Research Statement

Chenxi Wang, Postdoc Researcher, UCLA
wangchenxi@g.ucla.edu

1 Research Overview: Semantics-aware Cloud Systems

I am interested in compute systems in general, and in particular, future cloud systems. Recently, my research focuses on developing runtime and operating systems for cloud applications running on emerging hardware.

Cloud computing is dominating the IT industry. Due to its high cost-efficiency, reliability, flexibility and other benefits, 81% of businesses are already using cloud technology in one capacity or another [1]. Cloud systems exhibit two clear trends: *(1) more diverse applications and (2) more heterogeneous architecture*. First, cloud workloads span a large number of domains (*e.g.*, ML, data processing, web service, *etc.*), with drastically different performance patterns and performance characteristics. Second, the innovation of the general-purpose computing technology is slowing down. A great number of domain-specific hardware architectures have been proposed to satisfy workloads with specific resource profiles. For example, *non-volatile memory* (NVM) aims to provide persistency, capacity, and energy efficiency for memory-intensive cloud applications, while *resource-disaggregation* segregates resources of different kinds into their dedicated servers to improve source utilization.

Although such hardware devices provide numerous benefits to cloud providers, it is difficult for user applications to see clear gains. For example, applications must have good data locality to efficiently run on a resource-disaggregated cloud (where data are physically separated from compute), while many kinds of cloud applications do not exhibit locality (*e.g.*, graph workloads). Consequently, when such workloads run on NVM or disaggregated memory *as is*, significant performance degradation can result. My research tackles this problem head on, **building cloud systems that align the behavior of high-level workloads and the characteristics of low-level architecture with *application semantics***.

2 Past Research

2.1 Panthera: Semantics-aware Runtime for Hybrid Memory [PLDI'19]

Non-volatile memory (NVM) was developed to satisfy the rapidly increasing memory demand of data-intensive applications. Compared to DRAM, NVM offers much larger capacity, lower price, and higher energy efficiency. On the downside, NVM has longer access latency and lower read/write bandwidth. To maximize NVM's benefits and minimize its performance impact, cloud infrastructures often deploy NVM together with DRAM in a hybrid manner. To effectively use hybrid memory, one should place hot, frequently-accessed data into DRAM (for faster accesses) and cold, infrequently-accessed data into NVM (to utilize its capacity and energy efficiency). Consequently, a major challenge here is how to monitor data access patterns and properly migrate data between DRAM and NVM to achieve superior efficiency in both time and energy.

The conventional technique is to rely on the OS' swap system to profile data access patterns and migrate frequently-accessed data into DRAM at the fine-grained, *page* granularity. However, the overhead of page-level monitoring and migration is often high, reaching up to 25% of application execution time, and increases as the memory usage goes up [3].

To minimize this overhead, we develop Panthera [4], a new runtime system that can efficiently tracks and moves data at a coarse granularity. Panthera builds on the following two important insights. First, many cloud applications perform bulk object creation, and data objects exhibit strong epochal behavior and clear access patterns at a coarse granularity. For example, Apache Spark manages data using resilient distributed dataset (RDD); TensorFlow manages data as tensors. Leveraging such user-defined, coarse-grained data abstractions (*i.e.*, framework semantics) can enable accurate data access monitoring with a much lower overhead. These data abstractions can be easily obtained through types (such as RDD) and static compiler analysis. Second, most cloud applications are written in managed languages that run on top of a managed runtime. The runtime performs periodical garbage collection (GC) that already moves objects to compact the heap. As such, GC provides natural opportunities to migrate hot/cold objects without incurring extra overhead. Furthermore, the runtime system manages the program execution and hence has direct access to the aforementioned coarse-grained information about data abstractions.

To leverage the first insight, Panthera performs a simple static analysis to identify accesses to user-defined data abstractions (such as RDDs) and communicates such information down to the runtime system. To leverage the second insight, Panthera uses a redesigned GC to migrate objects based on their coarse-grained access statistics (*e.g.*, whether they belong to a hot or cold RDD). An evaluation with a suite of real-world Spark applications demonstrates that *Panthera* can accurately classify and place objects between DRAM and NVM with **less than 1% runtime overhead**, and reduce energy consumption by **up to 52%**.

2.2 Semantics-aware Runtime for Disaggregated Memory [OSDI'20, PLDI'22-sub, OSDI'22-sub]

As an emerging datacenter architecture, *resource-disaggregation* aims to reorganize datacenter hardware of each kind into their dedicated resource servers to improve resource utilization and fault tolerance, and simplify hardware adoption. These servers are connected by advanced network fabrics such as RDMA over Infiniband, making it possible for programs running in one (CPU) server to use data located in another (memory) server. Despite its benefits to cloud providers, disaggregation creates a performance impact on cloud applications—particularly those without good spatial/temporal locality because most of their memory accesses trigger remote fetching, incurring a large performance penalty.

Unfortunately, many cloud applications exhibit poor locality and hence will suffer greatly from disaggregation. This is particularly the case with programs written in managed languages. Such programs run frequent GC activities, which traverse a heap graph to identify and move live objects. Even if the program itself has good locality, it can often be messed up when GC runs. This is because modern GCs often run concurrently with application threads (for performance). Under memory disaggregation, however, each program can only use a very small amount of local memory (on the CPU server) as cache space. As such, GC and application threads frequently compete for memory resources, leading to severe interference. For example, due to their disjoint working sets, when the application (or GC) needs space, it evicts objects needed by GC (or the application), resulting in frequent thrashing.

To solve the problem, I developed a series of novel runtime systems that can efficiently run managed cloud applications on a resource-disaggregated architecture. The first work was Semeru (OSDI'20 [5]), a new managed runtime that *offloads GC's tracing task onto memory servers*. A modern GC has two major phases: tracing and reclamation. Tracing traverses the object graph to identify live objects and reclamation frees the space taken by dead objects. By offloading tracing onto memory servers, Semeru physically separates GC tasks from application threads, reducing their interference. Furthermore, tracing runs near data; hence, poor locality is not a concern and its performance can be significantly improved. With a thorough evaluation over a set of widely-deployed cloud applications, we show that Semeru can improve their end-to-end performance by a factor of $3\times$.

Although Semeru offloads the tracing task, memory reclamation is still performed on the CPU server in a long stop-the-world pause. Since many cloud workloads are latency-sensitive (*i.e.*, they have strict service-level objectives), such pauses are often intolerable. Although there are a number of low-pause GCs that can perform concurrent object reclamation (without incurring a long pause), these GCs are not designed for a distributed setting where memory is disaggregated. As the second work, our team developed a new concurrent GC called Mako [2] that offloads both tracing and reclamation onto memory servers. With a set of carefully designed coordination algorithms, Mako outperforms Shenandoah, a commercial concurrent GC in the HotSpot JVM, by $\mathbf{4}\times$ in pause time and $\mathbf{3}\times$ in throughput.

While Semeru and Mako improve GC performance, they rely on an assumption that memory servers are equipped with sufficient compute power so that GC tasks can be offloaded and efficiently executed by memory servers. However, in real-world datacenters, memory servers may carry wimpy cores or even only accelerators, making offloading an impractical approach. As the third work on this topic, I developed MemLiner [6], a novel runtime and kernel co-design that can significantly reduce the application-GC interference without performing any offloading. Memliner achieves this by *aligning memory accesses of GC and application threads*: we adjust the order of object tracing in GC so that tracing starts from those objects that were recently accessed by the application (and are likely still in cache); for objects that have not been accessed for a long time and likely have been evicted out of the CPU server, we delay their tracing (by a bounded period of time). By aligning their memory access paths, MemLiner successfully shrinks the application's working set, reducing the interference and simultaneously making the application's access patterns more clear for an underlying OS prefetcher to identify (so as to improve prefetching accuracy).

2.3 Semantics-aware Swap System for Disaggregated Memory [NSDI'23-revision]

At the operating system level, applications running on top of disaggregated memory often use the paging and swap system to fetch data from remote memory. However, the current swap system was designed long ago for slow disk-based swapping, and hence can become a major performance bottleneck when fast remote memory is used. In particular, we found two major problems with the swap system design: **(1)**

lack of performance isolation and **(2) poor scalability**. When multiple applications co-run on the CPU server, sharing the same swap system of the OS leads to significant interference and performance degradation. We conducted a simple experiment with three native applications and Apache Spark. Running them all together for multiple times, we observed large performance variations across different runs (*e.g.*, the running time of the same application can be **3.9**× apart in different co-runs). Co-running also results in up to **6.4**× slowdown compared to running each application individually, as well as dramatically reduced RDMA throughput. The reason is that applications that generate high swap throughput *aggressively invade* the (RDMA and swap cache) resources of other applications that have less frequent swap needs. This is fundamentally due to the lack of resource isolation in the current swap system.

To solve the problem, I developed *Canvas* [7], a redesigned swap system with holistic resource isolation. In particular, Canvas isolates all kinds of swap resources including swap partition (*i.e.*, remote memory), swap cache, and RDMA bandwidth prevent applications from invading others' resources, eliminating interference and reducing performance variations. Furthermore, isolation enables the kernel to use different policies and optimization strategies for different applications based on their semantics and resource needs. For example, Canvas uses adaptive locking for applications to efficiently allocate swap entries, leverages runtime support to perform effective semantics-aware prefetching, and schedules prefetching and on-demand swap-in requests in response to each application's unique remote access behavior. These semantics-aware, application-specific optimizations bring a further boost to applications' performance.

An evaluation with a wide spectrum of applications including machine learning frameworks, key-value databases, and data analytics systems demonstrates that *Canvas* reduces performance variation by **31**× and improves throughput by up to **6**×.

3 Future Research

I will continue to work on building datacenter infrastructures. In particular, my future work will center around two main research axes: *(1) improve the customizability of the cloud front-end* and *(2) improve the cost-efficiency of the cloud backend*.

3.1 The Customizable Cloud

As the needs of cloud users are becoming increasingly diverse, we envision that the next cloud should enable a resource-centric cloud-user interface where the cloud provides users flexibility to customize services by assembling the hardware and software resources they need. For example, grocery stores and hospitals have dramatically different resource needs *e.g.*, in their workload types and security and privacy requirements. From the user's perspective, these needs can be hardly satisfied at the same time by existing services provided by today's cloud. From the provider's perspective, it is extremely difficult to predict the diverse resource demands of users to pre-allocate the Virtual Machine instances with proper software and hardware resources for a timely instance allocation. To make matters worse, the needs of users can change dynamically. For example, users may need GPU resources occasionally. Under today's cloud, they must reserve GPU resources from the beginning, paying a high price for a minor usage.

My plan is to build a *user-customizable cloud* where users see the cloud as a collection of resources. They pick the resources they need and the cloud provider provides system software that can effectively manage these resources. This can be done at two distinct levels. At the *system layer*, the compute hardware resources should be disaggregated and abstracted as dedicated resource process pools, *e.g.*, X86/ARM CPU process pool, GPU process pool, accelerator process pool *etc.* Under this resource abstraction, the cloud can timely allocate any kind of hardware resources and schedule the compute tasks to these resource processes according to the needs of users. Each operation will involve a bunch of compute processes with dedicated memory and storage resources.

At the *programming layer*, the cloud should provide a better abstraction for cloud users to describe their resource requirements or program semantics. Users know program semantics while the cloud provider better understands the correlation between tasks and software/hardware resources. For example, if the user specifies that the tasks are Garbage Collection (GC) tasks or Machine Learning tasks, the cloud can schedule these compute task to the proper resources accurately. Hence, we can let users build a task DAG and specify the program semantics of each node (task), *e.g.*, whether it is a memory-intensive, compute-intensive, or compute with high privacy needs. Based on these semantics, the cloud OS should choose the proper hardware resources and schedule these tasks to the corresponding processes pool.

3.2 Improve the cost-efficiency of the cloud backend

Another important problem I plan to work on is how to improve the performance-per-dollar of the cloud backend by significantly improving its resource utilization. I will achieve this goal with the following

research activities:

Predictable Latency for Memory Disaggregation. Although *memory disaggregation* can improve the resources utilization of the datacenter, it brings about performance challenges, which can result in service level objective (SLO) violations. For example, applications running on a memory-disaggregated cluster often experience unpredictable pause times due to remote fetching and unpredictable network latency. This is because existing disaggregated memory platforms rely on the OS and hardware to automatically manage data fetching. As a result, the frequency and length of fetching-induced pauses are beyond what users can control. I plan to work to a framework with runtime, OS, and programming language support that allows users to program an application into two distinct paths: one that is not latency sensitive and can go through the kernel's original data path and a second that is latency sensitive and there must be a fetching-time guarantee for using remote memory.

Lightweight Virtual Machine. Cloud computing often utilizes virtual machines to provide strong performance and function isolation for applications. However, many of these isolation mechanisms are already provided by a language virtual machine such as JVM. I plan to investigate the possibility of unifying an OS virtual machine with a language runtime, producing a unikernel-like design in which one (JVM, Python, *etc.*) single process runs directly in the kernel mode. Such a design can significantly reduce the time for launching a VM and performing sys calls, offering large performance gains to lightweight cloud workloads such as serverless functions—one of the most popular cloud services today.

References

[1] I. D. Group. Cloud computing study. https://www.idg.com/tools-for-marketers/2020-cloud-computing-study/, 2020.

[2] H. Ma, S. Liu, C. Wang, Y. Qiao, M. D. Bond, S. Blackburn, M. Kim, and G. H. Xu. Mako: A low-pause, high-throughput evacuating collector for memory-disaggregated datacenters. In *Submission for PLDI'22*.

[3] C. Wang, T. Cao, J. Zigman, F. Lv, Y. Zhang, and X. Feng. Efficient management for hybrid memory in managed language runtime. In *Network and Parallel Computing*, pages 29–42. Springer International Publishing, 2016.

[4] C. Wang, H. Cui, T. Cao, J. Zigman, H. Volos, O. Mutlu, F. Lv, X. Feng, and G. H. Xu. Panthera: Holistic memory management for big data processing over hybrid memories. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 19)*, page 347–362.

[5] C. Wang, H. Ma, S. Liu, Y. Li, Z. Ruan, K. Nguyen, M. D. Bond, R. Netravali, M. Kim, and G. H. Xu. Semeru: A memory-disaggregated managed runtime. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 261–280. USENIX Association, Nov. 2020.

[6] C. Wang*, H. Ma*(co-first), S. Liu, Y. Qiao, J. Eyolfson, C. Navasca, S. Lu, and G. H. Xu. Memliner: Lining up tracing and application for a far-memory-friendly runtime. In *Submission for OSDI'22*.

[7] C. Wang*, Y. Qiao*(co-first), H. Ma, S. Liu, Y. Zhang, W. Chen, R. Netravali, M. Kim, and G. H. Xu. Canvas: Isolated and adaptive swapping for multi-applications on remote memory. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)- One-Shot Revision and Resubmit to NSDI'23 Srping*, 2023.