBand topology service to enable network-topology-aware placement of processes. In *Proc. the International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov. 2012. DOI: 10.1109/SC.2012.47.

[22] Lim K, Chang J C, Mudge T, Ranganathan P, Reinhardt S K, Wenisch T F. Disaggregated memory for expansion and sharing in blade servers. In *Proc. the 36th Annual International Symposium on Computer Architecture*, Jun. 2009, pp.267–278. DOI: 10.1145/1555754.1555789.

[23] Wang C X, Qiao Y F, Ma H R, Liu S, Zhang Y Y, Chen W G, Netravali R, Kim M, Xu G H. Canvas: Isolated and adaptive swapping for multi-applications on remote memory. In *Proc. the 20th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2023.

[24] Vilanova L, Maudlej L, Bergman S, Miemietz T, Hille M, Asmussen N, Roitzsch M, Härtig H, Silberstein M. Slashing the disaggregation tax in heterogeneous data centers with FractOS. In *Proc. the 17th European Conference on Computer Systems*, Mar. 2022, pp.352–367. DOI: 10.1145/3492321.3519569.

[25] Guo Z Y, Blanco Z, Shahrad M, Wei Z R, Dong B L, Li J M, Pota I, Xu H, Zhang Y Y. Decomposing and executing serverless applications as resource graphs. arXiv: 2206.13444, 2022. https://arxiv.org/abs/2206.13444, Oct. 2023.

[26] Liu M. Fabric-centric computing. In *Proc. the 19th Workshop on Hot Topics in Operating Systems*, Jun. 2023, pp.118–126. DOI: 10.1145/3593856.3595907.

[27] Al Maruf H, Chowdhury M. Effectively prefetching remote memory with leap. In *Proc. the 2020 USENIX Conference on USENIX Annual Technical Conference*, July 2020, Article No. 58.

[28] Li H F, Liu K, Liang T, Li Z J, Lu T Y, Yuan H, Xia Y B, Bao Y G, Chen M Y, Shan Y Z. HoPP: Hardware-software co-designed page prefetching for disaggregated memory. In *Proc. the 2023 IEEE International Symposium on High-Performance Computer Architecture*, Feb. 25–Mar. 1, 2023, pp.1168–1181. DOI: 10.1109/HPCA56546.2023.10070986.

[29] Zhou Y, Wassel H M G, Liu S H, Gao J Q, Mickens J, Yu M L, Kennelly C, Turner P, Culler D E, Levy H M, Vahdat A. Carbink: Fault-tolerant far memory. In *Proc. the 16th USENIX Symposium on Operating Systems Design and Implementation*, July 2022, pp.55–71.

[30] Ousterhout A, Fried J, Behrens J, Belay A, Balakrishnan H. Shenango: Achieving high CPU efficiency for latency-sensitive datacenter workloads. In *Proc. the 16th USENIX Conference on Networked Systems Design and Implementation*, Feb. 2019, pp.361–378.

[31] Ruan Z Y, Park S J, Aguilera M K, Belay A, Schwarzkopf M. Nu: Achieving microsecond-scale resource fungibility with logical processes. In *Proc. the 20th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2023, pp.1409–1427.

[32] Shen J C, Zuo P F, Luo X C, Yang T Y, Su Y X, Zhou Y F, Lyu M R. FUSEE: A fully memory-disaggregated key-value store. In *Proc. the 21st USENIX Conference on File and Storage Technologies*, Feb. 2023, pp.81–97

[33] Li P F, Hua Y, Zuo P F, Chen Z Y, Sheng J J. ROLEX: A scalable RDMA-oriented learned key-value store for disaggregated memory systems. In *Proc. the 21st USENIX Conference on File and Storage Technologies*, Feb. 2023, pp.99–113.

[34] Luo X C, Zuo P F, Shen J C, Gu J Z, Wang X, Lyu M R, Zhou Y F. SMART: A high-performance adaptive radix tree for disaggregated memory. In *Proc. the 17th USENIX Symposium on Operating Systems Design and Implementation*, July 2023, pp.553–571.

[35] Zuo P F, Sun J Z, Yang L, Zhang S W, Hua Y. One-sided RDMA-conscious extendible hashing for disaggregated memory. In *Proc. the 2021 USENIX Annual Technical Conference*, July 2021, pp.15–29.

[36] Ma H R, Liu S, Wang C X, Qiao Y F, Bond M D, Blackburn S M, Kim M, Xu G H. Mako: A low-pause, high-throughput evacuating collector for memory-disaggregated datacenters. In *Proc. the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, Jun. 2022, pp.92–107. DOI: 10.1145/3519939.3523441.

[37] Li S, Chen K, Brockman J B, Jouppi N P. Performance impacts of non-blocking caches in out-of-order processors. Technical Report, HPL-2011-65, HP Laboratories, 2011. https://www.hpl.hp.com/techreports/2011/HPL-2011-65.html, Sept. 2023.

[38] Kroft D. Lockup-free instruction fetch/prefetch cache organization. In *Proc. the 8th Annual Symposium on Computer Architecture*, May 1981, pp.81–87.

[39] Farkas K I, Jouppi N P. Complexity/performance tradeoffs with non-blocking loads. *ACM SIGARCH Computer Architecture News*, 1994, 22(2): 211–222. DOI: 10.1145/192007.192029.

[40] Tuck J, Ceze L, Torrellas J. Scalable cache miss handling for high memory-level parallelism. In *Proc. the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006, pp.409–422. DOI: 10.1109/MICRO.2006.44.

[41] Amaro E, Branner-Augmon C, Luo Z H, Ousterhout A,

Aguilera M K, Panda A, Ratnasamy S, Shenker S. Can far memory improve job throughput? In *Proc. the 15th European Conference on Computer Systems*, Apr. 2020, Article No. 14. DOI: 10.1145/3342195.3387522.

[42] Mars J, Tang L J, Hundt R, Skadron K, Soffa M L. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proc. the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2011, pp.248–259.

[43] Delimitrou C, Kozyrakis C. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Notices*, 2013, 48(4): 77–88. DOI: 10.1145/2499368.2451125.

[44] Liu Y H, Deng X, Zhou J P, Chen M Y, Bao Y G. Ah-Q: Quantifying and handling the interference within a datacenter from a system perspective. In *Proc. the 2023 IEEE International Symposium on High-Performance Computer Architecture*, Feb. 25–Mar. 1, 2023, pp.471–484. DOI: 10.1109/HPCA56546.2023.10071128.

[45] Nelson J, Holt B, Myers B, Briggs P, Ceze L, Kahan S, Oskin M. Latency-tolerant software distributed shared memory. In *Proc. the 2015 USENIX Conference on USENIX Annual Technical Conference*, July 2015, pp.291–305.

[46] Goodman J R, Woest P J. The Wisconsin Multicube: A new large-scale cache-coherent multiprocessor. In *Proc. the 15th Annual International Symposium on Computer Architecture*, May 30–June 2, 1988, pp.422–431. DOI: 10.1109/ISCA.1988.5253.

[47] Kuskin J, Ofelt D, Heinrich M, Heinlein J, Simoni R, Gharachorloo K, Chapin J, Nakahira D, Baxter J, Horowitz M, Gupta A, Rosenblum M, Hennessy J. The Stanford FLASH multiprocessor. *ACM SIGARCH Computer Architecture News*, 1994, 22(4): 302–313. DOI: 10.1145/192007.192056.

[48] Goodman J, Hum H H J. MESIF: A two-hop cache coherency protocol for point-to-point interconnects. Technical Report, University of Auckland, 2009. https://www.cs.auckland.ac.nz/~goodman/TechnicalReports/MESIF-2009.pdf. Setp. 2023.

[49] Kalia A, Kaminsky M, Andersen D G. Datacenter RPCs can be general and fast. In *Proc. the 16th USENIX Conference on Networked Systems Design and Implementation*, Feb. 2019.

[50] International Data Group. 2020 IDG cloud computing survey, 2020. https://cdn2.hubspot.net/hubfs/1624046/2020%20Cloud%20Computing%20executive%20summary_v2.pdf, Sept. 2023.

[51] Dally W J, Turakhia Y, Han S. Domain-specific hardware accelerators. *Communications of the ACM*, 2020, 63(7): 48–57. DOI: 10.1145/3361682.

[52] Esmaeilzadeh H, Blem E, Amant R S, Sankaralingam K, Burger D. Dark silicon and the end of multicore scaling. In *Proc. the 38th Annual International Symposium on Computer Architecture*, Jun. 2011, pp.365–376. DOI: 10.1145/2000064.2000108.

[53] Zaharia M, Chowdhury M, Franklin M J, Shenker S, Stoica I. Spark: Cluster computing with working sets. In *Proc. the 2nd USENIX Conference on Hot Topics in Cloud Computing*, Jun. 2010.

[54] Chen L, Zhao J C, Wang C X, Cao T, Zigman J, Volos H, Mutlu O, Lv F, Feng X B, Xu G H, Cui H M. Unified holistic memory management supporting multiple big data processing frameworks over hybrid memories. *ACM Trans. Computer Systems*, 2021, 39(1/2/3/4): Article No. 2. DOI: 10.1145/3511211.

[55] Tsai S Y, Zhang Y Y. LITE kernel RDMA support for datacenter applications. In *Proc. the 26th Symposium on Operating Systems Principles*, Oct. 2017, pp.306–324. DOI: 10.1145/3132747.3132762.

[56] McClure S, Ousterhout A, Shenker S, Ratnasamy S. Efficient scheduling policies for microsecond-scale tasks. In *Proc. the 19th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2022.

[57] Ma J Y, Sui X F, Sun N H, Li Y P, Yu Z H, Huang B W, Xu T N, Yao Z C, Chen Y, Wang H B, Zhang L X, Bao Y G. Supporting differentiated services in computers via programmable architecture for resourcing-on-demand (PARD). In *Proc. the 20th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2015, pp.131–143. DOI: 10.1145/2694344.2694382.

[58] Ziegler T, Tumkur Vani S, Binnig C, Fonseca R, Kraska T. Designing distributed tree-based index structures for fast RDMA-capable networks. In *Proc. the 2019 International Conference on Management of Data*, Jun. 2019, pp.741–758.

[59] Wang Q, Lu Y Y, Shu J W. Sherman: A write-optimized distributed B+tree index on disaggregated memory. In *Proc. the 2022 International Conference on Management of Data*, Jun. 2022, pp.1033–1048. DOI: 10.1145/3514221.3517824.

[60] Wei X D, Chen R, Chen H B. Fast RDMA-based ordered Key-Value store using remote learned cache. In *Proc. the 14th USENIX Symposium on Operating Systems Design and Implementation*, Nov. 2020, pp.117–135.

[61] Kraska T, Beutel A, Chi E H, Dean J, Polyzotis N. The case for learned index structures. In *Proc. the 2018 International Conference on Management of Data*, May 2018, pp.489–504. DOI: 10.1145/3183713.3196909.

[62] Gibson D, Hariharan H, Lance E, McLaren M, Montazeri B, Singh A, Wang S, H. Wassel H M G, Wu Z H, Yoo S, Balasubramanian R, Chandra P, Cutforth M, Cuy P, Decotigny D, Gautam R, Iriza A, Martin M M K, Roy R, Shen Z W, Tan M, Tang Y, Wong-Chan M, Zbiciak J, Vahdat A. Aquila: A unified, low-latency fabric for datacenter networks. In *Proc. the 19th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2022. pp.1249–1266.

[63] Xu Z W, Li C D. Low-entropy cloud computing systems. *SCIENTIA SINICA Informationis*, 2017, 47(9): 1149–1163. DOI: 10.1360/N112017-00069.

[64] Guo Z Y, Shan Y Z, Luo X H, Huang Y T, Zhang Y Y. Clio: A hardware-software co-designed disaggregated memory system. In *Proc. the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Feb. 2022, pp.417–433. DOI: 10.1145/3503222.3507762.

[65] Korolija D, Koutsoukos D, Keeton K, Taranov K,

Milojičić D, Alonso G. Farview: Disaggregated memory with operator off-loading for database engines. arXiv: 2106.07102, 2021. https://arxiv.org/abs/2106.07102, Oct. 2023.

[66] Sidler D, Wang Z K, Chiosa M, Kulkarni A, Alonso G. StRoM: Smart remote memory. In *Proc. the 15th European Conference on Computer Systems*, Apr. 2020, Article No. 29. DOI: 10.1145/3342195.3387519.

[67] Yoon W, Oh J, Ok J, Moon S, Kwon Y. DiLOS: Adding performance to paging-based memory disaggregation. In *Proc. the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*, Aug. 2021, pp.70–78. DOI: 10.1145/3476886.3477507.

[68] Lattner C, Adve V. Automatic pool allocation: Improving performance by controlling data structure layout in the heap. In *Proc. the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation*, Jun. 2005, pp.129–142.

[69] Akram S, Sartor J B, McKinley K S, Eeckhout L. Write-rationing garbage collection for hybrid memories. *ACM SIGPLAN Notices*, 2018, 53(4): 62–77. DOI: 10.1145/3296979.3192392.

[70] Larus J, Hunt G. The singularity system. *Communications of the ACM*, 2010, 53(8): 72–79. DOI: 10.1145/1787234.1787253.

[71] Lee Y, Al Maruf H, Chowdhury M, Cidon A, Shin K G. Hydra: Resilient and highly available remote memory. In *Proc. the 20th USENIX Conference on File and Storage Technologies*, Feb. 2022, pp.181–198.

[72] Chen S, Delimitrou C, Martínez J F. PARTIES: QoS-aware resource partitioning for multiple interactive services. In *Proc. the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr. 2019, pp.107–120. DOI: 10.1145/3297858.3304005.

[73] Delimitrou C, Kozyrakis C. Amdahl's law for tail latency. *Communications of the ACM*, 2018, 61(8): 65–72. DOI: 10.1145/3232559.

[74] Fried J, Ruan Z Y, Ousterhout A, Belay A. Caladan: Mitigating interference at microsecond timescales. In *Proc. the 14th USENIX Conference on Operating Systems Design and Implementation*, Nov. 2020, Article No. 16.

[75] Zhao J C, Feng X B, Cui H M, Yan Y L, Xue J L, Yang W S. An empirical model for predicting cross-core performance interference on multicore processors. In *Proc. the 22nd International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2013, pp.201–212. DOI: 10.1109/PACT.2013.6618817.

[76] Liu L, Li Y, Cui Z H, Bao Y G, Chen M Y, Wu C Y. Going vertical in memory management: Handling multiplicity by multi-policy. In *Proc. the 41st International Symposium on Computer Architecture*, Jun. 2014, pp.169–180. DOI: 10.1109/ISCA.2014.6853214.

[77] Hwang J, Vuppalapati M, Peter S, Agarwal R. Rearchitecting linux storage stack for µs latency and high throughput. In *Proc. the 15th USENIX Symposium on Operating Systems Design and Implementation*, July 2021, pp.113–128.

Chen-Xi Wang is currently an associate professor at the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing. His research focuses on building systems for emerging architectures. He got his Ph.D. degree in computer science and technology from the ICT, CAS, Beijing in 2018.

Yi-Zhou Shan is now a research scientist at Huawei Cloud, Shenzhen. He works on large-scale distributed storage systems, LLM serving, hardware resource disaggregation, etc. He got his Ph.D. degree from University of California San Diego, California, 2022.

Peng-Fei Zuo is currently a technical expert at Huawei Cloud, Shenzhen. His research interests include memory systems, storage systems, and distributed systems. He obtained his B.S. and Ph.D. degrees in computer science from Huazhong University of Science and Technology, Wuhan, in 2014 and 2019, respectively.

Hui-Min Cui is currently a professor at the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing. Her research interests include compiler optimizations, programming languages, and programming environments. She received her B.S. and M.S. degrees in computer science from Tsinghua University, Beijing, in 2001 and 2004, respectively, and her Ph.D. degree from ICT, CAS, Beijing, in 2012.