# Exploring Dynamic Memory Allocation of CXL Memory Pools in Enterprise In-Memory Database Management Systems

Donghun Lee
dong.hun.lee@sap.com
SAP Labs Korea
Seoul, South Korea

Minseon Ahn
minseon.ahn@sap.com
SAP Labs Korea
Seoul, South Korea

Jungmin Kim
jimmy.kim@sap.com
SAP Labs Korea
Seoul, South Korea

Jaemin Jung
j.jaemin@samsung.com
Samsung Semiconductor Inc.
San Jose, California, USA

Norman May
norman.may@sap.com
SAP SE
Walldorf (Baden), Germany

Daniel Ritter
daniel.ritter@sap.com
SAP SE
Walldorf (Baden), Germany

Jongmin Gim
gim.jongmin@samsung.com
Samsung Semiconductor Inc.
San Jose, California, USA

Heekwon Park
heekwon.p@samsung.com
Samsung Semiconductor Inc.
San Jose, California, USA

Changho Choi
changho.c@samsung.com
Samsung Semiconductor Inc.
San Jose, California, USA

Yang Seok Ki
yangseok.ki@samsung.com
Samsung Semiconductor Inc.
San Jose, California, USA

## Abstract

Managing large data volumes presents a significant challenge for enterprise systems, particularly within in-memory database management systems (IMDBMSs). Traditional server architectures with limited memory slots lead to increased total cost of ownership (TCO) when scaling memory capacity. This paper investigates Compute Express Link (CXL) as a solution, offering a disaggregated memory architecture that facilitate memory pooling. By enhancing scalability and operational flexibility, CXL significantly reduces TCO through efficient capacity sharing and minimizes over-provisioning. Our study evaluates CXL memory pools in enterprise IMDBMS contexts, specifically using SAP HANA. We explore CXL's dynamic memory allocation capabilities and analyze its impact on IMDBMS performance, while considering potential challenges like increased latency and limited bandwidth inherent in CXL switches. Through practical insights and performance evaluations of a memory pool, this work highlights the promising use cases such as compute and data node disaggregation and dynamic assignment of intermediate query results. Our findings indicate improved flexibility of server architecture and reduced TCO in cloud environments.

## Keywords

CXL, Memory Disaggregation, Pooled Memory, In-Memory Database, DBMS, Database Management Systems

## 1 Introduction

Flexible memory configuration has emerged as a significant challenge as enterprise systems increasingly manage large volumes of data, particularly in in-memory database management systems (IMDBMSs). The constraint imposed by the limited number of memory slots in a single server necessitates the adoption of high capacity DIMMS for a high memory-to-core ratio in compute servers. Furthermore, expanding memory capacity leads to a significant rise in the total cost of ownership (TCO) [9] due to increased costs for scale-out. This growing imbalance between data growth and memory scalability underscores the need for new architectural solutions that can extend memory capacity beyond a single server boundary without compromising performance or manageability.

The recent emergence of Compute Express Link (CXL) offers a promising solution to overcome these limitations by enabling flexible, disaggregated memory architectures. Moreover, CXL supports memory pooling, where a portion of CXL-attached memory can be dynamically allocated to multiple hosts. This capability facilitates fine-grained memory elasticity and improves overall resource utilization especially in enterprise cloud environments. By decoupling memory capacity from compute resources, CXL not only enhances scalability and operational flexibility but also significantly reduces total cost of ownership (TCO) by minimizing over-provisioning and enabling efficient capacity sharing across multiple hosts.

However, the high latencies and limited bandwidth of CXL memory devices may raise concerns about potential negative performance impacts. Our previous work [2, 3] explored the performance implications of directly attached CXL memory and Intel Flat Memory Mode [31] for in-memory database management systems (IMDBMSs). Our studies revealed that performance impacts are variable based on workload characteristics and specific memory access patterns. On-Line Transaction Processing (OLTP) workloads, such as TPC-C [26], do not experience performance degradation due to small CXL traffic and high synchronization overhead. In contrast, On-Line Analytical Processing (OLAP) workloads, such as TPC-DS [27], exhibit a wide range of performance degradation depending on the memory access patterns. CXL pooled memory has the potential to enable unprecedented flexibility in system architecture for enterprise software systems. However, there is concern about the performance impact due to

*EDBT '26, Tampere (Finland)*

increased latency and limited memory bandwidth caused by the CXL switch, which could potentially add around one hundred nanoseconds to the latency.

This work introduces the application of a real-world CXL memory pool and explores its use cases for enterprise IMDBMSs. We present our performance evaluation results of the CXL memory pool in SAP HANA, tested with both OLTP and OLAP workloads. Our study specifically investigates a CXL pooled memory system for dynamic memory allocation, which provides dedicated memory space to each server. The approach differs from directly attached memory expansion in a single server by enabling memory pooling rather than static memory extension, thereby allowing for flexible memory configurations in cloud environments. We consider memory sharing of memory areas in the pool among multiple servers as future work.

The main contributions of this work include:

- sharing practical insights and experiences related to using a real-world memory pool based on a CXL switch,
- evaluation of real-world memory pool on an enterprise IMDBMS (SAP HANA) for dynamic memory allocation usage,
- investigation of performance characteristics according to workload and memory access patterns to a memory pool on IMDBMSs,
- use case development of CXL memory pool on IMDBMSs to enable flexible architecture to maintain a low server TCO: (1) System-wide: disaggregation of compute and data nodes, (2) Query-wise: dynamic assignment of intermediate results during large query executions.

The remainder of this paper is organized as follows: Section 2 introduces CXL memory pool and SAP HANA. Section 3 describes our implementation. Section 4 discusses use cases of CXL memory pool for IMDBMSs. We conduct the performance evaluation and share its analysis in Section 5 and discuss our insight in Section 6. We give an overview of related work in Section 7. Finally, we conclude our work in Section 8.

## 2 Background

In this section, we introduce CXL memory pool and SAP HANA in-memory database management system.

### 2.1 CXL Memory Pool

Compute Express Link (CXL) is an open standard interconnect that has rapidly evolved from a point-to-point CPU–device link into a full-fledged fabric for disaggregated memory. The initial CXL 1.x (2019) specification [7, 17] established a low-latency, cache-coherent interface by defining three sub-protocols—CXL.io, CXL.cache, and CXL.mem –– primarily targeting accelerators and memory expanders in a host-centric model. CXL 2.0 (2020) extended these capabilities by introducing switching, memory pooling, persistent memory support, and multi-host access, enabling dynamic sharing of memory resources across servers. Building on this foundation, CXL 3.0 (2022) and 3.1 (2023) [5] transformed the standard into a scalable fabric interconnect with support for PCIe 6.0 PHY, multi-level switching, fabric IDs, and coherent multi-host memory sharing across thousands of devices, thereby enabling large-scale disaggregated and composable infrastructures. Recently, CXL 3.2 (2024) [6] was proposed to enhance the ecosystem with advanced monitoring and management features.

A CXL switch–based memory pool requires a fully compliant CXL 2.0 fabric that enables hosts to access disaggregated memory devices through a shared switch interconnect. To construct such a pool, the switch must support multi-host routing of CXL.mem transactions and maintain host–device address mappings via Host-Managed Device Memory (HDM) decoders. A dedicated fabric manager is essential to configure and control these mappings, enabling dynamic memory assignment across hosts. On the host side, BIOS and firmware must correctly enumerate the pooled CXL topology, while the operating system integrates with the CXL kernel subsystem to online or offline memory regions at runtime. Furthermore, adequate switch bandwidth, low forwarding latency, and quality-of-service isolation are critical to ensure predictable performance and to prevent interference among concurrently accessing hosts. These requirements collectively enable a scalable and flexible CXL memory pool capable of dynamic capacity provisioning across hosts.

CXL switch–based memory pooling [10] represents a fabric-centric approach, in which multiple hosts access disaggregated memory devices through a shared CXL switch. This topology enables high fan-out scalability and flexible resource allocation, at the cost of an additional hop and increased management complexity. In contrast, multi-headed CXL memory devices embody a device-centric pooling model [4] that allows direct attachment to multiple hosts through independent CXL links. While this design reduces switching latency and simplifies the topology, its scalability is inherently constrained by the limited number of device ports and the need for internal coherence management. Consequently, switch-based pooling is preferable for large-scale, dynamically reconfigurable infrastructures, whereas multi-headed memory devices suit tightly coupled, low-latency host clusters. In this paper, we adopt the switch-based pooling architecture as our foundation, leveraging its flexibility and scalability to construct a unified memory pool that can dynamically allocate and rebalance CXL memory capacity among multiple hosts.

### 2.2 SAP HANA In-Memory Database Management System

SAP HANA is one of the leading Hybrid Transaction / Analytical Processing (HTAP) platforms supporting OLTP and OLAP workloads in a single system [11, 15], which simplifies the overall system architecture with low TCO [18–20]. Relational tables are organized in a columnar storage layout and enable fast read accesses. The memory footprint is reduced by applying various compression schemes. The columnar data is stored in the read-optimized main storage and maintains a separate delta storage for optimized writes [8, 24]. The delta storage is periodically merged with the main storage [16] to keep the delta storage small and to benefit from the better compression in the main storage. Operational memory encompasses all temporary data used by the HANA Execution Engine (HEX) [23], excluding the main storage and the delta storage. For the experiments conducted in this study, a few new features have been integrated into our HANA prototype to support CXL memory space and allocate operational memory to CXL memory. Now, HEX enables far memory allocation, allowing portions of memory to be allocated to specific Non-Uniform Memory Access (NUMA) nodes. Particularly, we introduce a new feature that allocates temporary tables to specific NUMA nodes to store intermediate results during SQL processing within SQL scripts. However, there is no assurance that these features will be available in the commercial version of SAP HANA in the near future.

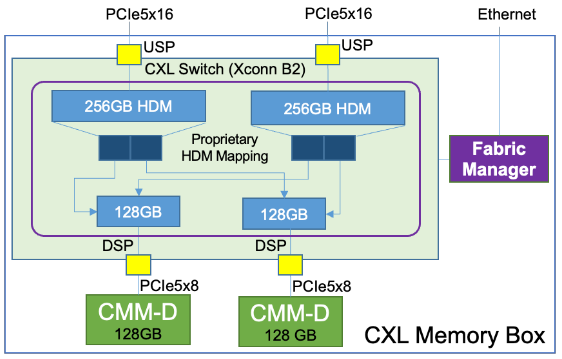


**Figure 1: System Configuration of CXL Memory Pool**

## 3 Implementation of CXL Memory Pool

In this section, we detail the CXL memory pool setup and dynamic memory allocation within the pool of multiple hosts.

### 3.1 System Architecture

Figure 1 illustrates CXL memory pool consisting of a CXL switch and CXL memory devices. We use the XConn B2 CXL switch chip that supports up to 256 PCIe Gen5 lanes for CXL upstream and downstream links. We configure our memory pooling system with 2 hosts, each connected to PCIe Gen5x16 upstream ports (USPs). On downstream ports (DSPs), we install two 128GB Samsung CXL memory devices (CMM-D), each consisting of a 128GB DDR5 DIMM supporting 5200 MT/s, an internal memory controller, an ASIC logic converting CXL protocols to DDR5, and a PCIe Gen5x8 interface. The CXL switch provides a Host-Managed Device Memory (HDM) for each upstream port. From the host's perspective, the HDM appears as a dedicated CXL memory space. In our setup, each host is allocated a 256 GB HDM. The CXL switch maintains the mapping between the upstream HDM region and downstream devices. This HDM mapping is configured by the fabric manager and can be dynamically updated during operation. In our test vehicle, we use XConn's proprietary HDM mapping.

### 3.2 Dynamic Allocation on a CXL Memory Pool

In our hardware setup, dynamic memory pooling can be enabled either at the switch layer or the operating system (OS) layer. At the switch layer, the HDM mapping can be dynamically modified via a fabric manager. However, this approach is not applied in our evaluation due to system stability concerns with our current hardware. Instead, we enable dynamic memory pooling through Linux kernel–level memory onlining and offlining, while maintaining a static configuration in the fabric manager. Specifically, we configure the HDM mapping to bind all 128GB CMM-D devices sequentially, and form a single 256GB HDM region for each host to enable dynamic pooling over the entire downstream devices. In this configuration, both CMM-D devices are technically accessible by each host. After the CXL memory pool is enumerated by the BIOS, the entire 256GB region becomes visible to the system. Once the operating system boots, it can dynamically online or offline memory blocks to expand or shrink the available memory capacity for each host. Applications can transparently access the pooled memory region via standard load/store semantics over the CXL.mem protocol, thereby leveraging disaggregated memory without modification in applications. Since the operating system provides an identical memory abstraction to applications, regardless of the underlying CXL memory configuration, we can adopt the same use cases as in our previous work [3, 13], such as moving table data and allocating operational heap memory. Unlike directly attached CXL memory in our previous work [3, 13], CXL memory pools enables more flexible and scalable memory system architecture, with the reduced system TCO. However, the introduction of a CXL switch adds an extra hop in the memory access path, which increases access latency and slightly reduces the effective bandwidth compared to directly attached CXL memory. In this experiment, we set the disjoint pooled memory region for each host, ensuring no overlap, to avoid memory access violations by multiple hosts.

## 4 Use Cases of CXL Memory Pool for IMDBMSs

This section describes the valuable use cases of CXL memory pool in in-memory database management systems.

Disaggregated memory shown in Fig. 2 illustrates the fundamental trade-off between the scalability and the latency in disaggregated memory systems. Local DRAM offers the highest bandwidth and lowest latency (`t0`) but is limited in capacity and scalability. As memory is disaggregated–first across sockets via UPI (`t1`) and then beyond the processor package through CXL interconnects–the latency of access increases due to protocol serialization (`t2`), switch traversal, and device buffering (`t3`), while effective bandwidth declines with link serialization and shared fabric contention. Nevertheless, CXL-attached and pooled memory tiers provide unprecedented scalability and flexibility, enabling elastic memory provisioning across hosts. Therefore, system designers must balance performance locality against capacity elasticity when integrating CXL memory into database systems such as SAP HANA and developing their meaningful use cases.

### 4.1 Moving Table Data and Allocating Operational Heap Memory

One key challenge in utilizing CXL memory lies in determining which data should be transitioned to it. The access characteristics of internal memory in SAP HANA can be broadly divided into two categories: table data in main storage and operational memory used by runtime components such as the HEX heap memory. Table data, which stores columnar tables in compressed form, exhibits high spatial locality and predominantly sequential access patterns, since scan-intensive analytical queries typically traverse contiguous column segments and perform vectorized processing over large datasets. In contrast, operational memory is used for query execution structures, intermediate buffers, and metadata management, and thus shows low spatial locality and high temporal volatility. The HEX heap memory exemplifies this behavior, with random access patterns, frequent writes, and short-lived allocations. These different access behaviors result in distinct sensitivity to latency and bandwidth: sequential, read-dominant table scans benefit from high-bandwidth memory tiers, whereas random, write-intensive operational workloads are more latency-sensitive and better served by local DRAM. Thus, negative performance impacts caused by the high latency and limited bandwidth of CXL memory pool depend on the workloads and the memory access patterns of internal memory structures. In Section 5.1, we share our investigation results on this use case.

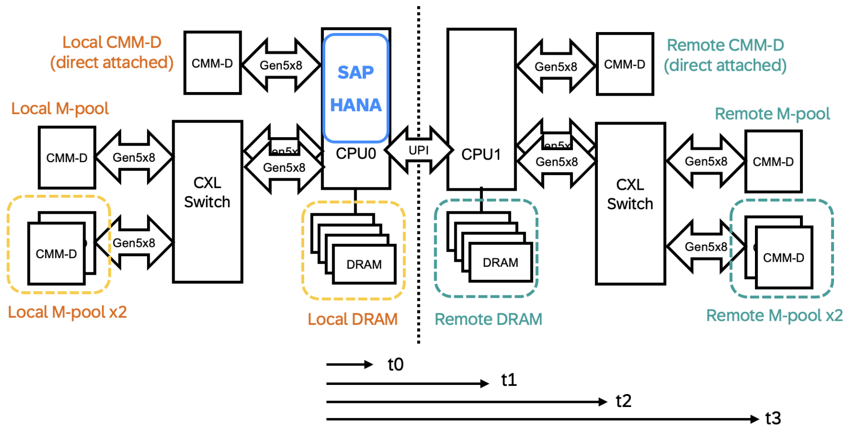


**Figure 2: CXL memory configuration with different latency**

## 4.2 Fast Restart using HANA NVM Feature

Introducing CXL (Compute Express Link) pooled memory with an independent power source offers significant advantages for enterprise systems, particularly in the context of IMDBMSs, such as SAP HANA. One key benefit is the ability to maintain data integrity in the memory pool even when systems or database instances shut down or restart. This property ensures that data is not lost and remains available immediately upon restarting the server. Leveraging the existing HANA non-volatile memory (NVM) feature, the table data stored in CXL memory pools can significantly enhance performance during restarts. Instead of reloading data from data volumes, HANA instances can access the data directly from the CXL memory pool, facilitating a faster restart process. This capability not only improves system resilience but also reduces the time for systems to become operational after maintenance or unexpected shutdowns, providing a substantial boost to overall efficiency and availability.

## 4.3 Dynamic Assignment of Intermediate Results for Big Queries

Handling large-scale analytical workloads that generate massive intermediate results exceeding the capacity of local DRAM is challenging because such workloads place extreme and unpredictable pressure on memory resources. Analytical queries with multi-way joins, group-by aggregations, or sorting operations, may frequently produce intermediate datasets that exceed the local DRAM capacity. In these situations, the system must either spill operational data to disk resulting in a performance degradation due to the large gap between DRAM and storage latency, or terminate the query with an out-of-memory (OOM) error. In this approach, temporary tables created during complex joins, aggregations, or sorts, are dynamically allocated to a shared CXL memory pool. By leveraging disaggregated CXL memory, the system can elastically expand memory resources on demand without statically overprovisioning large virtual machines from the outset. This not only prevents frequent OOM errors during query execution but also significantly reduces management overhead and server TCO.

In this use case, temporary tables generated during SQL Script execution in SAP HANA are dynamically stored in a pooled memory region to handle large intermediate results efficiently. During complex analytical workloads, SQL Scripts often produce multiple intermediate results that must be passed between sequential SQL statements. The total size of these temp tables increases linearly with the number of concurrent users or processes, frequently exceeding the capacity of local DRAM and leading to potential OOM failures. The CXL memory pool serves as a scalable, cost-efficient extension of system memory, enabling high-performance query processing even under memory-intensive workloads without manual intervention or performance collapse due to OOM conditions.

## 5 Performance Evaluation

In this section, we evaluate the performance of the CXL memory pool use cases.

## 5.1 Moving Table Data and Allocating Operational Heap Memory

*5.1.1 System Configuration.* For evaluating dynamic memory allocation of table data and operational heap memory, we set up a system based on Intel's 5th generation Xeon® processor, code-named Emerald Rapids. The system is equipped with a single processor featuring 48 physical cores, totally 96 cores with Hyper-Threading enabled. Each of the eight memory channels is populated with a 128 GB DDR5 4800 MT/s DIMM, providing a total of 1,024 GB of local DRAM. The CXL memory pool is provided by a commercial-grade CXL memory box integrating one Xconn CXL switch and two CXL memory devices. We configure the CXL memory capacity by dynamically bringing memory regions online or offline in Linux kernel. To investigate the performance impact of increased latency, both the directly attached CXL memory devices (CMM-D) and the CXL memory pool are connected to the remote socket rather than the local socket in the server as seen in Fig. 2.

When moving table data to the CXL memory pool, we set up an `fsdax` file system to manage the main storage in the allocated CXL space. For allocating HEX heap memory in the CXL pool, we use the NUMA library to bind allocations to the NUMA node corresponding to that pool. We evaluate OLTP workloads using TPC-C [26] with 100 warehouses and OLAP workloads using TPC-DS [27] with a scale factor of 100. We adopt the same use cases as in our previous work [3, 13] as mentioned in Section 3.2. Using the same use cases allows us to isolate the impact of the memory substrate itself. In this paper, we therefore focus on evaluating the effectiveness of a CXL memory pool—which exhibits higher latency than directly attached CXL memory—under identical workloads, enabling a fair and controlled comparison.

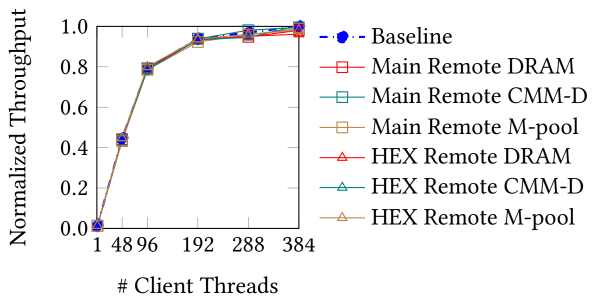


**Figure 3: Performance Comparison of TPC-C**

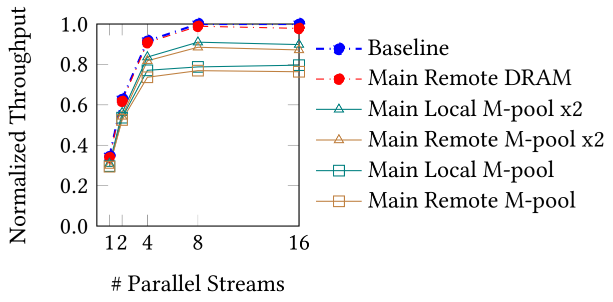


**Figure 4: Performance Comparison of TPC-DS**

*5.1.2 Experimental results of TPC-C.* Figure 3 presents the overall performance when executing the TPC-C benchmark with 100 warehouses. We measure the total number of transactions per unit time while increasing the number of client threads within a single client process. The throughput is normalized to the baseline performance at the 384-thread configuration. To study the performance effects on two different CXL memory devices, we measure the performance in the following configuration; (1) **Baseline** where both the main storage and the HEX heap memory are in local DRAM, (2) **Main Remote DRAM** where the main storage is moved to DRAM in a remote socket, (3) **Main Remote CMM-D** where the main storage is moved to directly attached CXL memory connected to a remote socket, (4) **Main Remote M-pool** where the main storage is moved to a pooled memory region connected to a remote socket, (5) **HEX Remote DRAM** where HEX heap memory is allocated in DRAM in a remote socket, (6) **HEX Remote CMM-D** where HEX heap memory is allocated in directly attached CXL memory connected to a remote socket, (7) **HEX Remote M-pool** where HEX heap memory is allocated in a pooled memory region connected to a remote socket. The results show that OLTP workloads experience no observable performance degradation, despite the higher latency and limited bandwidth of the CXL memory pool. This observation is consistent with our previous studies [1, 3, 13]. During the test, peak CXL traffic reached approximately 13 GB/s with 288 client threads on the HEX Remote M-pool configuration—-well below the device's offered bandwidth—-so the fabric was not saturated and bandwidth was not the limiting factor. These bandwidth measurements align with our previous study [3], where we confirmed that OLTP workloads are largely insensitive to additional memory latency due to high synchronization overhead between transactions.

*5.1.3 Experimental results of TPC-DS.* Figure 4 illustrates the performance when executing the TPC-DS benchmark with scale

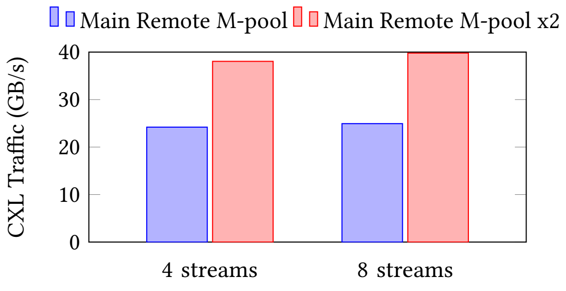


**Figure 5: CXL Traffic of TPC-DS**

factor 100. We measure the total number of queries per unit time while increasing the number of parallel streams when moving table data to the CXL memory pool. In this study, we do not consider the allocation of HEX heap memory within the CXL memory pool as a viable use case. Our previous work [3, 13] shows that this approach generally results in significant performance degradation (more than 50%) due to the dominant random access patterns, even when using directly attached CXL devices, which have lower latency compared to CXL memory pools.

In this measurement, we use 6 different configurations. **Baseline** stores both the main storage and HEX heap memory in local DRAM. **Main Remote DRAM** stores the main storage in remote DRAM shown in Fig. 2, while HEX heap memory is allocated in local DRAM. **Main Local M-pool** stores the main storage in local CXL memory pool and **Main Remote M-pool** stores the main storage in remote CXL memory pool. Additionally, we double the CXL bandwidth on the server side to see the performance improvement when moving the main storage to CXL memory pool. First, we create two `fsdax` devices in our test machine, one per downstream device. Then, we create a mapped device by interleaving these two `fsdax` devices using the Linux device mapper. In this configuration, we put **x2** at the end of the configuration name.

Unlike TPC-C, TPC-DS exhibits noticeable performance degradation influenced by latency and available bandwidth when moving table data to the CXL memory pool. TPC-DS workloads primarily exhibit sequential access patterns, allowing hardware prefetching to mitigate the effects of high latency. Our results indicate that performance is more sensitive to available bandwidth than to latency. Doubling the bandwidth leads the performance improvement, resulting in only a single-digit percentage performance degradation compared to the baseline. Thus, the increased bandwidth sufficiently accommodates the required CXL traffic, as illustrated in Fig. 5. Nevertheless, a slight performance difference exists between configurations with CXL memory boxes connected to local versus remote sockets. This is attributable to a minor portion of random accesses to dictionaries in the main storage, which are latency-sensitive. Consequently, the latency difference affects performance slightly when the main storage is relocated.

## 5.2 Fast Restart with CXL Memory Pool

To evaluate the performance of fast restart with CXL memory pool, we use the same system configuration described in Section 5.1 using TPC-DS data with a scale factor of 100. SAP HANA provides a fast restart option [22], which makes it possible to reuse main data fragments after an SAP HANA service restarts without the need to reload the main data from the persistent storage or disk. To enable fast restart, the main storage (table

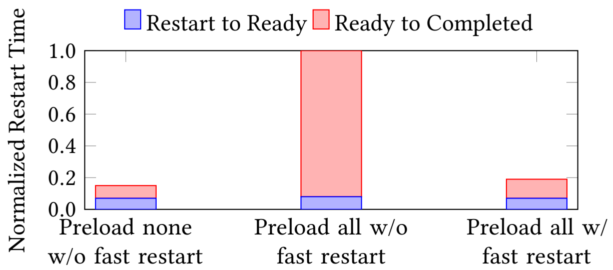


**Figure 6: Normalized Restart Time**

data) must be placed in the CXL memory pool. When the database instance restarts, it can directly attach the existing main storage loaded in the CXL memory pool instead of reloading the table data from the persistent data volume to local memory, because the contents of the CXL memory pool remain intact across restarts. As a result, SAP HANA significantly reduces restart time, a capability referred to as fast restart.

We assess restart performance by measuring SAP HANA's restart time after a controlled shutdown, leveraging SAP HANA's persistent memory feature to place the main storage in the CXL memory pool. During the restart, the database records three timestamps in the trace log: (1) when the instance begins the restart (Restart), (2) when the instance completes its initialization and takeover (Ready), and (3) when data preload is completed (Completed). To measure the elapsed time, we collect these timestamps and calculate the elapsed time. Since the size of the delta storage is usually much smaller than that of the main storage in real-world scenarios, it is assumed that the additional time to redo the delta log is negligible. Thus, we compare the restart time when the delta storage is fully merged to the main storage.

Figure 6 shows the normalized restart time when the database instance restarts. In the first configuration, fast restart is disabled and no table data are preloaded, which means each table data will be loaded on demand when a query first accesses it. While this minimizes restart time to database initialization, the first query execution will incur additional time due to table loading. In the second configuration, fast restart is disabled but all table data are configured to be preloaded. The query processing service is enabled only after all table data are fully loaded into local memory. Because the data reside on disk-backed data volumes, the restart time is dominated by this preloading phase. The third configuration activates fast restart after we move table data to a CXL memory pool, and all the data is set to preload during the restart. Even though preloading is set, the actual table loading is not executed as the table data already reside in the CXL memory pool. Thus, it saves 87% of data preloading time, totally resulting in a 5.25x acceleration of restart time, compared to the second configuration. In comparison to the first configuration, the restart time increases by 27%, primarily due to the initialization required to validate data consistency and reconstruct metadata in local memory. During the restart, the entire table data is fully accessible in the memory, ensuring that any subsequent queries can proceed without the need for additional data loading unlike the first configuration.

## 5.3 Allocating Temporary Tables

As another use case of CXL memory pool, we evaluate allocating temporary tables to CXL memory pool as mentioned in Section 4.3. We use internal temporary tables to hold intermediate

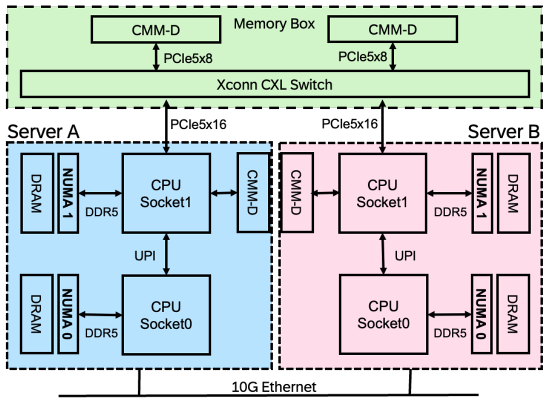


**Figure 7: System Configuration with Two Servers**

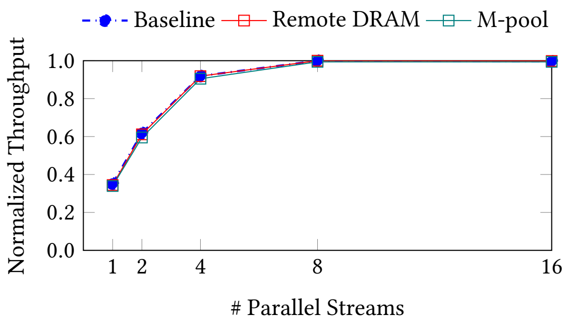


**Figure 8: Performance Comparison of TPC-DS SQL scripts**

results produced by SQL scripts. To mitigate out-of-memory (OOM) conditions by huge temporary tables, we can allocate temporary tables in CXL memory pool. We measure the performance impact of allocating these temporary tables in the CXL memory pool.

5.3.1 *System configuration.* To evaluate dynamic memory allocation across multiple servers using a CXL memory pool, we deploy a commercial CXL memory box integrating an Xconn CXL switch and Samsung CMM-D memory devices. The switch's multiple ports are used to distribute memory traffic across servers in parallel. The test environment comprises two compute nodes as shown in Fig. 7. Server A is equipped with two Emerald Rapids Intel Xeon Platinum 8568CXL CPUs (96 logical cores per socket); each socket has eight memory channels populated with 128 GB DIMMs, yielding 1 TB of local DRAM per socket. Server B is equipped with two Emerald Rapids CPUs (112 logical cores per socket); each socket has eight memory channels populated with 64 GB DIMMs, yielding 512 GB of local DRAM per socket. We enable expanded memory for both servers by 128 GB by configuring dynamic allocation from the shared pool to bring CXL memory regions online or offline as needed. Our workloads consist of SQL scripts derived from the top five TPC-DS queries with the largest temporary table footprints.

5.3.2 *Experimental results.* Prior to the performance evaluation, we validate dynamic capacity changes in the CXL memory pool using Intel® Memory Latency Checker (MLC) and modified TPC-DS SQL scripts. We observe no measurable delays, and no additional lifecycle operations, such as a system or database restart, are required when attaching or detaching memory regions or

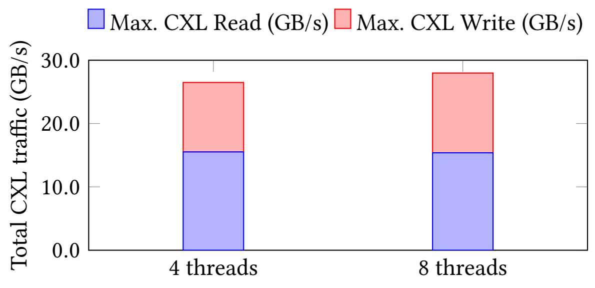
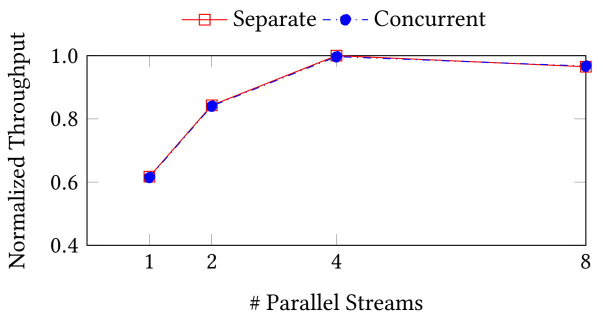
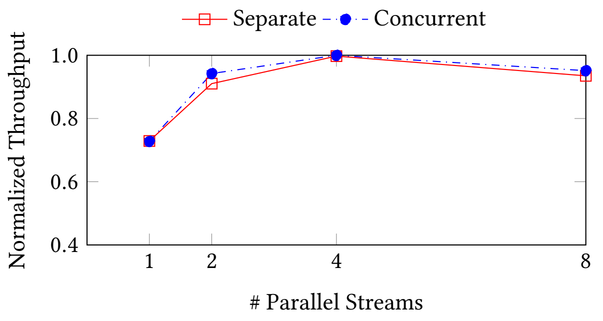


**Figure 9: CXL Traffic of TPC-DS SQL scripts**



**(a) Server A**



**(b) Server B**

**Figure 10: Performance Interference in CXL Memory Box**

modifying the size of each memory region in the CXL memory pool. To assess the performance impact of allocating temporary tables in the memory pool, we execute the workloads on Server A under three memory placements for temporary table allocation: **Baseline** using local DRAM, **Remote DRAM**, and **M-pool** using the CXL memory pool. As shown in Fig. 8, performance is effectively unchanged across these configurations. The selected scripts are dominated by sequential access patterns that benefit from prefetching; combined with the low CXL traffic observed, this makes the workload largely insensitive to latency differences across the memory tiers. Bandwidth usage on the single CXL device remains marginal, keeping the link and device queues well below saturation. With less-intensive reads/writes and low CXL traffic as shown in Fig. 9, contention is negligible and no interference is observed.

*5.3.3 Performance interference.* To examine interference between servers sharing a remote CXL memory pool, we connect two servers and maintain the total CXL bandwidth while both allocate and access memory concurrently. Server A uses the first 128 GB range located at the first CXL memory device and Server B uses the remaining 128 GB range at the other CXL memory device. Under this setup, all CXL traffic passes through the same CXL switch within the memory box during benchmark execution. To evaluate the interference probably by the CXL switch, we measure the performance in two conditions: (1) when both servers execute the scripts concurrently and (2) when they execute the scripts separately. As shown in Fig. 10, no noticeable performance degradation is observed, indicating that shared memory access through the CXL memory pool does not introduce any interference between two servers. Thanks to the CXL switch's ability to relay traffic in parallel across devices, performance overhead remains negligible as long as there are no other resource conflicts, such as device bandwidth contention.

## 6 Discussion

This section emphasizes the design factors in utilizing CXL memory pool to build enterprise software systems, taking into account the performance impact of increased latency and limited bandwidth from our experimental results. It also discusses the anticipated benefits of CXL memory pool from the perspective of enterprise IMDBMSs.

### 6.1 Performance Impact of CXL Memory Pool

Our study demonstrates that the increased latency and limited memory bandwidth of CXL memory pool can adversely affect performance, depending on the workload and memory access patterns. For lightweight workloads causing small CXL bandwidth usage, accessing CXL memory pool usually results in negligible performance impact regardless of whether the workloads are OLTP or OLAP. In contrast, heavy workloads with substantial bandwidth demands may experience noticeable performance degradation. However, when these workloads involve mainly sequential memory access patterns, CPU prefetching effectively mitigates the latency penalty, keeping the performance impact within acceptable bounds. Conversely, for heavy workloads with random memory access patterns, it is strongly recommended to use local memory instead of CXL memory pool for optimal performance, as the benefits of prefetching diminish and CXL latency becomes more pronounced.

### 6.2 Expected Benefits

CXL enables a more flexible system architecture through adaptable memory configurations at system-wide, query-wise, or mixed levels with reduced TCO.

*6.2.1 System-wide perspective.* CXL facilitates the disaggregation of compute and data nodes. For instance, table data can be relocated to a memory pool while compute nodes retain only operational memory in local DRAMs. As data volumes grow, CXL offers architectural flexibility by supporting scale-up with high-capacity DIMMs, memory expansion to the memory pool, or a combination of both. These adaptive options contribute to improved performance and a lower server TCO compared to simply scaling out by adding more servers. It minimizes communication overhead between servers and avoids over-provisioning during the initial setup of server infrastructure. Additionally, CXL memory pool includes a persistent memory feature backed by an independent power source for the memory pool, accelerating the restart of IMDBMSs. These system-wide flexible architectures reduce fragmentation and wasted memory, leading to more efficient resource utilization. They also enhance system resilience and simplify memory management tasks for system administrators and developers.

#### 6.2.2 *Query-wise perspective.*

The dynamic assignment of intermediate results during large query executions allows for selective memory usage, helping to maintain a low TCO. For example, as demonstrated in our use cases, memory can be offloaded to CXL pooled memory to accommodate substantial temporary workloads, such as those encountered during month-end or year-end reporting. On-demand memory allocation helps avoid: (i) unnecessary overprovisioning of large virtual machines from the outset, and (ii) management overhead caused by frequent out-of-memory (OOM) occurrences. This approach can be easily extended to other use cases, such as maintaining static cached views to store intermediate results across multiple queries.

#### 6.2.3 *Low TCO.*

CXL memory pool offers unprecedented flexibility and scalability in memory systems for enterprise software substantially reducing total cost of ownership (TCO) of server infrastructure by overcoming physical limitations of DIMM slots. CXL enables flexible memory configurations across multiple servers, reducing stranded memory through shared resource utilization and improving overall memory efficiency. Its dynamic memory expansion capability eliminates the need for over-provisioning, allowing the incremental growth of memory capacity in response to workload demands rather than committing to excessive capacity. Furthermore, for lighter workloads, the memory pool can be configured with affordable, cost-effective DIMMs of lower capacity to further reduce the infrastructure costs. Collectively, these strategies significantly contribute to lowering server TCO [14].

## 7 Related Work

With the advent of CXL, main memory can be expanded in two complementary ways: capacity expansion and bandwidth expansion. Capacity expansion uses CXL-attached memory devices to extend the total addressable memory space beyond on-board DRAM, enabling larger in-memory datasets and reducing data spilling. In contrast, bandwidth expansion seeks to increase effective memory throughput by interleaving data accesses across DRAM and CXL memory channels. Recent work on CXL memory capacity expansion has demonstrated its potential to extend the addressable memory space of in-memory and data-intensive systems. Ahn et al. [3] examined practical CXL memory use cases for SAP HANA, showing that database components such as column stores and intermediate data can be placed in CXL-attached memory with modest performance degradation, enabling dynamic capacity scaling. Earlier, [1] and [13] established the feasibility of integrating CXL-based memory into existing in-memory engines while maintaining near-baseline throughput. Building on this, Riekenbrauck et al. [21] proposed a three-tier buffer manager that incorporates CXL device memory as an intermediate tier between DRAM and SSD, improving data throughput and reducing spill-over delays. At the hardware characterization level, studies such as Sun et al. [25] and Weisgut et al. [29], analyzed real devices and highlighted latency, bandwidth, and page-allocation trade-offs critical for large-scale deployments. While prior studies have mainly focused on CXL memory capacity expansion, Lebedev et al. [12] provide the first systematic evaluation of bandwidth expansion in analytical workloads, revealing that software-managed transparent DRAM–CXL interleaving often harms performance and highlighting the need for adaptive, hardware-conscious approaches.

There has been several recent work exploring how CXL-based memory pooling can reduce stranded capacity and enable elastic provisioning of memory across servers, making it a promising approach for cloud and HPC environments. PolarDB [30] demonstrates the first production-grade integration of Compute Express Link (CXL) memory into a commercial cloud-native DBMS, enabling large disaggregated memory pools that reduce cost and improve elasticity while outperforming RDMA-based solutions. While its approach utilizes CXL memory pool as a page buffer pool, our paper presents the investigation results on a broader range of use cases of CXL memory pools for IMDBMSs.

Pond [14] proposes a CXL-based memory pooling system for cloud platforms that dynamically allocates shared memory across servers, showing that pooling across a modest number of sockets yields substantial cost savings with minimal performance impact. Wahlgren et al. [28] evaluate emerging CXL-based memory pooling for HPC workloads, showing that scientific applications can tolerate a large fraction of their memory footprint in pooled CXL memory with limited performance degradation, though interference effects remain a challenge. Octopus [4] proposes a topology for CXL memory pooling in which each host connects to a bounded number of pooling devices (instead of full connectivity), yet ensures that every pair of hosts shares at least one pooling device, achieving similar memory savings to conventional designs with much lower cost and reduced latency overheads.

## 8 Conclusion

This study evaluated the performance and architectural implications of integrating a CXL memory pool in enterprise in-memory DBMSs. Our findings reveal that while the increased latency and limited bandwidth of CXL memory can negatively influence performance under certain conditions, the impact varies substantially across workloads.

Our work also highlighted the architectural flexibility and cost efficiency enabled by CXL-based memory pooling. By decoupling memory from compute nodes, CXL allows dynamic memory expansion and resource sharing across servers. This flexible configuration supports both system-level disaggregation and query-level adaptive memory assignment, leading to improved resource utilization and lower total cost of ownership (TCO). Furthermore, CXL's persistent memory capability accelerates system recovery and simplifies memory management for large-scale enterprise workloads.

Overall, CXL memory pooling presents a promising path toward scalable and cost-efficient memory architectures for next-generation in-memory database systems. While performance sensitivity to workload intensity and access pattern must be carefully managed, the overall benefits in flexibility, utilization, and resilience make CXL a key enabler of disaggregated and elastic data infrastructures.

For future work, we plan to investigate sharing the intermediate results and table data across instances over CXL memory pools, with cache coherency expected in upcoming CXL standards.

## References

[1] 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. doi:10.1145/3533737.3535090

[2] Minseon Ahn, Thomas Willhalm, Donghun Lee, Norman May, Jungmin Kim, Daniel Ritter, and Oliver Rebholz. 2025. Exploiting Locality in Flat Memory

with CXL for In-Memory Database Management Systems. In *Proceedings of the 21st International Workshop on Data Management on New Hardware (DaMoN '25)*. Association for Computing Machinery, New York, NY, USA, Article 1, 9 pages. doi:10.1145/3736227.3736230

[3] Minseon Ahn, Thomas Willhalm, Norman May, Donghun Lee, Suprasad Mutalik Desai, Daniel Booss, Jungmin Kim, Navneet Singh, Daniel Ritter, and Oliver Rebholz. 2024. An Examination of CXL Memory Use Cases for In-Memory Database Management Systems using SAP HANA. *Proceedings of the VLDB Endowment* 17, 12 (2024), 3827–3840.

[4] Daniel S. Berger, Yuhong Zhong, Fiodar Kazhamiaka, Pantea Zardoshti, Shuwei Teng, Mark D. Hill, and Rodrigo Fonseca. 2025. Octopus: Scalable Low-Cost CXL Memory Pooling. arXiv:2501.09020 [cs.AR] https://arxiv.org/abs/2501.09020

[5] CXL. 2023. CXL 3.1 Specification. https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-3.1-Specification.pdf

[6] CXL. 2024. CXL 3.2 Specification. https://computeexpresslink.org/wp-content/uploads/2024/12/CXL_3.2-Spec-Announcement_FINAL-1.pdf

[7] CXL. 2024. Past CXL Specifications. https://computeexpresslink.org/past-cxl-specifications/

[8] 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. doi:10.1561/1900000058

[9] Reece Hayden and Paul Schell. 2024. *Opportunities and Challenges For Compute Express Link (CXL)*. Technical Report. ABI research.

[10] Yibo Huang, Newton Ni, Vijay Chidambaram, Emmett Witchel, and Dixin Tang. 2025. Pasha: An efficient, scalable database architecture for cxl pods. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*.

[11] Kihong Kim, Hyunwook Kim, Jin Su Lee, Taehyung Lee, Mihnea Andrei, Alexander Böhm, Ralf Dentzer, Heiko Gerwens, Irena Kofman, Norman May, Daniel Ritter, and Guido Moerkotte. 2025. Enterprise Application-Database Co-Innovation for Hybrid Transactional/Analytical Processing: A Virtual Data Model and Its Query Optimization Needs. In *Companion of the 2025 International Conference on Management of Data* (Berlin, Germany) *(SIGMOD/PODS '25)*. Association for Computing Machinery, New York, NY, USA, 485–498. doi:10.1145/3722212.3724436

[12] Georgiy Lebedev, Hamish Nicholson, Musa Ünal, Sanidhya Kashyap, and Anastasia Ailamaki. [n. d.]. Demystifying CXL Memory Bandwidth Expansion for Analytical Workloads. *Proceedings of the VLDB Endowment. ISSN* 2150 ([n. d.]), 8097.

[13] Donghun Lee, Thomas Willhalm, Minseon Ahn, Suprasad Mutalik Desai, Daniel Booss, Navneet Singh, Daniel Ritter, Jungmin Kim, and Oliver Rebholz. 2023. Elastic Use of Far Memory for In-Memory Database Management Systems. In *DaMoN (DaMoN '23)*. ACM, 35—-43. doi:10.1145/3592980.3595311

[14] 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. doi:10.1145/3575693.3578835

[15] Norman May, Alexander Böhm, Daniel Ritter, Frank Renkes, Mihnea Andrei, and Wolfgang Lehner. 2025. SAP HANA Cloud: Data Management for Modern Enterprise Applications. In *Companion of the 2025 International Conference on Management of Data* (Berlin, Germany) *(SIGMOD/PODS '25)*. Association for Computing Machinery, New York, NY, USA, 580–592. doi:10.1145/3722212.3724452

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

[17] 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 Hot Chips 34 Symposium (HCS)*. 1–27. doi:10.1109/HCS55958.2022.9895633

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

[19] 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. doi:10.14778/2733004.2733074

[20] 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. doi:10.1007/978-3-319-15350-6_7

[21] Niklas Riekenbrauck, Marcel Weisgut, Daniel Lindner, and Tilmann Rabl. 2024. A Three-Tier Buffer Manager Integrating CXL Device Memory for Database Systems. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. 395–401. doi:10.1109/ICDEW61823.2024.00063

[22] SAP. 2024. *SAP HANA Fast Restart Option*. https://help.sap.com/docs/SAP_HANA_PLATFORM/6b94445c94ae495c83a19646e7c3fd56/ce158d28135147f099b761f8b1ee43fc.html

[23] 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 (aug 2019), 2047–2058. doi:10.14778/3352063.3352123

[24] 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. doi:10.1145/2213836.2213946

[25] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, et al. 2023. Demystifying cxl memory with genuine cxl-ready systems and devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 105–121.

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

[27] TPC-DS. 2023. *TPC-DS*. https://www.tpc.org/tpcds/

[28] Jacob Wahlgren, Maya Gokhale, and Ivy B. Peng. 2022. Evaluating Emerging CXL-enabled Memory Pooling for HPC Systems. In *2022 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE. doi:10.1109/mchpc56545.2022.00007

[29] Marcel Weisgut, Daniel Ritter, Pinar Tözün, Lawrence Benson, and Tilmann Rabl. 2025. CXL Memory Performance for In-Memory Data Processing. *Proc. VLDB Endow.* 18, 9 (Sept. 2025), 3119–3133. doi:10.14778/3746405.3746432

[30] Xinjun Yang, Yingqiang Zhang, Hao Chen, Feifei Li, Gerry Fan, Yang Kong, Bo Wang, Jing Fang, Yuhui Wang, Tao Huang, Wenpu Hu, Jim Kao, and Jianping Jiang. 2025. Unlocking the Potential of CXL for Disaggregated Memory in Cloud-Native Databases. In *Companion of the 2025 International Conference on Management of Data* (Berlin, Germany) *(SIGMOD/PODS '25)*. Association for Computing Machinery, New York, NY, USA, 689–702. doi:10.1145/3722212.3724460

[31] Yuhong Zhong, Daniel S. Berger, Carl Waldspurger, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D. Hill, Mosharaf Chowdhury, and Asaf Cidon. 2024. Managing Memory Tiers with CXL in Virtualized Environments. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 37–56. https://www.usenix.org/conference/osdi24/presentation/zhong-yuhong