Elastic Use of Far Memory for In-Memory Database Management Systems

Donghun Lee
Minseon Ahn
Jungmin Kim
dong.hun.lee@sap.com
minseon.ahn@sap.com
jimmy.kim@sap.com
SAP Labs Korea
Seoul, South Korea

Daniel Booss
Daniel Ritter
Oliver Rebholz
daniel.booss@sap.com
daniel.ritter@sap.com
oliver.rebholz@sap.com
SAP SE
Walldorf, Germany

Thomas Willhalm
thomas.willhalm@intel.com
Intel Deutschland GmbH
Feldkirchen, Germany

Suprasad Mutalik Desai
Navneet Singh
suprasad.desai@intel.com
navneet.singh@intel.com
Intel Technology India Pvt. Ltd.
Bengaluru, India

ABSTRACT

The separation and independent scalability of compute and memory is one of the crucial aspects for modern in-memory database systems (IMDBMSs) in the cloud. The new, cache-coherent memory interconnect Compute Express Link (CXL) promises elastic memory capacity through memory pooling. In this work, we adapt the well-known IMDBMS, SAP HANA, for memory pools by features of table data placement and operational heap memory allocation on far memory, and study the impact of the limited bandwidth and higher latency of CXL. Our results show negligible performance degradation for TPC-C. For the analytical workloads of TPC-H, a notable impact on query processing is observed due to the limited bandwidth and long latency of our early CXL implementation. However, our emulation shows it would be acceptably smaller with the improved CXL memory devices.

CCS CONCEPTS

• **Hardware → Emerging interfaces**; • **Information systems → Database management system engines**.

KEYWORDS

CXL, Far memory, Memory pool, In-Memory Database, DBMS, Database Management Systems

ACM Reference Format:
Donghun Lee, Minseon Ahn, Jungmin Kim, Daniel Booss, Daniel Ritter, Oliver Rebholz, Thomas Willhalm, Suprasad Mutalik Desai, and Navneet Singh. 2023. Elastic Use of Far Memory for In-Memory Database Management Systems. In *19th International Workshop on Data Management on New Hardware (DaMoN '23), June 18–23, 2023, Seattle, WA, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3592980.3595311

1 INTRODUCTION

The cloud computing industry is constantly evolving, and the needs of the elasticity in computing power and memory capacity are becoming increasingly important. The new memory interconnect Compute Express Link (CXL) [7] promises large, cache-coherent memory capacity and independent, elastic scalability. CXL enables dynamic memory expansion, a disaggregated memory system, and a memory pool, all of which could help to meet the growing demand for the flexible system architecture of cloud-based in-memory database management systems (IMDBMSs) and reduce their total cost ownership (TCO). CXL-attached memory capacity can be more elastically scaled, based on the actual demand, which makes it easier to manage and optimize the resource utilization.

In our previous work [2], we laid out the vision of a CXL memory device for elastic memory expansion of IMDBMSs, and gave initial performance results for standard database benchmarks. We showed that we can easily expand the memory space with nearly no or small amount of performance impact. Additionally in [11], we suggested the classification of the different memory distances, based on their actual physical distance but taking the transport layer into account, and introduced general CXL use cases for the disaggregated memory system (e. g., shared memory pooling, multi-socket). Due to various latencies and different raw materials of CXL memory devices, there would be diverse configurations of tiered memory systems for IMDBMSs. As in [1, 3, 11], we use the term *far memory* for memory devices connected via network – without mediation by a local processor and better availability due to separate fault domains – like CXL 1.1 or CXL-switch based on CXL 2.0, which could consist of DRAM or PMEM to support direct load or store commands within a reasonable duration.

In this work, we introduce and describe use cases for enterprise-scale IMDBMS that leverage a memory pool, namely *use idle compute*, *near-zero downtime upgrade*, *query burst*, *distributed compute*, and *failover*. We extend the well-known SAP HANA IMDBMS by two main features that enable dynamic memory allocation using far memory for those use cases: (i) moving the main storage of any table data to far memory and (ii) allocating the heap memory of SAP HANA execution engine (HEX) [25] to far memory. As far memory usually has a higher latency and a limited bandwidth compared to local memory, understanding the performance impact of the two features (cf. (i), (ii)) on an IMDBMS is crucial. Hence, we evaluate their performance impact on far memory using TPC-C and TPC-H. The main contributions of this work are:

- Specification of CXL use cases for enterprise-scale IMDBMSs,
- Adaptation of a SAP HANA with far memory by features of (i) table data placement and (ii) operational heap memory allocation, and
- Extensive evaluation of the adapted IMDBMS for transactional and analytical workloads.

Our experimental analysis shows that transactional workloads like TPC-C have nearly no performance degradation, despite longer latencies of far memory. Analytical workloads like TPC-H suffer from performance degradation due to the limited bandwidth and long latency of our early CXL implementation, but it would be acceptable with the improved CXL memory devices.

The remainder of this paper is organized as follows: Section 2 introduces CXL and explains the elasticity requirement in cloud-based IMDBMSs. In Section 3, we introduce use cases of far memory for IMDBMSs. The implementation details of far memory are presented in Section 4. In Section 5, we conduct the performance evaluation and discuss important insights. Section 6 discusses related works and Section 7 concludes the paper.

2 BACKGROUND

In this section, we introduce the CXL memory interconnect and describe requirements on the elasticity of cloud IMDBMSs.

2.1 Compute Express Link

Compute Express Link (CXL) is an open standard to support cache-coherent interconnect between a variety of devices [7]. After the introduction of CXL in 2019, the standard has evolved and continues to be enhanced. CXL 1.1 defines the protocol for three major device types: accelerators with cache-only (type 1), cache with attached memory (type 2), and memory expansion (type 3). CXL 2.0 expands the specification – among other capabilities – to memory pools using CXL switches on a device level. CXL 3.0 introduces fabric capabilities and management, improved memory sharing and pooling with dynamic capacity capability, enhanced coherency, and peer-to-peer communication.

Since the market of CXL devices is emerging, several vendors have announced products using CXL. For example, Samsung [18] and SK Hynix [19] introduce CXL DDR5 modules, AsteraLabs [5] announced a CXL memory accelerator, and Montage technology [28] will offer a CXL memory expander controller.

2.2 Elasticity of Cloud IMDBMSs

One of the key requirements of modern IMDBMSs in the cloud is "elasticity". Current public cloud infrastructures are offering virtually unlimited computing and storage resources on demand. With the emergence of disaggregated memory technology such as CXL, Gen-Z [8], OpenCAPI [9] (latter two subsumed by CXL), and CCIX [6], elastic memory capacity in cloud infrastructure will be available in the near future.

However, traditional database architectures are designed for fixed amounts of resources, which would make it difficult for them to leverage such elasticity offered in public cloud infrastructures. To fulfill the requirements of elastic computing capabilities in cloud environments, SAP HANA provides independently, scalable workers, called Elastic Compute Node (ECN). ECNs are similar to SAP HANA scale-out instances[1], except that ECNs have an ephemeral persistence, which does not need to be persisted via NSE [25], and is excluded from services like HANA backup and recovery. Therefore, ECNs do not store normal database tables, while temporary tables or replica tables are populated to ECN instances. With temporary persistence, ECN instances can be easily added and removed depending on the incoming workload state or the customer's demands.

Since CXL version 2.0, the memory pooling capability is enabled. This feature goes beyond simple memory expansion and allows for the dynamic scaling of memory capacity for multiple systems, supporting a more flexible architecture for systems that use it and resulting in the reduced TCO. Memory pooling is particularly useful for IMDBMSs, where large amounts of memory are needed to achieve optimal performance. Thus, our use cases will center around memory pooling for IMDBs and we need to evaluate their impacts on performance for various workloads.

2.3 Data Storage in SAP HANA

SAP HANA is one of the leading Hybrid Transaction / Analytical Processing (HTAP)[2] supporting OLTP and OLAP workloads in a single system, which simplifies the overall system architecture with low TCO [22–24]. It uses a compressed, columnar storage layout for fast-read accesses and a low memory footprint. The columnar data is stored in the read-optimized main storage and maintains a separate delta storage for optimized writes [10, 26]. The delta storage is periodically merged with the main storage [17].

3 FAR MEMORY USE CASES FOR SAP HANA

Memory devices are some of the most expensive parts in the modern computer architectures. Separating memory from compute and memory pooling are promising trends to reduce the memory consumption of an IMDBMS like SAP HANA. That is achieved by allocating only required memory at a particular time and enabling

[1]SAP HANA: https://help.sap.com/docs/SAP_HANA_PLATFORM/6b94445c94ae495c83a19646e7c3fd56/a165e192ba374c2a8b17566f89fe8419.html
[2]Translytical data platforms: https://news.sap.com/2022/12/translytical-data-platforms-forrester-wave-sap-a-leader/

dynamic memory scaling. Subsequently, we present use cases leveraging memory pooling for IMDBs, before we describe how far memory can be practically used in SAP HANA.

3.1 Use Cases

We argue that the following far memory use cases, leveraging CXL memory pooling, are beneficial for IMDBs.

3.1.1 Use idle compute. For workloads running on a SAP HANA process that utilizes only a portion of the available compute capability but does not have enough memory to run another SAP HANA process, hosts with free CPU resources can execute another SAP HANA process using the memory from the memory pool. The memory of the process is bound to the memory exposed by the memory pool, providing simplified life cycle management, since all memory allocated in both local and far by the process gets freed when it terminates. In a variant of this case, only the main storage of a SAP HANA process could be put into far memory, which keeps columnar tables on the pool using the same techniques used for PMem [4]. This case could be most promising for SAP HANA ECNs to leverage idle compute capacity.

3.1.2 Near-zero downtime upgrade. SAP provides near-zero downtime upgrades of enterprise software systems such as SAP S/4HANA. For the upgrade, a "bridging" database subsystem is created as a part of the existing system where a view is generated for each SAP-specific or customer-specific table in the bridge's schema. Business users are transparently transferred and reconnected to the bridge subsystem without any interruption while upgrading the original subsystem. Meanwhile, the bridge subsystem imitates the original system and contains all data of the production system that users need to continue their work. Even though not the entire database is cloned but only selected tables based on the changes to be performed by the maintenance event, we need additional memory capacity for the bridge subsystem. Therefore, a memory pool could be utilized, supporting a tighter sizing on the existing system, regardless of the additional memory requirements during the upgrade.

3.1.3 Query Burst. Occasionally, more memory is needed during phases with more memory-demanding queries. For example, there are increasing memory demands for big queries during the fiscal closing process at the end of every month, quarter, or year. In addition to the memory space for main storage and delta storage, SAP HANA needs temporal, operational memory space for keeping intermediate results and data structures for query processing. SAP HANA's execution engine (HEX) [25] has a clean memory management design and allows for specifying whether memory allocations are done in local or far memory. Using this feature, we allocate the operational memory in a memory pool. It contributes to tight memory sizing and elastic memory capability so that we can allocate more memory dynamically according to the increased memory demands.

3.1.4 Distributed compute. When additional compute nodes are added for SAP HANA's system replication, it may be advantageous to move table data into a memory pool to avoid unnecessary data replication. This allows data to be read from multiple SAP HANA hosts, with synchronized, on-demand updates of the main or delta storage. This usage has the potential to reduce the overall memory costs and enable efficient memory usage in SAP HANA. However, this requires CXL 3.0, since it needs synchronization among multiple hosts. As an interim solution, a dedicated, independent memory space for multiple SAP HANA instances in a memory pool could be utilized, which is supported by CXL 2.0. For that, we gather all the table data for each SAP HANA instance in a memory pool and allocate only the operational memory in the local memory in each host. This approach contributes to better global memory utilization and lower TCO.

3.1.5 Failover. During the planned or unplanned downtime of a SAP HANA instance, a takeover will be performed by another SAP HANA instance. If the main storage of columnar tables is allocated in the memory pool, the SAP HANA processes, which want to take over, can attach to that memory pool and read the tables after a failover without loading the data into memory. This scenario will significantly reduce the required failover time. The ownership change of the memory space in a memory pool among multiple hosts, however, requires CXL 3.0.

3.2 Far Memory Usages in SAP HANA

Similar to [11], we use the term "far memory" as the memory device connected via CXL 1.1 or CXL-switch based on CXL 2.0. To support the use cases of the CXL memory pool mentioned in the previous section, SAP HANA provides some features to enable (i) dynamic data movement and (ii) memory allocation to far memory. Subsequently, we discuss more details of the two features supported by SAP HANA.

3.2.1 Moving main storage. CXL memory pooling can be used by moving the main storage to far memory. We use SAP HANA's persistent memory feature [4] to put the main storage of the columnar tables on far memory with fairly clear semantics on the lifecycle of the main storage. The far memory could be configured as fsdax or tempfs. Any specified table data can be moved to far memory. This feature can be used in the scenarios addressed in Sects. 3.1.1, 3.1.2, 3.1.4 and 3.1.5.

3.2.2 Allocating HEX heap memory. CXL memory pool can be used by allocating HEX heap memory in the far memory. As mentioned in Section 3.1.3, the new SAP HANA execution engine (HEX) [25] can allocate the heap memory required to process a given query to near or far memory according to the configured memory only NUMA node information in SAP HANA. Among the several memory allocators within SAP HANA, the HEX heap memory is the most dominant part of dynamic memory allocation during its query processing. This feature can be used in the scenarios addressed in Sects. 3.1.1 and 3.1.3.

Usually, far memory has a relatively longer latency than local memory – probably about three to four times longer than a single NUMA hop in modern computer architecture. Therefore, it is important to understand the performance characteristics when we apply the far memory usages in SAP HANA. The accesses to the main storage are usually sequential and mostly done by data-intensive operations where L2 prefetching scheme may hide the longer latency of the far memory. Rather, the accesses to the HEX heap memory

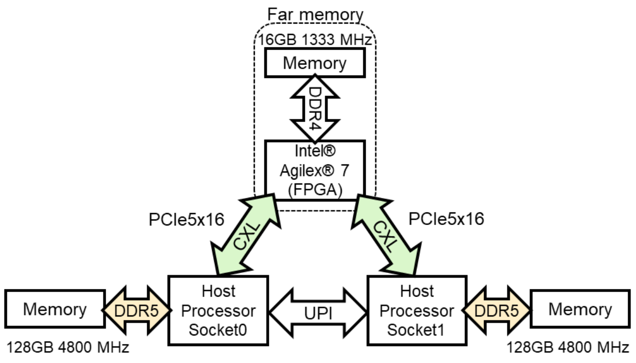


Figure 1: System setup and far memory CXL prototype

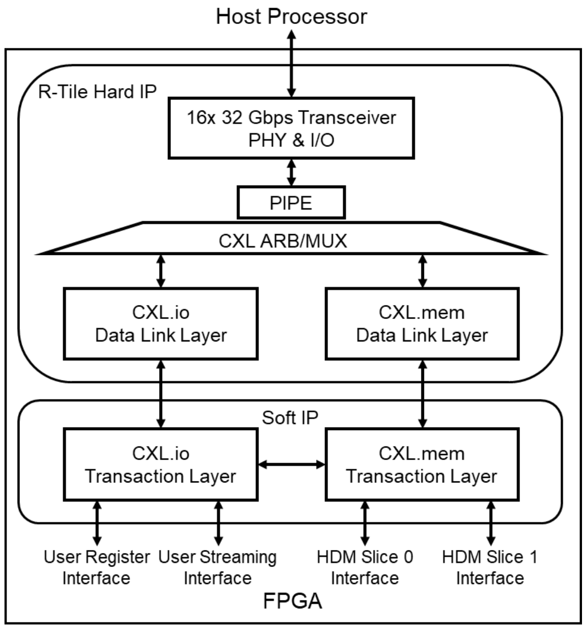


Figure 2: Implementation of the CXL memory pool

are mainly random and frequently done during compute-intensive operations. We will show the performance effect of the two usages against the two different workloads (OLTP and OLAP) in Sect. 5.

4 CXL PROTOTYPE IMPLEMENTATION

In this section, we describe the CXL prototype that we integrated into SAP HANA as far memory, give more details on its internal implementation and discuss limitations.

4.1 CXL Prototype Overview

Figure 1 shows the system setup used to study the performance impact on IMDBMSs. The experimental memory pool is realized using an Intel® Agilex® AGI027 FPGA card which supports PCIe Gen5 x16 CXL connectivity. This FPGA card is connected to the second socket by directly inserting it in a PCIe slot at Socket 1 with 16 PCIe lanes and connected to Socket 0 through two MCIO 8x cables.

4.2 Memory Pool Implementation Details

Figure 2 grants a more detailed view into the implementation of our CXL memory pool on the FPGA card. The R-Tile Intel FPGA IP for CXL implements the CXL link and transaction layer management functions needed to build FPGA-based CXL 1.1/2.0 compliant Type 1, Type 2, and Type 3 endpoint designs. The CXL IP solution consists of a combination of a protocol Soft IP in the FPGA main fabric die paired with the Hard IP companion R-Tile. R-Tile manages all CXL link functions. It connects to a CPU host with a PCIe Gen5x16 interface and supports up to a 64GB/s theoretical bandwidth. The device enumerates as a CXL endpoint.

The Soft IP manages transaction layer functions. For Type 3, the CXL.mem transaction layer receives CXL.mem requests issued from a CPU host, and generates host-managed device memory (HDM) requests to an HDM subsystem. CXL.io transaction layer receives CXL.io requests, either configuration space requests or memory space requests issued from a CPU host, and forwards them to the targeted control and status registers. The User Streaming Interface allows a user design to implement additional custom CXL.io features.

The FPGA card hosts 2x8GB on-board DDR4 1333 MHz memory, which is exposed to the host as memory. It is important to note that in this experimental setup, the FPGA allows accesses to exactly the same amount of memory over each of the two CXL links. In other words, the same far memory can be exposed to two different NUMA nodes of the same size without address overlap between them. The FPGA does not create a single cache-coherent domain, and thus cannot support coherency between these two NUMA nodes assigned to the far memory. Therefore, applications must manage accesses to the shared memory.

4.3 Known Limitations of Prototype

It is worth noting that the bandwidth is limited due to the current implementation of our CXL prototype, and not inherent to CXL. Options to increase the prototype's bandwidth would include:

- Use a faster speed FPGA which supports DDR4 3200 Mbps or DDR5 5600 Mbps.
- Increase the number of slices for the CXL IP.
- Increase the number of independent DDR channels within the device from one to four channels, for example.

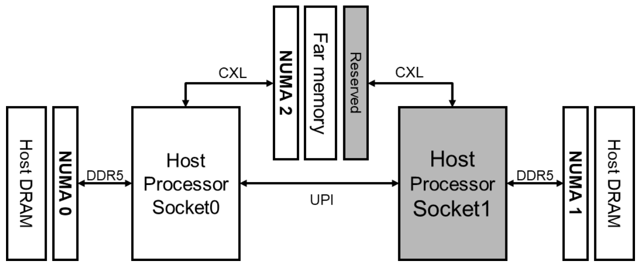
Apart from improving the performance of the device itself, multiple devices can be interleaved to aggregate their bandwidth.

5 PERFORMANCE EVALUATION

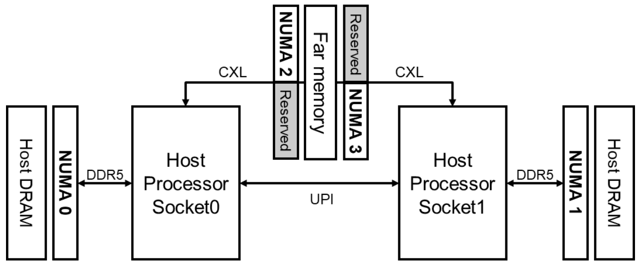
In this section, we evaluate SAP HANA on CXL far memory implementation for common transactional and analytical workloads.

5.1 System Configuration

Our experimental setup is based on Intel 4th generation Xeon processor code-named *Sapphire Rapids*. The system is equipped with two processors with a base frequency of 2.0 GHz and 56 cores each



(a) I-config



(b) Y-config

Figure 3: Far memory setup variants

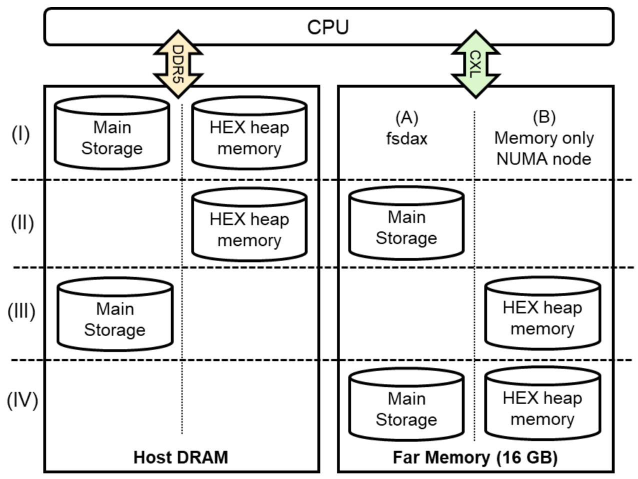


Figure 4: Far memory configurations in I-config

plus Hyper-Threading. Each processor has 128 GB in 8 channels, one 16 GB DDR5 4800 MHz DIMM per channel.

To see the performance effect on various use cases, we use two different far memory setups, I-config and Y-config, as shown in Fig. 3. First, I-config enumerates the entire far memory as NUMA node 2 of size 16 GB directly connected to **Socket 0** as shown in Fig. 3a. We enable the far memory connection only from **Socket 0** to see the performance impact on far memory use cases. In this setup, the CXL connection to **Socket 1** is disabled and the whole far memory in FPGA is accessed through NUMA node 2. Since the far memory is directly connected to the first socket, CPU affinity of SAP HANA is set to **CPU0** on **Socket 0** to avoid any NUMA effect.

To study the performance effects on two different usage types of far memory, i. e., (i) moving the main storage and (ii) allocating HEX heap memory, we divide the far memory into two parts, (A) `fsdax` for the main storage and (B) memory only NUMA node for the HEX heap memory. Then, we evaluate four different configurations by combining these two usage types. The first configuration, **Both DRAM**, has both the main storage and the HEX heap memory in DRAM, which is our baseline in this experiment. The second configuration, **Main FAR**, puts the main storage in the far memory and the HEX heap memory remaining in DRAM while the third configuration, **HEX FAR**, keeps the main storage in DRAM and puts the HEX heap memory in the far memory. The last configuration, **Both FAR**, has both in the far memory. Figure 4 shows the location of the main storage and the HEX heap memory for each configuration in this far memory setup.

The second far memory setup is Y-config where the far memory is connected to both CPUs as shown in Fig. 3b. We enable both far memory connections to confirm the effect of mixed workloads from multiple sockets to a memory pool. We believe it is a general form of the memory pool with CXL 2.0. To avoid any memory access violation, we partition the whole far memory into two halves. The first half is set to NUMA node 2 directly connected to **Socket 0**, and the second half is set to NUMA node 3 directly connected to **Socket 1**.

5.2 Experiment on I-Config

Our experiments are conducted using a transactional / OLTP and an analytical / OLAP benchmark. First, we use TPC-C [29] with 100 warehouses for OLTP workloads, with an `fsdax` size of 12 GB, to support intermediate delta merges, while the memory only NUMA node is set to 4 GB for the HEX heap memory. Second, we use TPC-H [30] for evaluating OLAP workloads. Due to the limited capacity of the far memory, we use TPC-H SF10, which requires 4.7 GB for the main storage. When testing TPC-H, the FPGA memory is divided into 8 GB for `fsdax` and 8 GB for memory only NUMA node because TPC-H requires more HEX heap memory during query processing. Subsequently, we report our experimental results for transactional and analytical query processing on IMDBs on CXL far memory.

5.2.1 OLTP (OnLine Transaction Processing). We use TPC-C for OLTP workload evaluation with 100 warehouses. To populate the transactions in this benchmark test, the number of client threads per process is set to 28. Figure 5 shows the performance and CXL traffic in all four configurations of TPC-C. The performance is normalized to the value at the 16-process configuration of the baseline **Both DRAM**. As shown in Fig. 5, TPC-C has no significant performance degradation even with the smaller available bandwidth and the longer latency of our far memory implementation. Since 8-process configuration, CPU utilization of SAP HANA shows up to 80-90% at maximum. The total number of client connections to SAP HANA is 224 with 8 client processes, which is more than twice the total number of available logical cores in a single socket. Even with the 16-process configuration, where SAP HANA shows nearly full CPU utilization, there is no performance degradation when using far memory.

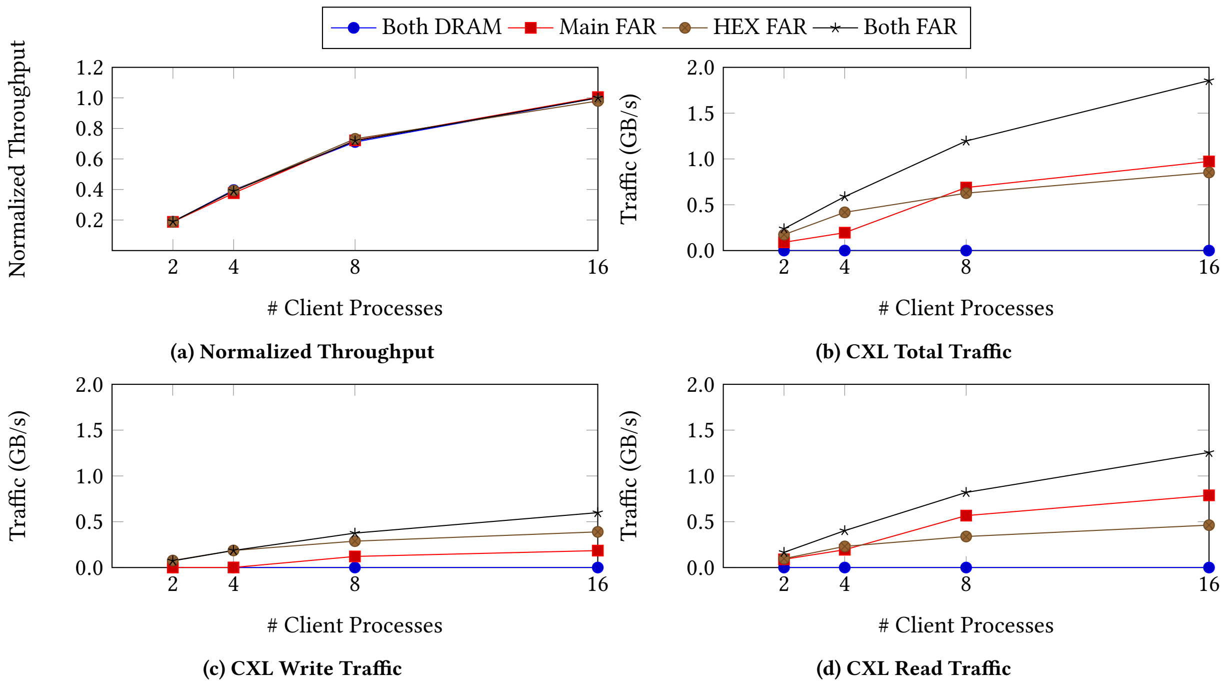


Figure 5: TPC-C throughput and CXL traffic

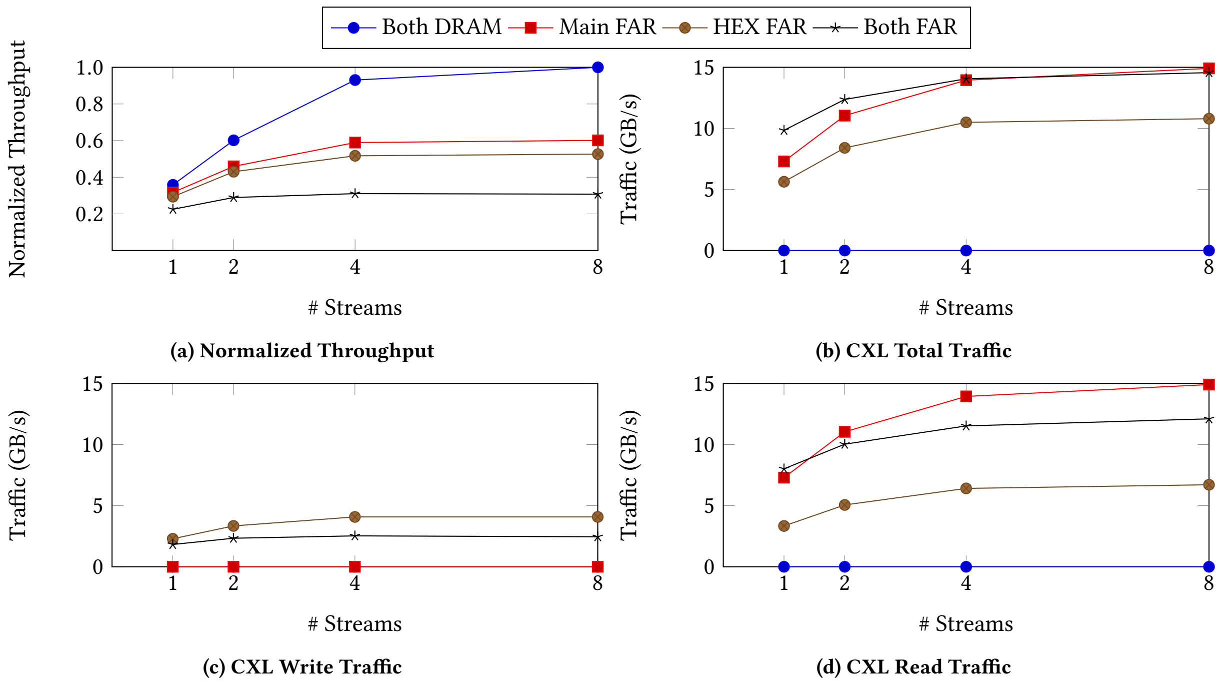
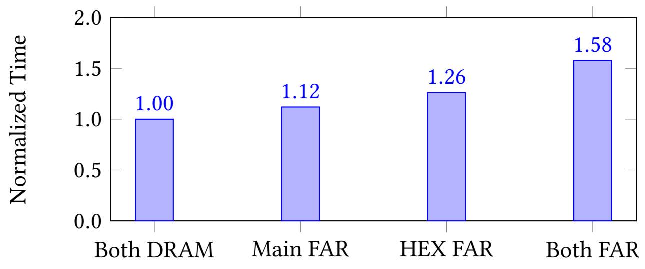
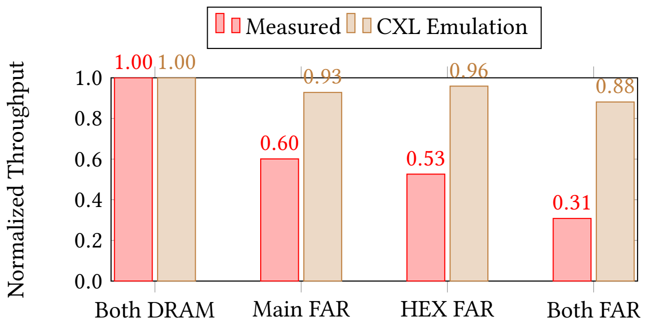


Figure 6: TPC-H throughput and CXL traffic

Our analysis of CXL traffic measured with customized Intel® Performance Counter Monitor [21] shows that there is a meaningful amount of data access to the far memory, but the total amount of CXL traffic is much lower than the maximum bandwidth of our far memory implementation. **HEX FAR** always has a similar amount of CXL write traffic to that of CXL read traffic. It is also observed that **Main FAR** has CXL write traffic only in the configurations with 8 or more processes. This is due to a delta merge after collecting a



(a) Normalized execution time at 1 stream



(b) Normalized throughput at 8 streams

Figure 7: TPC-H performance comparison

certain amount of deltas in the host DRAM. Since CXL traffic is not saturated in TPC-C, the amount of CXL traffic in **Both FAR** appears as the sum of those in both **Main FAR** and **HEX FAR**. Our analysis with Intel® VTune [13] shows that the TPC-C workload causes many lock conflicts and a high synchronization overhead within SAP HANA. We conclude that this synchronization overhead is hiding the longer latency of far memory. Thus, it is not sensitive to the bandwidth or latency of the far memory.

5.2.2 OLAP (OnLine Analytical Processing). We use TPC-H with SF10 for OLAP workload evaluation due to the limited capacity in the CXL memory device. Figure 6 shows the overall performance and CXL traffic in all four configurations of TPC-H. The performance is normalized to the value at the 8-stream configuration of the baseline **Both DRAM**. We observe that **Main FAR** has no CXL write traffic, but it has much more CXL read traffic than **HEX FAR**. Thus, it has a larger amount of CXL total traffic than **HEX FAR**. However, **Main FAR** shows less performance degradation than **HEX FAR**. Our analysis with Intel® VTune reports that **Main FAR** has higher memory bandwidth bound than **HEX FAR**. When moving the main storage, the access pattern is mainly sequential [14] and the prefetching scheme efficiently exploits the bandwidth. Thus, **Main FAR** is affected by limited bandwidth of far memory. Prefetching contributes to the better performance even with the much larger amount of CXL traffic. When allocating HEX heap memory, the access pattern is rather random and, thus is affected by long latency of far memory. Note that **Both FAR** has smaller CXL write traffic than **HEX FAR** because the CXL total traffic is already saturated.

Figure 7 shows the performance comparison for TPC-H. In the execution time with a single stream in Fig. 7a, the performance degradation compared to the baseline **Both DRAM** is measured 12.0% for **Main FAR**, 26.1% for **HEX FAR**, and 57.9% for **Both FAR**. In the throughput with 8 streams, it is measured 39.9% in **Main FAR**, 47.4% in **HEX FAR**, and 69.2% in **Both FAR** as shown in Fig. 7b. The overall performance in TPC-H is bound by the limited bandwidth of our far memory prototype. Since most of the internal bandwidth in our far memory prototype is consumed from the 4-stream configuration, the performance becomes saturated in all configurations using the far memory as shown in Fig. 6b. Thus, TPC-H results show a higher performance degradation because of the bandwidth limitation in our implementation.

To project the performance impact on future CXL devices, we perform CXL emulation using the memory in the remote NUMA node (cf. **Socket1** in Fig. 1), assuming that the access latency to the memory in the remote NUMA node through UPI is similar to the latency of future CXL devices with improved bandwidth. According to the performance test in CXL emulation, performance degradation can be reduced up to 7.2% in **Main FAR**, up to 4.1% in **HEX FAR**, and up to 11.9% in **Both FAR** as shown in Fig. 7b. The results demonstrate that the performance degradation by analytical workload can be significantly reduced when the latency and bandwidth of far memory are improved. Similar results have been observed in our previous work [2].

5.3 Experiment on Y-Config

In this experiment, we use only TPC-H SF10 because OLAP workloads are more sensitive to the bandwidth limitation in far memory configurations. To generate mixed workloads for Y-config, we install two separate SAP HANA instances and assign each instance to a different socket to allow them to run independently and concurrently without any interference except the far memory. The first instance running at **Socket 0** is set to **HEX FAR** by allocating the HEX heap memory to NUMA node 2. The second instance running at **Socket 1** is set to **Main FAR** by moving the main storage to `fsdax` in NUMA node 3.

Figure 8 shows performance throughput of TPC-H in Y-config normalized to the previously measured performance value at the 8-stream configuration of the baseline **Both DRAM**. To see the performance impact on the bandwidth competition in the far memory, we measure the performance for each instance in both cases (1) when they are running **separate**ly consuming the far memory bandwidth exclusively and (2) when they are running **concurrent**ly competing each other for the limited bandwidth of the far memory. This figure shows that **HEX Far** has less performance degradation than **Main FAR** when comparing the performance from the separate execution to the concurrent execution for each instance. In the 8-stream configuration, **HEX Far** has 42.1% performance degradation, while **Main FAR** has 54.8%. When they are running concurrently, it is observed that **HEX Far** always has better performance than **Main FAR**. We believe that the performance degradation comes from the CXL traffic reduction in the concurrent execution because **Main FAR** has a larger amount of CXL traffic

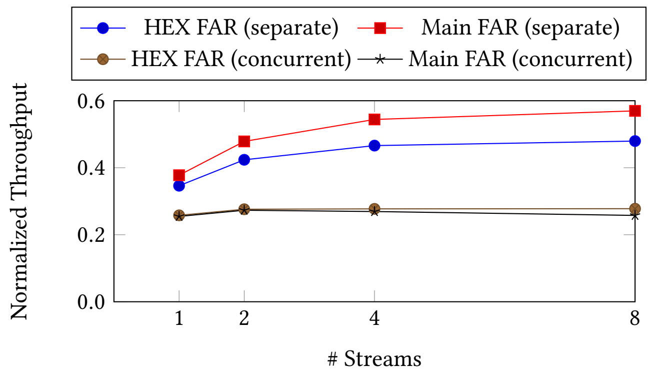


Figure 8: TPC-H throughput on Y-config

than **HEX Far**. Therefore, it is concluded that the prefetch benefit in **Main FAR** is reduced when workloads are competing for the limited available internal bandwidth in the CXL memory pool.

5.4 Discussion

Our experimental results show that the far memory usages have different impacts depending on workloads. TPC-C has negligible performance degradation in any far memory usage because of the small amount of data accesses and high synchronization overhead, which hides longer latency of far memory. Unlike TPC-C, TPC-H has a certain amount of performance degradation because of the limited available bandwidth and long latency when accessing the large amount of data. When the main storage is moved to the far memory, bandwidth dominantly affects the performance degradation because prefetching for the sequential accesses on the main storage saturates the available bandwidth. When the HEX heap memory is allocated in the far memory, latency dominantly affects the performance degradation because of the random accesses on the HEX heap memory. Our analysis with CXL emulation showed that the performance degradation would be acceptably smaller if the future CXL memory pool has better bandwidth and latency.

From the experimental results on Y-config, we observed that prefetch benefit is reduced when the available internal bandwidth in CXL memory pool is limited. Therefore, we conclude that CXL memory pool must have enough internal bandwidth to fully exploit the prefetch benefit of sequential accesses.

6 RELATED WORK

Memory disaggregation has been active in recent years, and several proposals have been made to address the challenges of managing and accessing remote memory. To enable high-performance memory disaggregation in modern data centers, CXL technology has been widely adopted. Gouk et al. [12] have introduced DirectCXL, a system that allows for direct access to remote memory and utilizes intelligent caching and prefetching techniques to achieve high performance. Similarly, Maruf et al. [16] have presented a memory placement strategy that utilizes CXL to create a tiered memory hierarchy that applications can seamlessly access.

CXL for memory expansion and acceleration is one of the hot research topics. Sim et al. [27] have proposed a computational CXL-memory solution that expands memory capacity and improves performance for memory-intensive applications. Park et al. [20] have developed a CXL memory expander that scales memory performance and capacity in modern data centers.

Furthermore, the performance and efficiency of CXL-enabled memory pooling systems have been investigated. Li et al. [15] have introduced a CXL-based memory pooling system that can be employed in cloud platforms to enhance memory utilization and performance. Additionally, Yang et al. [31] have presented a CXL-enabled hybrid memory pool that combines DRAM and PMEM resources to improve memory performance.

7 CONCLUSION

The separation of compute and memory, and their composition through far memory based on CXL enables new use cases for IMDBMSs, shows a path from current resource over-provisioning, and reduces their TCO. In this paper, we introduced and discussed promising far memory use cases for IMDBMSs and two fundamental adaptations (i. e., table data placement and operational heap memory allocation on far memory) to SAP HANA, and conducted a performance evaluation of these two adaptations using the far memory implemented on an early FPGA-based CXL prototype.

The results of our experimental evaluation show that transactional workloads like TPC-C have nearly no performance degradation in any far memory usage because of the small amount of data accesses and high synchronization overhead. However, analytical workloads, such as TPC-H, causing large amount of data accesses, have a certain amount of performance degradation because of the limited available bandwidth and long latency of far memory. As shown in our CXL emulation results using remote NUMA memory, we expect an acceptable performance degradation in TPC-H with improved bandwidth and latency of far memory in the future. Therefore, we conclude that the far memory can allow elasticity in IMDBMSs with the improved CXL memory devices.

In future work, we will further develop solutions on how to avoid performance impacts, especially for data intensive analytic workloads due to the limited bandwidth and long latency of far memory. Moreover, we will collect and specify more far memory use cases for memory pools to fully utilize the elastic memory capacity for cloud IMDBMSs.

REFERENCES

[1] Marcos K. Aguilera, Kimberly Keeton, Stanko Novakovic, and Sharad Singhal. 2019. Designing Far Memory Data Structures: Think Outside the Box. In *HotOS*. ACM, 120–126. https://doi.org/10.1145/3317550.3321433
[2] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna T. Malladi, and Yang-Seok Ki. 2022. Enabling CXL Memory Expansion for In-Memory Database Management Systems. In *DaMoN*. ACM, 8:1–8:5. https://doi.org/10.1145/3533737.3535090
[3] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can far memory improve job throughput?. In *EuroSys*. ACM, 14:1–14:16. https://doi.org/10.1145/3342195.3387522
[4] Mihnea Andrei, Christian Lemke, Günter Radestock, et al. 2017. SAP HANA adoption of non-volatile memory. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1754–1765. https://doi.org/10.14778/3137765.3137780
[5] AsteraLabs. 2022. *CXL Memory Accelerators*. https://www.asteralabs.com/products/cxl-memory-platform/
[6] CCIX Consortium. 2017. *CCIX*. https://www.ccixconsortium.com/

[7] Compute Express Link Consortium. 2019. *CXL*. https://www.computeexpresslink.org/

[8] Gen-Z Consortium. 2016. *Gen-Z*. https://genzconsortium.org/

[9] OpenCAPI Consortium. 2014. *OpenCAPI*. https://opencapi.org/

[10] Franz Faerber, Alfons Kemper, Per-Åke Larson, Justin J. Levandoski, Thomas Neumann, and Andrew Pavlo. 2017. Main Memory Database Systems. *Found. Trends Databases* 8, 1-2 (2017), 1–130. https://doi.org/10.1561/1900000058

[11] Andreas Geyer, Daniel Ritter, Dong Hun Lee, Minseon Ahn, Johannes Pietrzyk, Alexander Krause, Dirk Habich, and Wolfgang Lehner. 2023. Working with Disaggregated Systems. What are the Challenges and Opportunities of RDMA and CXL?. In *BTW (LNI, Vol. P-331)*. Gesellschaft für Informatik e.V., 751–755. https://doi.org/10.18420/BTW2023-47

[12] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct access, High-Performance memory disaggregation with DirectCXL. In *USENIX ATC*. 287–294. https://www.usenix.org/conference/atc22/presentation/gouk

[13] Intel®. 2021. *Intel® VTune™ Profiler*. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html

[14] Robert Lasch, Thomas Legler, Norman May, Bernhard Scheirle, and Kai-Uwe Sattler. 2022. Cost modelling for optimal data placement in heterogeneous main memory. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2867–2880. https://www.vldb.org/pvldb/vol15/p2867-lasch.pdf

[15] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. 2023. Pond: Cxl-based memory pooling systems for cloud platforms. In *ASPLOS*. ACM, 574–587. https://doi.org/10.1145/3575693.3578835

[16] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit O. Kanaujia, and Prakash Chauhan. 2022. TPP: Transparent Page Placement for CXL-Enabled Tiered Memory. *CoRR* abs/2206.02878 (2022). https://doi.org/10.48550/arXiv.2206.02878 arXiv:2206.02878

[17] J. McGlone, P. Palazzari, and J. B. Leclere. 2018. Accelerating Key In-memory Database Functionality with FPGA Technology. In *ReConFig*. 1–8. https://doi.org/10.1109/RECONFIG.2018.8641722

[18] Samsung Newsroom. 2022. *Samsung Electronics Introduces Industry's First 512GB CXL Memory Module*. https://news.samsung.com/global/samsung-electronics-introduces-industrys-first-512gb-cxl-memory-module

[19] SK Hynix Newsroom. 2022. *SK hynix Develops DDR5 DRAM CXLTM Memory to Expand the CXL Memory Ecosystem*. https://news.skhynix.com/sk-hynix-develops-ddr5-dram-cxltm-memory-to-expand-the-cxl-memory-ecosystem/

[20] S. J. Park, H. Kim, K.-S. Kim, J. So, J. Ahn, W.-J. Lee, D. Kim, Y.-J. Kim, J. Seok, J.-G. Lee, H.-Y. Ryu, C. Y. Lee, J. Prout, K.-C. Ryoo, S.-J. Han, M.-K. Kook, J. S. Choi, J. Gim, Y. S. Ki, S. Ryu, C. Park, D.-G. Lee, J. Cho, H. Song, and J. Y. Lee. 2022. Scaling of Memory Performance and Capacity with CXL Memory Expander. In *2022 IEEE HCS*. IEEE, 1–27. https://doi.org/10.1109/HCS55958.2022.9895633

[21] Intel® PCM. 2023. *Intel® Performance Counter Monitor*. https://github.com/intel/pcm

[22] Hasso Plattner. 2009. A common database approach for OLTP and OLAP using an in-memory column database. In *SIGMOD*. ACM, 1–2. https://doi.org/10.1145/1559845.1559846

[23] Hasso Plattner. 2014. The Impact of Columnar In-memory Databases on Enterprise Systems: Implications of Eliminating Transaction-maintained Aggregates. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1722–1729. https://doi.org/10.14778/2733004.2733074

[24] Iraklis Psaroudakis, Florian Wolf, Norman May, Thomas Neumann, Alexander Boehm, Anastasia Ailamaki, and Kai-Uwe Sattler. 2015. Scaling Up Mixed Workloads: A Battle of Data Freshness, Flexibility, and Scheduling. *Performance Characterization And Benchmarking: Traditional To Big Data* 8904 (2015), 16. 97–112. https://doi.org/10.1007/978-3-319-15350-6_7

[25] Reza Sherkat, Colin Florendo, Mihnea Andrei, Rolando Blanco, Adrian Dragusanu, Amit Pathak, Pushkar Khadilkar, Neeraj Kulkarni, Christian Lemke, Sebastian Seifert, Sarika Iyer, Sasikanth Gottapu, Robert Schulze, Chaitanya Gottipati, Nirvik Basak, Yanhong Wang, Vivek Kandiyanallur, Santosh Pendap, Dheren Gala, Rajesh Almeida, and Prasanta Ghosh. 2019. Native Store Extension for SAP HANA. *Proc. VLDB Endow.* 12, 12 (2019), 2047–2058. https://doi.org/10.14778/3352063.3352123

[26] Vishal Sikka, Franz Färber, Wolfgang Lehner, Sang Kyun Cha, Thomas Peh, and Christof Bornhövd. 2012. Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. In *SIGMOD*. ACM, 731–742. https://doi.org/10.1145/2213836.2213946

[27] Joonseop Sim, Soohong Ahn, Taeyoung Ahn, Seungyong Lee, Myunghyun Rhee, Jooyoung Kim, Kwangsik Shin, Donguk Moon, Euiseok Kim, and Kyoung Park. 2023. Computational CXL-Memory Solution for Accelerating Memory-Intensive Applications. *IEEE Comput. Archit. Lett.* 22, 1 (2023), 5–8. https://doi.org/10.1109/LCA.2022.3226482

[28] Montage Technology. 2022. *Montage Technology Delivers the World's First CXL™ Memory eXpander Controller*. https://www.montage-tech.com/Press_Releases/20220506

[29] TPC-C. 2023. *TPC-C*. https://www.tpc.org/tpcc/

[30] TPC-H. 2023. *TPC-H*. https://www.tpc.org/tpch/

[31] Qirui Yang, Runyu Jin, Bridget Davis, Devasena Inupakutika, and Ming Zhao. 2022. Performance Evaluation on CXL-enabled Hybrid Memory Pool. In *NAS*. IEEE, 1–5. https://doi.org/10.1109/NAS55553.2022.9925356