Limitations of Disaggrated Memory and Innovations

Zhantong Qiu
Department of Computer Science
University of California, Davis
Davis, USA
ztqiu@ucdavis.edu

Ikhyeon Kwon
Department of Electrical and Computer Engineering
University of California, Davis
Davis, USA
ihkwon@ucdavis.edu

Amaan Mohammed
Department of Electrical and Computer Engineering
University of California, Davis
Davis, USA
aamohammed@ucdavis.edu

***Abstract*— Data centers have low memory utilization issues. Servers can be over-allocated memory to address worst-case scenarios. To reduce the amount of unused memory that can be shared between servers, memory disaggregation is one of the most promising solutions. Memory disaggregation separates computational elements from memory resources, allowing each to be provisioned and utilized separately. Due to the shared memory between different servers, an efficient page management mechanism is required. In this paper, we propose a hardware software co-design system to improve on top of the current solutions.**

Keywords*— *Data Centers, Memory Disaggregation, TPP, CXL, Memory Management Mechanism. Tiered-Memory, CXL-Memory.

I. Introduction

The demand for memory in the global market continues to increase due to the growth of large-scale computing, big-data processing, data-centric workloads and artificial intelligence technologies. However, the expansion of memory bandwidth and capacity is facing a bottleneck due to technical and cost limitations, and the imbalance between supply and demand of computing and memory resources per node has worsen. This imbalance continues to increase every year. The solution is to separate compute and memory resources called memory disaggregation. Figure 1(a) and (b) shows the architectures of memory disaggregation. It allows to share memory resources between different servers. Sharing memory between different servers across server boundaries requires a peripheral component interconnect-express (PCIe)-based new dynamic multi-protocol called CXL. CXL is a way to achieve optimized performance by considering latency and bandwidth between memory and communication technology. In Figure 1(c), the additional memory bandwidth of CXL can improve the access speed between processors and disaggregated memory nodes [1].

However, CXL memory disaggregation still has some challenges that should be solved for high performance memory system. In particular, effective memory allocation techniques are required. As the number of memory components and applications increases in large clusters, it is more complicated to find unallocated remote memory and to find efficient matching at scale without violating application specific SLOs (Service Level Objectives). Additionally, memory disaggregation has different layers of memory that have different performances and access time. Figure 1(c) shows different latency characteristics and a heterogeneous memory set up. Therefore, simply allocating memory to applications is not enough. It is necessary to consider how much memory to allocate in which tier at what time is critical.

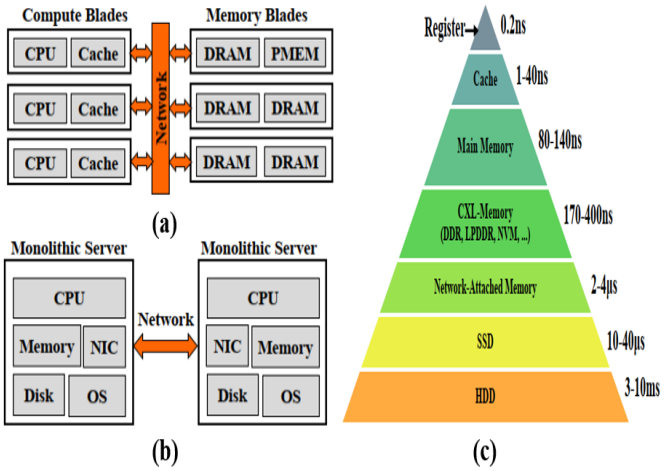

Fig. 1. (a) Physical memory disaggregation architecture, (b) logical memory disaggregation architecture, and (c) Latency characteristics of different memory

For efficient memory management, some software support techniques have been investigated and suggested. Non-uniform memory access (NUMA) is a mechanism designed to allow a processor to access its own local memory [2]. It has faster access time than non-local memory. Therefore, in NUMA architecture, fast access to local memory contributes to improved performance, and processing data from local memory is ideal. When accessing non-local memory, it must consider optimized memory usage patterns due to the additional delay and performance penalty. It also has additional traffic due to the promotions that do not consider the active state of the page. This is called the ping-pong issue.

Auto Tiering has the ability to automatically move data to the appropriate storage tier based on the data's access pattern, frequency, and importance. However, with a tight allocation-reclaim path, it uses a buffer of a fixed size for promotion operations, and this buffer maintains the fixed size even under pressure. This causes traffics on CXL node [3].

TPP is an OS-level application transparent page placement mechanism for CXL-enabled memory. It has shown the following key features. TPP strategically places 'hotter' pages (frequently accessed) in local memory and relocates 'colder' pages (infrequently accessed) to CXL-Memory, which optimize the memory performance. This page placement considers the application's memory access patterns, ensuring that frequently used data remains in local memory for quick response times, while less used data is moved to CXL-memory to efficiently utilize the overall memory. Also, TPP minimizes performance degradation by addressing the demotion to CXL-Memory with lightweight operations. By

separating allocation and reclamation paths, it can enhance the resource management efficiency. Hot page promotion to local nodes allows to achieve enhanced performance. As a result, these features contribute to TPP outperforming the traditional mechanisms such as NUMA balancing and Auto Tiering. It has demonstrated to outperform the performance of two state-of-the-art tiered-memory management mechanisms by 5–17 %. Also, TPP enhances the performance of applications on the default Linux environment by 18%. However, TPP uses least recently used (LRU) algorithm to identify hot pages. LRU algorithm operates only based on the recent usage history of pages and it cannot consider the past access patterns. Additional hardware support is required to more accurately identify hot pages [4].

In this paper, we expect improved CXL disaggregated memory performance by applying hardware-software co-designed page prefetching technology to TPP. In detail, we expect efficient page promotion/demotion management through more accurate hot page detection.

II. CXL-Disaggregated Memory: Open Challenges

Although CXL allows memory disaggregation to overcome the existing limitations in memory management and application, it still requires more organized memory management and allocation techniques. To realize more efficient data center operation, the following challenges must be considered.

A. Latency and Bandwidth

CXL disaggregated memory requires bandwidth to move and share data. The high latency of CXL remains a challenge, especially in large rack-scale systems. In large-scale systems, effective latency and bandwidth management are critical factors in determining performance.

There are two main reasons why CXL disaggregated memory remains challenging at rack-scale. First, CXL has relatively high latency in moving data between remote nodes. Especially in rack-scale systems, data can take longer to move between multiple nodes. This can result in increased latency and degraded performance for applications. Second, rack-scale systems require very large bandwidth to process large amounts of data. In CXL disaggregated memory, it is important to effectively manage the bandwidth. In particular, effectively utilizing and distributing the bandwidth of the entire system is a very complicated challenge, especially when moving data between a large number of nodes. For these reasons, CXL disaggregation in rack-scale systems still has high latency and bandwidth management challenges. It is important to find effective solutions to these challenges through research and technological advancements.

B. Network Congestion

Network Congestion presents a situation where data transmission is delayed or bottlenecked on a network. This can occur in situations where multiple applications or nodes are competing for network resources.

Detailed explanation is as follows. When multiple applications or nodes use the network simultaneously, the amount of data and transmission requests increases. This can lead to situations where network bandwidth is limited or network equipment is unable to handle excessive data. In terms of bandwidth, networks have a certain bandwidth, and if the amount of data exceeds this limit is transmitted, a bottleneck occurs. Bandwidth can be a significant constraint, when large amounts of data must be transferred, such as memory disaggregation. These network bottlenecks requires advanced network management and optimization techniques.

C. Memory Coherency

Memory coherency is a key challenge in disaggregated memory architectures. Memory coherency means multiple replicas of the same data should always have the same values. In disaggregated memory, maintaining this coherency is important because data is spread across multiple nodes. Data that changes on one node must be consistently updated on other nodes. To achieve this, various algorithms are used such as distributed transactions, consistency protocols, and data synchronization mechanisms. These technologies are applied to coordinate data changes, updates and coherency. Memory coherency is a key aspect for building stable and reliable disaggregated systems in data center environments, and is an especially important consideration in modern memory architectures such as CXL disaggregated memory.

D. Data Security

Memory disaggregation can raise concerns about data security and privacy. Memory disaggregation requires data to be transmitted via networks, which implies the possibility that the data can be intercepted or manipulated. While data is being transmitted externally, its confidentiality and integrity can be exposed to risk. Therefore, it requires security protocols to ensure safe data transmission. This includes encryption of data, secure communication channels, and reliable authentication mechanisms. Also, it is important to control data access and manage permissions. Strict access control and permission management are required to prevent illegal access and unauthorized use of data. Therefore, when implementing a memory disaggregation system, these security considerations must be carefully considered, and secure protocols and mechanisms must be introduced to ensure data safety.

E. Memory Efficiency and Scalability

As the number of memory and applications increases, large clusters require to find unallocated remote memory and find efficient matches without violating application-specific service level objectives (SLOs). Large data center environments have hundreds or thousands of servers and applications. This causes a rapid increase in the amount of memory and number of applications that must be managed. According to the increased memory, resource allocation and management must be optimized. It requires efficient algorithms, mechanisms, and hardware supports.

III. Related Work and Limitations

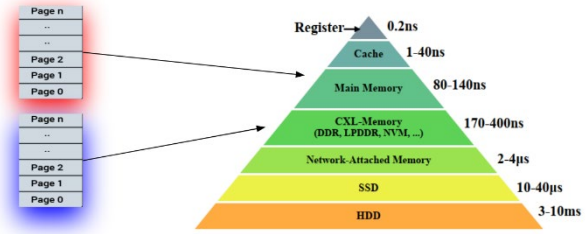

Fig.2 Desired page temperature organization of memory hierarchy with CXL memory

In seeking to mitigate the drawbacks posed by memory disaggregation, this paper takes a look at smart memory management mechanisms to utilize tiers of the memory

hierarchy effectively. The aim is to sort pages based on their frequency of use/access or "temperature". An illustration of the hierarchy can be seen in Fig. 2. As depicted in the figure, for an ideal scenario, colder pages would be demoted to remote memory and hotter pages would be promoted to local memory.

However, before discussing appropriate management mechanisms, one must consider the technology that enables this memory hierarchy with remote memory. Compute Express Link (CXL) is a cache-coherent interconnect that supports memory-pooling between disaggregated memory nodes. CXL reduces the latency associated with remote memory by connecting the remote memory and CPU with a new memory bus interface. This enables DRAM-like bandwidth and lower latency, allowing remote memory to appear to the system as main memory. The improvements provided by CXL facilitates the implementation of low latency remote memory within large memory systems' memory hierarchies. While latency does become lower, remote memory still is not as fast as local memory (ex. DRAM). Thus, this paper's focus on smarter memory management becomes relevant.

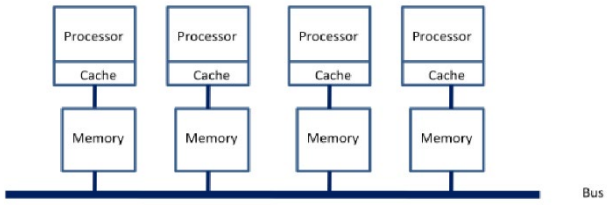

Fig.3 Example of NUMA platform structure

A. NUMA Balancing

One such memory management technique is NUMA Balancing. NUMA Balancing's original aim was to minimize memory access with NUMA nodes. NUMA or non-uniform memory access is a platform design that uses smaller, cost-effective, modules to build computer systems. These modules can contain memory, controllers, I/O and/or processors. An example of such a structure is shown in Fig. 3. For this discussion, what's important to note is that NUMA Balancing was not designed for use with remote memory. Rather, its main purpose is to enable multi-core processors with their own local memories to access other nodes' memories over a shared bus with varying access times. A behavior that carries over from this design is the NUMA hint fault [4]. Within NUMA Balancing, the kernel routinely checks a 256MB subset of process memory on each node. "When a CPU accesses a sampled page, a minor page-fault is generated (known as NUMA hint fault). Pages that are accessed from a remote CPU are migrated to that CPU's local memory node (known as promotion) [4]" As explained above, pages found in a memory not local to the relevant process are automatically promoted to local memory by way of the NUMA hint fault. This instant promotion occurs without checking the page's active state within software. This behavior is one of many that causes limitations when NUMA Balancing is used to manage memory disaggregation.

When a page is instantly promoted without a check of the PageActive list, a "ping-pong" issue can occur. Pages with infrequent accesses can become instantly promoted to the local node. Shortly after, these pages can become demoted if local nodes are under stress for usage. This cycle can occur again and again and create unnecessary traffic, deteriorating application performance. Furthermore, when local memory fills up and promotions from remote memory are attempted, high overhead from these failed promotions can result [4]. Lastly, the routine sampling of process memory can also degrade performance due the constant sampling of highly accessed pages.

B. Transparent Page Placement

Transparent page placement (TPP) [4] is a new OS-managed mechanism that sorts pages in memory based on their usage temperature. The goal of TPP is to effectively sort hotter pages in local memory and colder pages in CXL memory.

TPP overcomes the limitations of NUMA-Balancing and Auto-Tiering by implementing a lightweight reclamation mechanism for demotion of cold pages, decoupling allocation and reclamation logic, allowing dynamic configuration on the reclamation's aggressiveness, the addition of a page promotion mechanism that promotes hot pages to local memory nodes, reducing the overhead of page temperature detection, and providing page type-aware allocation.

In order to speed up the reclamation, TPP proposes a new reclamation mechanism for local memory reclamation. Instead of using the swapping mechanism, TPP uses a demotion list to store the reclamation candidates and migrate them to the CXL nodes asynchronously. It is an order of magnitude faster than the swapping mechanism. TPP selects reclamation candidates from inactive anonymous pages and decreases the chances of demoting hot pages from local memory.

To utilize the local memory and increase local memory allocation performance, TPP decouples the allocation and reclamation. It introduces two new watermarks, demotion_watermark and allocation_watermark. Reclamation happens in the background until the count of the free pages reaches the demotion_watermark while new allocation can happen if the count of the free pages satisfies the allocation_watermark [4]. This way reduces the halts in memory allocation and improves the performance by allocating more new pages in the local nodes than in the CXL nodes. TPP allows users to set the threshold of the aggressiveness of reclamation in local nodes with the sysctl interface. It is important because in some scenarios, such as running applications with frequently accessed pages that are bigger than the local node's capacity, high aggressiveness of reclamation will thrash hot memory across NUMA nodes.

With the decoupling feature enabled, large bursts in memory allocation are provided space in local memory. The decoupling from TPP enables 1.6x more in allocation rate without being limited by reclamation. Additionally, the decoupling enables higher levels of hot-page promotion. Trapped cold pages in CXL memory can cause up to 55% of memory traffic leading to a 12% throughput drop [4]. The decoupling ensures consistent promotion of cold pages and maintenance of throughput. With decoupling enabled, promotion rates remain consistently above 10 KB/s.

Unlike NUMA balancing, TPP only uses the generated minor page fault mechanism in CXL nodes to assess page

temperatures (in local nodes "least-recently used" is used for temperature assessments). This prevents the high overhead of minor page fault generation from "warm" pages. Thus, the local node is protected from the addition of pages that may not have an appropriate "active state" to be placed in local memory. The removal of the minor page fault mechanism in CXL nodes is justified because it is assumed the CXL node will not contain "hot" pages.

TPP enjoys a 42x faster reclamation rate than NUMA balancing, an 11x faster promotion rate, and a lower CPU overhead in comparison to NUMA Balancing in the Web1 workload [4]. Against AutoTiering, TPP succeeds in maintaining throughput (13% drop for AutoTiering vs 0.5% drop for TPP), during surges of CXL-node page accesses [4]. These stats show how TPP is quantitatively more effective than existing solutions such as NUMA Balancing and AutoTiering.

Though TPP can bring benefits over NUMA_balancing such as novel migration, decoupling, and higher reclamation, it too can pose certain challenges. Firstly, if processes have differing Quality-of-Service priorities, TPP cannot provide optimal performance [4].

Another more important limitation to note is TPP's use of an LRU algorithm for hot pages [4]. This algorithm restricts TPP's operation to be based only on recent usage history of pages rather than historical access patterns. Additional ideas to more accurately identify hot pages and potential hot page patterns becomes necessary.

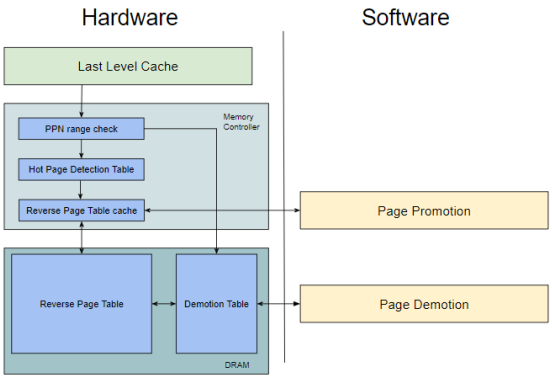

Fig. **4**. Diagram of our plan for improvement

IV. PLAN FOR IMPROVEMENT

We believe a hardware-software co-designed page management is the solution to the limitations of the current solutions.

A. Hardware

1) Inspiration

The hardware side of our plan for improvement is largely inspired by the *HoPP*[5] prefetcher. *HoPP* is a hardware-software co-designed page prefetching framework proposed for disaggregated memory. Instead of using page faults to detect page temperature, they proposed to use the last level cache misses to detect page temperature. We will focus on analyzing their hardware modules.

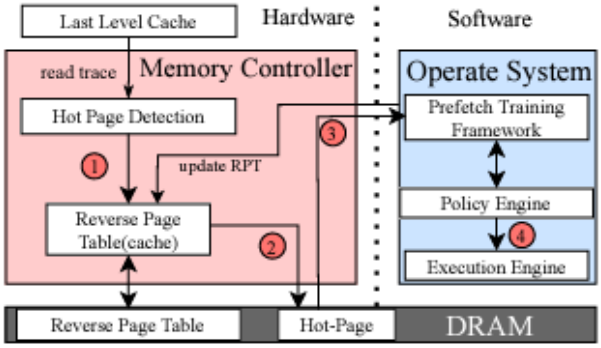

Fig.5 . Design of *HoPP [5]*

HoPP proposes to place a light-weighted Hot Page Detection Table in the memory controller to track the access frequency of the physical page numbers from the last level cache and use the information to find hot pages. Their design of the Hot Page Detection Table is a 16-way 4-set associative cache with LRU replacement policy. Each entry stores the PPN, an access number, a send bit, and a LRU bit. When the access number of a PPN exceeds a threshold N, the page is marked as a hot page. It then sets the send bit to stop future increasement on the access number. In the *HoPP* paper, they analyze the impact of the threshold N on the number of hot pages extracted. The smaller the threshold N is, the more hot pages are extracted, however, it also means more repeated extraction of the same page and causing more memory bandwidth and potentially harming the application performance. The larger the threshold N is, the chance of evicting a hot page before reaching the threshold gets higher. *HoPP* chooses N=8 as their default threshold.

The other important hardware module proposed by *HoPP* is the Reverse Page Table. Since the prefetching is handled by the software and memory controller tracks physical addresses, *HoPP* needs to translate the physical page number of the hot page to virtual page number and links the page with its process ID. The translation process requires page table walks so it is an expensive process. To reduce the cost of translation, *HoPP* reserves a space in the DRAM to map the PPN to VPN and PID. They call it a Reverse Page Table. They also put a small Reverse Page Table cache inside the memory controller. All operations only interact with the Reverse Page Table cache. They found that with 4KB pages, 64GB local memory requires a 112MB Reverse Page Table, which is only 112MB/64GB = 0.17% of the physical memory.

2) Our plan

With the inspiration from *HoPP*, we designed a hardware page temperature detection module that impacts both page promotion and demotion.

In the memory controller, we mostly used the design of *HoPP* beside having an extra PPN range check unit. When a physical address passed from the last level cache, we break its physical page number out and check if it is in the range of the memory in the local node. If it is, we send the PPN to a buffer and write the access to the Demotion Table. If it is outside of the local node's memory range, we pass the PPN to the Hot Page Detection Table. The Hot Page Detection Table is the same as the one in the *HoPP* design, which stores the PPN, the access number, the send bit, and the LRU bit.

We also use a threshold N to determine if a page is hot or not. The Reverse Page Table cache is also the same as the *HoPP* proposed. When we get the VPN and PID with the hot page's PPN, we send it to the software for page promotion.

In the DRAM, we have a Reverse Page Table and a Demotion Table. The hardware design of *HoPP* is able to detect hot pages but it is not able to find cold pages for demotion. Demotion is an important part in page management, so we designed a Demotion Table for cold pages detection.

The Demotion Table is built almost the same way as the Reverse Page Table. When the software starts, it traverses the local page table and stores all the existing pages' PPN and an access number of 0. The access number of a PPN increases when it is missed in the last level cache. All the pages with an access number below a threshold M are considered as cold pages. When the software reaches the reclamation watermark and triggers reclamation, the demotion candidate list is formed using the cold pages in the Demotion Table. The reverse address translation is still done with the Reverse Page Table. When the reclamation is finished, we remove the demoted PPN's entry and refresh the access frequency information. In order to track all the existing pages in the local nodes, we also need to add the new PPN for every new page allocation in the local node.

3) Discussion

The Demotion Table is our most concerning hardware module in our plan. The Demotion Table will add extra memory bandwidth that potentially harms the application performance on top of the extra memory bandwidth produced by accessing the Reverse Page Table. It also means extra area reserved and extra power consumption. Beside all of these, it also adds extra steps to page allocation, reclamation, and the critical path in the memory controller. Will the benefit getting from demoting the cold pages overcome the cost of maintaining the Demotion Table?

In order to find the answer, we need insights about the performance gain from demoting the cold pages and the actual cost of maintaining the Demotion Table.

The other question we have for our plan is that we should use 64 entries as *HoPP* designed for our Hot Page Detection Table? The range of the last level cache misses in general datacenter applications is the key to answer our question.

We will talk about the experiments we need to find the answers for the questions and concerns we have in the plan for evaluation section.

B. Software

1) Our plan

We plan to add page temperature clustering to the page management software to predict the promotion and demotion candidates.

Page migration has high latency. It is possible for a hot page to become a cold page after the promotion, or a cold page becomes a hot page after the demotion. To prevent this ping-pong issue and hide the high latency, we can cluster and predict the next list of promotion and demotion candidates through the promotion and demotion history.

Ideally, once we identify the clusters and their patterns, we can promote and demote groups of pages according to their temperature pattern.

We are planning to use a simple table that is similar to the Branch Target Buffer as our first testing algorithm. Instead of recording the branch program counter and the predicted program counter, we can record the promoted/demoted VPN and the most frequently followed promoted/demoted VPN.

2) Disucssion

Because of a lack of time, we do not have enough time to look at current research on prefetching algorithms and clustering algorithms. We believe the idea of finding the page temperature patterns and clustering the pages with similar patterns are important for improving the current page management software.

V. PLAN FOR EVALUATION

A. Performance Evaluation of the Demotion Table

In order to find the performance gain from demoting the correct cold pages, we will first create a hypothetical system that ensures all demotion is demoting a cold page.

We will create a system with a new module connected to the last level cache in the gem5 simulator, a computer architecture simulator that is largely used by both industry and academia, and maps all the existing pages in the local node to a list of counters. As proposed in our plan for hardware improvement,, we will increment the counter every time the page is missed in the last level cache. When the OS processes reclamation, it has to pick the page with the lowest access frequency in the new module. In this experiment, we will ignore the cost of having such a module, and compare the performance with a system without this module. This should give us an idea of the maximum performance gain of demoting only cold pages in a full-system setting.

Then, we will evaluate the cost of having such a table by adding the latency on building, updating, and maintaining the table. This experiment should give us an answer of if the Demotion Table is an improvement or not.

B. Hot Page Detection Table size

In this experiment, we will trace the PPN of the last level cache misses for every set of time frames and get the most frequent PPN range. We will gather this data for multiple general benchmark suites to find the best Hot Page Detection Table size for general cases.

C. Page Temperature Pattern and Clustering

In this experiment, we will use the memory access trace of the last level cache to find the connection between PPNs and the possible signature that can identify these connections. If we can find the signature, we can apply it to our software and identify the pattern and clusters in real-time.If we can not find any signature, it means that we might be unable to identify and predict the page temperature pattern.

D. The Overall Evaluation of Our Plan for Improvement

Since our plan for improvement has two different aspects, hardware and software, we do three different experiments to evaluate the overall plan for improvement.

1) Hardware

TPP with our hardware	TPP without our hardware
NUMA Balancing with our hardware	NUMA Balancing without our hardware

Table 2. Performance evaluation system settings

To evaluate our hardware, we will need two system configurations: a system with our hardware modules and a system without our hardware modules. We also need to enable the Linux kernel to connect and communicate with our hardware modules. Because *TPP* does not open the benchmarks they use for evaluation, we can not use their evaluation result as our baseline. Therefore, we will test our hardware module with both *TPP* and *NUMA-Balancing*, and use the evaluation result of the *NUMA-Balancing* without using our hardware modules as the baseline. We will select multiple macro-benchmark suites to represent the general applications in the data center and use customized micro-benchmarks to test the theoretical and corner cases.
We will use the gem5 simulator for the full-system performance evaluations. We will use CACTI to estimate the area and power consumption gain of our hardware modules.

2) Software

For the software part, we will add our page temperature prediction and clustering algorithm inside the Linux kernel with *TPP* and compare it with the original *TPP*. We can continue using the gem5 simulator to evaluate the execution time and memory bandwidth. We can also use the QEMU emulator to extract and analyze the vmstats, then compare the performance gain/loss from adding our software algorithm.

3) Overall

At the end, we will compare the performance between *TPP* without our hardware and the whole system (hardware+software) of what we proposed in our plan for improvement.

[1] H. Al Maruf and M. Chowdhury, “Memory disaggregation: Advances and open challenges,” ACM SIGOPS Operating Systems Review, vol. 57, no. 1, pp. 29–37, 2023. doi:10.1145/3606557.3606562.

[2] NUMA Balancing (AutoNUMA). https://mirrors.ed ge.kernel.org/pub/linux/kernel/people/andrea/autonuma/autonuma_bench20120530.pdf.

[3] J. Kim, W. Choe, and J. Ahn. Exploring the design space of page management for Multi-Tiered memory systems. In USENIX ATC, 2021.

[4] H. A. Maruf et al., “TPP: Transparent page placement for CXL-enabled tiered-memory,” Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, 2023. doi:10.1145/3582016.3582063.

[5] H. Li et al., "HoPP: Hardware-Software Co-Designed Page Prefetching for Disaggregated Memory," 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Montreal, QC, Canada, 2023, pp. 1168-1181, doi: 10.1109/HPCA56546.2023.10070986.

[6] “What is NUMA?,” *Vmware.com*, 2023. https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.resmgmt.doc/GUID-1DAB8F35-BA86-4063-8459-55D2979B593E.html

[7] “HOME,” *Compute Express Link*. https://www.computeexpresslink.org (accessed Dec. 12, 2023).