Pasha: An Efficient, Scalable Database Architecture for CXL Pods

Yibo Huang, Newton Ni, Vijay Chidambaram, Emmett Witchel, Dixin Tang
The University of Texas at Austin
{ybhuang,nwtnni,vijayc,witchel,dixin}@cs.utexas.edu

Abstract

Memory connected via Compute Express Link (CXL) presents a new opportunity for building efficient and scalable databases. A *CXL Pod* is a small number of independent hosts connected via CXL to shared memory (termed CXL memory). A small part of the CXL memory is hardware-cache-coherent across hosts. This paper proposes Pasha, a new database architecture for the CXL Pod, that aims to efficiently leverage the fast synchronization enabled by the hardware-cache-coherence of CXL memory and the low latency and high bandwidth of local DRAM in each host. To achieve this goal, Pasha divides and stores data into partitions, with each partition owned by a single host, and a shared region that every host has access to. Then, Pasha uses CXL memory to efficiently synchronize concurrent accesses to the shared region across hosts and stores each partition of data in local DRAM to leverage its low latency without cross-host synchronization overhead. In addition, Pasha ensures that each host completes its transactions by directly reading or writing data in either its own partition or the shared region, circumventing multi-host transactions and two-phase commit (2PC). Pasha obtains the best of partition-based distributed databases and single-node shared-memory databases; preliminary results show it outperforms both architectures.

1 Introduction

Scaling a single-node OLTP database to multiple nodes with minimal overhead is a holy-grail problem in the database community and has been studied for decades [8, 17, 20, 24, 36, 42, 46, 48, 52, 53, 57]. Efficiency and scalability are critical not only for supporting high transaction throughput and large volumes of data, but also for providing the elasticity required to handle fluctuating transaction workloads. These goals are increasingly crucial for today's cloud databases [3–5], which elastically provision resources for dynamic workloads and charge customers only for consumed resources.

The traditional approach to scaling a single-node database is partitioning data across multiple hosts (as Figure 1(a) shows), and letting each host process all read/write operations to its partition [17, 24, 36, 42]. This partition-based architecture works well when the workload mainly includes *single-host transactions*, i.e., transactions whose read/write operations only access the data in a single partition and can be completed by a single host. However, for a transaction that needs to access multiple partitions (i.e., *multi-host transactions*), performance is degraded because read/write operations and transactions must be coordinated across multiple hosts, which usually requires numerous cross-host message exchanges and costly protocols such as two-phase commit (2PC). (We avoid the term distributed transaction because it is not used consistently in the literature to distinguish single-host from multi-host transactions.) As a result, the database performance drops significantly as the number of multi-host transactions increases, even after decades of optimizations [33, 34, 40, 51, 55].

A New Opportunity: CXL Pod. We believe the emergence of Compute Express Link (CXL) provides a new opportunity for efficiently scaling databases without introducing multi-host transactions. One use for CXL is to allow a small number of hosts (e.g., sixteen hosts [31]) to be physically connected via the PCIe bus to a shared memory module, called *CXL memory*. CXL is a serial protocol to allow CPUs to directly access memory and it provides a much lower latency compared to RDMA-based shared memory (see §2). The CPU issues regular, cacheable loads and stores to CXL memory, unlike RDMA-based access, which requires specialized verbs and a support library. Additionally, the new features of CXL 3.0 and 3.1 [1, 7] allow CXL memory to support cross-host hardware cache coherence through back-invalidations based on a snoop filter, which is not supported by RDMA networks. Such a collection of machines that share CXL memory is called a **CXL pod** [47, 56].

Providing hardware cache coherence for the entire CXL memory address space will be too expensive to be practical, as shown by AMD and others [14, 23]. Based on the conversation with our industrial partners and CXL vendors, such as Microsoft, Samsung, and Micron, *the hardware cache-coherent region will be hundreds of MBs* while the capacity of CXL memory will be terabytes.

Despite including a small number of hosts, a CXL pod offers abundant computing resources for processing OLTP workloads. For instance, Intel's recently announced Sierra Forrest processor will feature 288 cores [9]. By building a CXL pod with 16 hosts, each equipped with a single Sierra Forrest processor and local DRAM, and physically connected to a CXL memory module, the pod has a total of 4,608 cores that share CXL memory.

Key Insights of Designing Databases for CXL Pods. The CXL pod makes feasible a new class of databases that get the best of both partition-based distributed databases and single-host shared-memory databases. There are two key insights. First, shared CXL memory, even if only a small part of it has hardware cache coherence, can significantly increase transaction throughput by turning multi-host transactions into single-host transactions. The shared region can efficiently synchronize the accesses across multiple hosts using shared data structures and atomic operations similar to a shared-memory database, which is more efficient than sending messages over the network to coordinate transactions (e.g., 2PC). Second, each host in the CXL pod has local DRAM and synchronization across processes/threads within a single host is more efficient than synchronization across multiple hosts. Therefore, it is desirable to leverage data partitioning from partition-based databases to reduce the synchronization overhead across hosts.

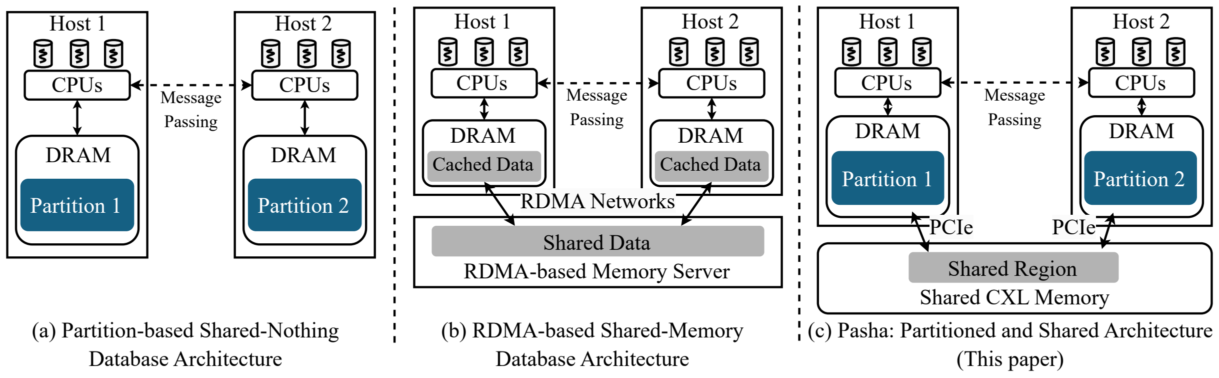

Figure 1: An illustration of comparing two existing architectures and Pasha (this paper) for scaling a single-node OLTP database

One challenge for our design is properly modeling the cost for atomics (e.g., test-and-set) that support inter-host cache coherence as we do not yet have hardware prototypes for inter-host cache coherence.

A New Database Architecture. In this paper, we propose Pasha, a novel database architecture that takes advantage of both Partitioned and shared database architectures for scaling a single-node shared-memory database on a CXL pod without introducing multi-host transactions. Figure 1(c) shows an example configuration of Pasha. In this architecture, data is divided and stored into *partitions* (e.g., Partition 1 and 2 in Figure 1(c)) and *a shared region*, which include disjoint sets of data. Each partition stores data assigned to a specific host and this host handles all read/write operations to its partition while the shared region stores data that every host has access to. Each host stores their partition in local DRAM and the shared region is stored in CXL memory.

To eliminate multi-host transactions, Pasha ensures all data tuples required by a transaction are either in a host's local partition or the shared region by dynamically moving data across the shared region and partitions (§3). This way, the database can complete all read/write operations of a transaction using a single host and leverage a single-node concurrency control protocol to coordinate transaction executions across hosts. For example, in Figure 1(c), Pasha ensures Host 1 only needs to access Partition 1 and the shared region to complete a transaction. When using two-phase locking (2PL) for concurrency control, before Host 1 writes a data tuple in the shared region it will acquire a lock, which is also stored in the shared region. The other hosts (e.g., Host 2 in Figure 1(c)) will later observe this lock if they need to access the same data tuple.

Pasha wants the best of both worlds: the fast coordination of shared memory and the reduced cross-host synchronization and conflicts enabled by partitioned data. Allowing transactions to access data either in a local partition or in the shared region enables each host to execute and coordinate all transaction operations on their own, which is the major benefit Pasha derives from a shared-memory architecture. On the other hand, partitioning data along with transactions significantly reduces the amount of data that requires coordination across hosts [18, 40], a key benefit from a partition-based architecture. By placing partitions in local DRAM, a partition-based architecture improves end-to-end performance by leveraging the lower latency and higher bandwidth of local DRAM relative to CXL memory. Our preliminary results have shown that Pasha has up to 5.9× and 1.4× higher throughput than a partition-based shared-nothing architecture and a completely shared, shared-memory architecture, respectively.

Research Challenges. Exploiting the performance benefits of Pasha requires addressing two broad challenges: (1) how to best utilize the hardware cache-coherent region of CXL memory, and (2) how to leverage the low latency and high bandwidth of local DRAM (compared to CXL memory). For the first challenge, we need to leverage the (limited) hardware cache-coherent region for synchronization primitives like latches and design an efficient software cache-coherence protocol to support a shared region larger than the hardware cache-coherent region. For the second challenge, we want to maximize the size of data stored in partitions to efficiently leverage the high performance of local DRAM. A partition of data is owned by a host and can be stored in local DRAM without synchronization across hosts. Effective partitioning in Pasha requires a new data partitioning algorithm and a data movement protocol between the shared region and partitions. Furthermore, we need to adapt existing concurrency-control protocols (e.g., MVCC [27]) to efficiently support the data movement protocol.

In addition, the abundant computing resources (e.g., possibly over 4,000 cores as discussed earlier) in a CXL pod introduces new challenges for database designs: addressing high concurrency and tolerating partial failures (e.g., a database process or a host fails, but the other processes continue to execute transactions). Finally, CXL memory provides a new research opportunity for auto-scaling and achieving efficient elasticity.

Related Work. There is a recent line of research that adopts fast RDMA networks along with a shared and disaggregated memory architecture to avoid multi-host transactions and 2PC in partition-based shared-nothing databases [8, 20, 46, 48, 52, 53, 57]. In this architecture, all hosts can read and write data in shared memory through RDMA networks without requiring multi-host transactions [46, 52, 53], as shown in Figure 1(b). These systems require global synchronization for all data while Pasha requires global synchronization only for shared data in CXL memory. In addition, the latency for memory access through RDMA networks can be one to two orders of magnitude higher than local DRAM and CXL memory, as shown in §2. While some papers consider caching data in local DRAM to reduce the cost of fetching data from the shared memory [8, 48], they require a potentially costly data coherence protocol across multiple hosts due to the lack of hardware cache-coherence support for RDMA. Pasha, instead, aims to leverage the low latency of local DRAM and CXL memory and the efficiency of the limited cross-host hardware cache coherence of CXL memory.

Two papers discuss the research challenges of leveraging CXL memory to scale databases, but neither of them propose a concrete database architecture for efficiently utilizing CXL memory [22, 30].

Contributions. This paper makes the following contributions: 1) we propose Pasha, a novel database architecture that has the best of partitioned and shared database architectures; 2) we describe the designs of Pasha and present the research challenges (§3); 3) we demonstrate the potential performance benefits of Pasha using preliminary experiments (§4).

2 CXL Pod Preliminaries

In this section, we briefly describe the architecture of CXL pods and show the performance characteristics of CXL memory relevant to database design. A more comprehensive background discussion can be found in prior papers [29–31].

2.1 CXL memory and a CXL pod

CXL is an open standard that facilitates low-latency and high-bandwidth interconnections between CPUs, devices, and heterogeneous memory based on PCIe 5.0 and 6.0. It includes three protocols. `CXL.io` is a PCIe-based block input/output protocol. `CXL.cache` allows peripheral devices to access and cache host CPU memory while `CXL.mem` allows host CPU to access peripheral memory with load/store commands [6]. Shared CXL memory is based on `CXL.io` and `CXL.mem` (i.e., a Type 3 device [6]).

CXL specifications define different capabilities of CXL protocols [6]. CXL 1.1 allows a single node to access PCIe-connected expanded memory in a cache-coherent manner. CXL 2.0 adds support for coarse-grained memory pooling where multiple nodes access disjoint CXL regions. CXL 3.0 adds fine-grained sharing and hardware-supported cache coherence across hosts using back-invalidates.

A CXL pod, as shown in Figure 2, includes a limited number of machines (e.g., 16) and a shared CXL memory module, connected by a CXL switch. We focus on a small number of machines because they already provide rich computing resources (e.g., up to 4,608 cores as shown in the introduction) and a prior paper shows that this scale of configuration has small latency and bandwidth penalties compared to a single host [31]. We assume that a limited region of the CXL memory module (e.g., 256MB) is hardware cache-coherent across different hosts using back-invalidates [7], while the rest of the memory relies on software protocols to provide coherence across hosts (§3).

2.2 Performance characteristics of CXL memory

Our evaluation (§4) uses a hardware prototype CXL module. The evaluation machine includes two Intel Xeon 8460H CPUs, 1 TB local DRAM, and a 128 GB CXL 1.1 memory device. Both local DRAM and CXL memory use DDR5 4800 DRAM. The CXL memory

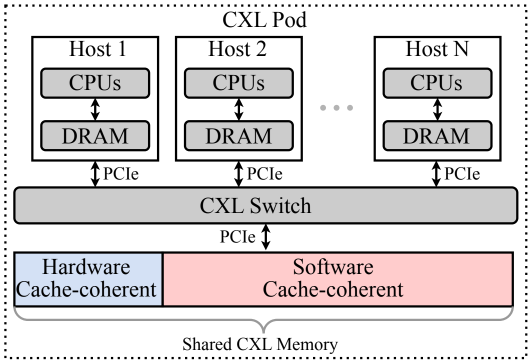

Figure 2: An example configuration of a CXL pod

module has a single memory channel and is connected to the CPU via a PCIe 5.0 x8 link. We measure the latency and single-channel bandwidth of local DRAM and CXL memory using Intel's Memory Latency Checker [2] under a 3:1 read-write ratio.

We find that the latency of the tested CXL memory is around 2.3× higher than local DRAM and its bandwidth is 58% of the single-channel bandwidth of local DRAM. Note that at the request of the hardware manufacturer, we only provide normalized performance, not absolute measurements. Nevertheless, our measurements are similar to previous systems measurements [31, 41, 56] that show local DRAM with higher performance than CXL memory, necessitating the use of local DRAM in high-performance systems. In addition, the latency of CXL memory is one to two orders of magnitude lower than RDMA-based disaggregated memory [21], showing the significant benefits of exploiting CXL memory to design next-generation databases.

3 Pasha Architecture and Challenges

In this section, we describe the Pasha architecture and the research challenges for realizing Pasha.

3.1 Pasha architecture

The Pasha architecture aims to leverage the fast synchronization of hardware-coherent CXL memory to eliminate multi-host transactions, thereby significantly increasing transaction throughput. It also exploits the low latency and high bandwidth of local DRAM to further accelerate transaction processing.

To achieve its goals, Pasha divides and stores data into disjoint partitions. Each host owns a partition and processes all accesses to it, and there is a single, shared region that all hosts can access. Pasha ensures that if a host reads/writes a data tuple that does not exist in its own partition, then the data tuple is present in the shared region. Pasha coordinates how tuples move between host-owned partitions and the shared region (described below). Because all tuples are either in a host's owned partition or the shared region, a host can process all operations pertaining to a transaction, eliminating multi-host transactions. Partitions owned by a single host are stored in local DRAM to leverage its low latency and high bandwidth. The shared region is stored in CXL memory.

A database based on Pasha initially partitions data across hosts and stores each partition in each host's local DRAM [18, 37, 40]. If a host needs to access a data tuple in a partition owned by another host, it asks the other host (e.g., by sending a message) to move the data tuple along with metadata information (e.g., locks) to the shared region, which is stored in CXL memory. The host then adopts a single-host concurrency control (CC) protocol to read/write this data tuple because it can access it directly via a pointer to the shared region. In addition, while a data tuple of a host's partition is resident in the shared region, the owning host uses the same CC protocol as other hosts to access the tuple from the shared region. For example, if 2PL is adopted, a host can acquire a lock on a data tuple in the shared region. This lock, also stored in the shared region, will be observed by other hosts, ensuring 2PL is followed for concurrent transactions across hosts. Finally, data in the shared region will be moved back to their original host's owned partition in local DRAM based on a system policy that considers access patterns to the shared region.

The database ensures durability and atomicity via logging and checkpoints. Parallel logging might be required to prevent logging from being the performance bottleneck [45, 54]. One interesting challenge for ensuring durability and atomicity is how to design new parallel logging and recovery protocols for dealing with partial failures: the failure of a single host or process does not impact the functioning of other hosts/processes and CXL memory.

3.2 Research challenges

We now discuss the challenges for realizing Pasha in the CXL pod.

(1) Accessing Data under Movement. To realize Pasha, the database needs a novel mechanism and policy for moving data between partitions and the shared region. The challenge is to ensure that transactions safely and efficiently read and write data when the data moves between these regions. For example, if one host tries to commit a transaction that updates data in its partition while another host asks to move this data item to the shared region, the database must resolve the conflict safely. One interesting direction is to co-design the data movement mechanism with existing access methods (e.g., a B-Tree index over a shared region) and transaction protocols to efficiently execute read/write operations while maintaining transaction semantics. An additional challenge is to design a novel policy that minimizes the size of data moved between local DRAM and CXL memory to improve performance.

(2) Software Cache Coherence. The recent report from AMD [23] and our discussions with industry partners show that the size of the hardware cache-coherent region could be limited; as a result, there may be frequent data movement between the shared region and partitions, leading to performance overheads. To compensate for a limited region of hardware cache-coherence, the database should use a software cache-coherence protocol for the remaining memory capacity. A software coherence protocol explicitly tracks data in order to minimizes data flushes from the CPU cache back to CXL memory, and to maximize the probability that needed data is already in the CPU cache.

While software-based cache-coherence has generally been expensive [39], we believe a promising direction is to design a software cache-coherence protocol that assumes a region of hardware cache-coherence and is tailored to the database (e.g., by using a coarser granularity than a cache line). For instance, one intuitive idea is to use the hardware cache-coherent region to store metadata that requires intensive synchronization and the software cache-coherent region to store and track tuples, which are often larger than cache lines.

(3) Supporting MVCC. Multi-version concurrency control (MVCC) is widely adopted in major database products due to its flexibility of allowing more serializable transaction schedules than traditional protocols (e.g., a read-only transaction never aborts in MVCC). Therefore, it is important to support MVCC in Pasha. The key challenge for supporting MVCC stems from the cost of moving all versions of a data tuple between partitions and the share region.

To address this challenge, one direction is only moving the requested version to the shared region and selectively moving the versions that will be useful for future transactions back to partitions. Therefore, different versions of a data tuple may stay in different places (i.e., some versions are in the shared region, stored in CXL memory, while some versions are in a partition, stored in local DRAM). The challenge here is how to commit a transaction if it reads an old version of data. For example, a host may read a version in local DRAM while another host may create a new version in CXL memory.

Our key insight is that MVCC protocols provide the flexibility of committing a transaction even when it reads an old version [16]. In MVCC, a write to a data tuple creates a new version instead of performing an in-place update, so it does not invalidate the old version, which may be stored in a different place. Additionally, when a transaction reads an old version from the local cache, it can still maintain serializability and commit if the MVCC protocol is carefully designed. The research opportunity, therefore, is to design an MVCC protocol that can efficiently validate transactions that read old versions of data and maintain serializability.

(4) Data Partitioning. Previous papers on optimizing partition-based distributed databases propose data partitioning algorithms to minimize the number of multi-host transactions and maintain load balance [18, 40]. However, in Pasha, multi-host transactions are converted into single-host transactions that access a partition and the shared region. These transactions will be executed more efficiently if there are fewer operations to the shared region because the shared region has higher synchronization overhead and higher data access costs. Therefore, Pasha necessitates a new class of data partitioning algorithms that minimizes operations on data in the shared region while maintaining load balance.

(5) High Concurrency. Concurrency control has been shown to be the performance bottleneck of shared-memory databases when it leverages a large number of CPU cores to process a skewed workload [50]. The abundant CPU resources of a CXL pod prompt us to rethink concurrency control in designing databases based on Pasha: given such high concurrency (e.g., 4608 cores), is it still feasible to resolve conflicts at runtime? We believe one interesting direction is to carefully schedule transactions to avoid conflicts, similar to some transactional memory systems [15, 49], rather than executing transactions immediately as they arrive and resolving conflicts on the fly. The challenge is how to quickly and accurately predict conflicts between queued transactions and running transactions.

(6) Tolerating Partial Failures. Databases on a CXL pod that want high availability need to cope with partial failures: a database process or its associated host fails independently without stopping progress for the database as a whole. The database should make any throughput loss from a partial failure proportional to the resources lost, so if a database is using 32 cores with one thread per core and it loses a core, throughput should drop by roughly $\frac{1}{32}$. The database should also minimize recovery time when the failed thread/process restarts.

One research challenge is that a partial failure may leave in-memory data structures in an illegal state such as when a process acquires a latch, partially modifies the lock table, and fails. The database needs to quickly restore invariants for in-memory data structures and release latches. An additional challenge is that software modules used by the database, like the memory allocator, need to tolerate partial failures to allow the database as a whole to tolerate them. A final challenge is to ensure that transactions processed on non-failed hosts are not impeded by failures on a different host. For example, if a host fails in Pasha, we can prioritize recovering the shared region such that many transactions on the non-failed hosts can proceed without waiting for the recovery of the partition owned by the failed host.

(7) Auto-Scaling. Many cloud databases elastically provision resources to accommodate fluctuating workloads (i.e., a varying incoming transaction rate), thereby reducing costs for customers without compromising performance [3–5]. Traditionally, auto-scaling in distributed databases involves replicating data through networks and adopting a distributed protocol for maintaining transaction semantics, leading to a long data migration time [25].

To support auto-scaling in Pasha, we consider using CXL memory to store migrating data and efficiently synchronizing accesses across hosts to avoid network-based data migration protocols. One research question for Pasha is how to leverage CXL memory to quickly migrate data across partitions while processing transaction operations to the migrating data. Additionally, CXL memory represents a new hardware resource dimension that can scale independently. Scaling different types of resources (i.e., local DRAM, CXL memory, CPUs), while achieving the same transaction processing performance, can result in varied costs for different workloads. One interesting question is how to choose the right type and quantities of resources to scale to address fluctuating workloads while minimizing costs.

4 Preliminary Results

We want to quantitatively estimate the potential performance benefits of Pasha.

Experimental Setup. We emulate a CXL pod on the machine described earlier (§2.2) because there are no commercially available (or hardware prototype) CXL devices that support fine-grained memory sharing with hardware cache coherence. We run 8 virtual machines (VMs) on this machine, each with 4 vCPUs and 8 GB local DRAM, and configure the VMs to share the CXL memory device to emulate a CXL pod that includes 8 machines sharing cache-coherent CXL memory. The cache coherence across VMs is maintained by hardware as the CXL 1.1 memory device is cache-coherent to its connected physical machine. That means that inter-host coherence actions in our testbed are faster than they would be on a CXL pod.

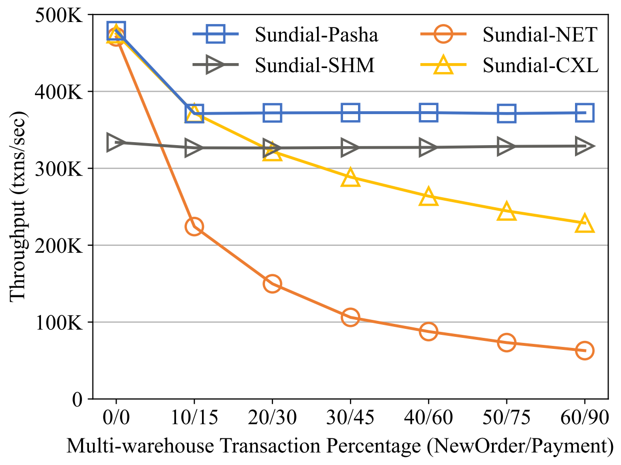

Figure 3: TPC-C throughput with varying percentages of multi-warehouse transactions.

Prototype and Baselines. We compare Pasha with a partition-based distributed architecture and a shared-memory architecture in a distributed in-memory OLTP database, Sundial [51]. Sundial adopts a partitioned-based, share-nothing architecture by default, which we denote *Sundial-NET*. We develop a variant of Sundial, *Sundial-CXL*, which does not use the network and instead leverages message queues implemented in the CXL memory to communicate messages across hosts.

In the shared-memory architecture, all data resides in CXL memory and all hosts synchronize their accesses via hardware cache-coherent shared memory (denoted *Sundial-SHM*). In this configuration, even tables that are only ever accessed by a single host are still stored in shared CXL memory. Finally, we implement Pasha by sharing the data that will be accessed by multiple hosts in CXL memory (i.e., the shared region of Pasha) and partitioning the rest of the data (which resides in local DRAM). We denote this configuration *Sundial-Pasha*. Note that our implementation of Pasha does not support dynamic data movement. Therefore, the data from shared tables is moved to the shared region before starting a test. All approaches use two worker threads for transaction processing.

Workload. We test two stored procedures, `NewOrder` and `Payment`, from the TPC-C benchmark [10] as the other three stored procedures are not supported in Sundial. Our test uses 50% `NewOrder` and 50% `Payment`. For Sundial-NET and Sundial-CXL, all the tables except `ITEM` are partitioned based on the warehouse ID while `ITEM` is replicated at each host because it is read-only and cannot be partitioned, which is the configuration adopted by Sundial [51]. For Sundial-SHM, all tables are stored in CXL memory. For Sundial-Pasha, the database shares the tables to be accessed by multiple hosts in CXL memory while the other tables are either partitioned or replicated (i.e., `ITEM`). We use 8 warehouses, with each warehouse assigned to a specific host.

Results. We measure the overall throughput of TPC-C with varying percentages of multi-warehouse transactions. In the default TPC-C setting, 10% of `NewOrder` and 15% of `Payment` access multiple warehouses (i.e., multi-warehouse transactions), which are multi-host transactions for Sundial-NET and Sundial-CXL in our experiment setup. We vary the two percentages proportionally.

The results in Figure 3 show that when there are no multi-warehouse transactions, Sundial-Pasha achieves the same performance as both Sundial-NET and Sundial-CXL since these approaches perfectly partition the data and store them in local DRAM. Sundial-Pasha has a 1.4× higher throughput than Sundial-SHM (with 0% multi-warehouse transactions) because Sundial-SHM shares all data in the CXL memory and suffers from a higher latency for memory access. As we increase the ratio of multi-warehouse transactions, the throughput of both Sundial-NET and Sundial-CXL drops significantly due to the overhead of inter-host communication of transaction execution and 2PC. For example, with 60% of `NewOrder` and 90% of `Payment` being multi-warehouse transactions, the database needs to send or receive 3.6 network messages, on average, for each transaction.

The throughput of Sundial-Pasha drops because the `CUSTOMER` and `STOCK` tables will be accessed by multiple hosts, so we move them to the shared region, which is stored in CXL memory. Nevertheless, Sundial-Pasha still has 1.1× higher throughput than Sundial-SHM because Sundial-Pasha partitions the other tables. In addition, Sundial-Pasha only migrates 58% of the data generated by TPC-C into shared CXL memory, and subsequently reduces the synchronization overhead across hosts while Sundial-SHM shared all data in all tables. If the database supported dynamic data movement between the shared region and partitions, then the full `CUSTOMER` and `STOCK` tables would not need to be in shared CXL memory. We leave this research challenge (§3.2) for future work. In addition, supporting dynamic data movement will allow the database to store much less data in the shared region and further reduce the overhead of CXL memory access and increase transaction throughput. Sundial-Pasha has up to 5.9× and 1.6× higher throughput than Sundial-NET and Sundial-CXL, respectively, (i.e., the case with 60% of NewOrder and 90% of Payment being multi-warehouse transactions).

5 Related Work

We now place our work in the context of prior research.

Databases over CXL Memory. Many papers consider optimizing databases using CXL memory [11, 13, 28, 38] such as elastically allocating CXL memory [28] or leveraging it for data shuffling [13]. One paper considers mapping an SSD to the memory address space through CXL protocols and adopts hardware-based optimizations (e.g., (de)compression) when databases interact with this memory address space [29]. Two papers discuss the research opportunities of building databases over CXL memory [22, 30], but none of them consider a new database architecture for CXL memory.

Partition-based Distributed Databases. Traditional distributed databases adopt a shared-nothing architecture, where data is partitioned across multiple hosts. Their performance degrades quickly when the number of multi-host transactions increases due to the high cross-host communication cost and 2PC [17, 24, 36, 42]. Many papers optimize this shared-nothing architecture by eliminating or reducing the number of multi-host transactions [18, 34, 37, 40], optimizing concurrency control protocols [26, 35, 51, 55], or eliminating or reducing the cost of 2PC [32, 33, 43]. However, this line of research leverages networks to coordinate and synchronize concurrent accesses and committing transactions, introducing a high overhead compared to a single-node shared-memory database.

Cloud Databases and RDMA-based Databases. The storage-disaggregated database architecture has been adopted by many cloud vendors [12, 19, 44]. Its performance bottleneck is the single primary node for processing read-write transactions. Therefore, several products and papers address this limitation by scaling the primary node to multiple nodes, including Oracle RAC [8], Tauras MM [20], and PolarDB-MP [48], using RDMA networks, similar to research on RDMA-based distributed databases [46, 52, 53, 57]. The common architecture is having each host read/write data stored in the RDMA-connected shared memory to avoid multi-host transactions and optionally cache data in local DRAM to reduce communication cost. Our research, instead, is based on a CXL pod, a new hardware setup with higher performance, and proposes a novel database architecture that prior research did not consider.

6 Conclusion

In this paper, we propose Pasha, a new database architecture for CXL pods, and discuss the research challenges for realizing Pasha. Pasha takes advantage of both partitioned and shared database architectures for scaling databases on the CXL pod. Our preliminary results on a prototype in-memory database show that Pasha has up to 5.9× and 1.4× higher throughput than a partition-based shared-nothing architecture and a shared-memory architecture, respectively.

7 Acknowledgments

Our work is supported in part by PRISM, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We thank Nam Sung Kim and Yan Sun at UIUC for technical consultation and for providing an experimental testbed.

References

[1] 2022. Compute Express Link (CXL) Specification, Revision 3.0, Version 1.0. https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-3.0-Specification.pdf
[2] 2023. Intel Corporation. Intel® memory latency checker v3.10. https://www.intel.com/content/www/us/en/developer/articles/tool/intelrmemory-latency-checker.html Accessed 2024.
[3] 2024. Aurora Serverless. https://aws.amazon.com/rds/aurora/serverless/ (Accessed 2024).
[4] 2024. Azure SQL Serverless. https://learn.microsoft.com/en-us/azure/azure-sql/database/serverless-tier-overview?view=azuresql&tabs=general-purpose (Accessed 2024).
[5] 2024. CockroachDB Serverless. https://www.cockroachlabs.com/blog/announcing-cockroachdb-serverless/ (Accessed 2024).
[6] 2024. Compute Express Link. https://en.wikipedia.org/wiki/Compute_Express_Link Accessed 2024.
[7] 2024. Compute Express Link (CXL) Specification, Revision 3.1. https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-3.1-Specification.pdf Accessed 2024.
[8] 2024. Oracle RAC. https://www.oracle.com/technetwork/database/options/clustering/overview/new-generation-oracle-rac-5975370.pdf (Accessed 2024).
[9] 2024. Sierra Forest. https://en.wikipedia.org/wiki/Sierra_Forest (Accessed 2024).
[10] 2024. TPC Benchmark C. https://www.tpc.org/tpcc/ Accessed 2024.

[11] 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 *International Conference on Management of Data, DaMoN 2022, Philadelphia, PA, USA, 13 June 2022*, Spyros Blanas and Norman May (Eds.). ACM, 8:1–8:5. https://doi.org/10.1145/3533737.3535090

[12] Panagiotis Antonopoulos, Alex Budovski, Cristian Diaconu, Alejandro Hernandez Saenz, Jack Hu, Hanuma Kodavalla, Donald Kossmann, Sandeep Lingam, Umar Farooq Minhas, Naveen Prakash, Vijendra Purohit, Hugh Qu, Chaitanya Sreenivas Ravella, Krystyna Reisteter, Sheetal Shrotri, Dixin Tang, and Vikram Wakade. 2019. Socrates: The New SQL Server in the Cloud. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1743–1756. https://doi.org/10.1145/3299869.3314047

[13] Alexander Baumstark, Marcus Paradies, Kai-Uwe Sattler, Steffen Kläbe, and Stephan Baumann. 2024. So Far and yet so Near - Accelerating Distributed Joins with CXL. In *Proceedings of the 20th International Workshop on Data Management on New Hardware, DaMoN 2024, Santiago, Chile, 10 June 2024*, Carsten Binnig and Nesime Tatbul (Eds.). ACM, 7:1–7:9. https://doi.org/10.1145/3662010.3663449

[14] Daniel S. Berger. 2024. Realistic Expectations for CXL Memory Pools. https://dimes.ws/program/#realistic-expectations-for-cxl-memory-pools

[15] Geoffrey Blake, Ronald G. Dreslinski, and Trevor N. Mudge. 2009. Proactive transaction scheduling for contention management. In *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009), December 12-16, 2009, New York, New York, USA*, David H. Albonesi, Margaret Martonosi, David I. August, and José F. Martínez (Eds.). ACM, 156–167. https://doi.org/10.1145/1669112.1669133

[16] Michael J. Cahill, Uwe Röhm, and Alan D. Fekete. 2008. Serializable isolation for snapshot databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, Jason Tsong-Li Wang (Ed.). ACM, 729–738. https://doi.org/10.1145/1376616.1376690

[17] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. Spanner: Google's Globally-Distributed Database. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, Chandu Thekkath and Amin Vahdat (Eds.). USENIX Association, 251–264. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett

[18] Carlo Curino, Yang Zhang, Evan P. C. Jones, and Samuel Madden. 2010. Schism: a Workload-Driven Approach to Database Replication and Partitioning. *Proc. VLDB Endow.* 3, 1 (2010), 48–57. https://doi.org/10.14778/1920841.1920853

[19] Alex Depoutovitch, Chong Chen, Jin Chen, Paul Larson, Shu Lin, Jack Ng, Wenlin Cui, Qiang Liu, Wei Huang, Yong Xiao, and Yongjun He. 2020. Taurus Database: How to be Fast, Available, and Frugal in the Cloud. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1463–1478. https://doi.org/10.1145/3318464.3386129

[20] Alex Depoutovitch, Chong Chen, Per-Åke Larson, Jack Ng, Shu Lin, Guanzhu Xiong, Paul Lee, Emad Boctor, Samiao Ren, Lengdong Wu, Yuchen Zhang, and Calvin Sun. 2023. Taurus MM: bringing multi-master to the cloud. *Proc. VLDB Endow.* 16, 12 (2023), 3488–3500. https://doi.org/10.14778/3611540.3611542

[21] Peter Xiang Gao, Akshay Narayan, Sagar Karandikar, João Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2016. Network Requirements for Resource Disaggregation. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, Kimberly Keeton and Timothy Roscoe (Eds.). USENIX Association, 249–264. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gao

[22] Yunyan Guo and Guoliang Li. 2024. A CXL-Powered Database System: Opportunities and Challenges. (2024). https://dbgroup.cs.tsinghua.edu.cn/ligl/papers/CXL_ICDE.pdf

[23] Sunita Jain, Nagaradhesh Yeleswarapu, Hasan Al Maruf, and Rita Gupta. 2024. Memory Sharing with CXL: Hardware and Software Design Approaches.

[24] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alex Rasin, Stanley B. Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi. 2008. H-store: a high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.* 1, 2 (2008), 1496–1499. https://doi.org/10.14778/1454159.1454211

[25] Junbin Kang, Le Cai, Feifei Li, Xingxuan Zhou, Wei Cao, Songlu Cai, and Daming Shao. 2022. Remus: Efficient Live Migration for Distributed Databases with Snapshot Isolation. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 2232–2245. https://doi.org/10.1145/3514221.3526047

[26] Ziliang Lai, Hua Fan, Wenchao Zhou, Zhanfeng Ma, Xiang Peng, Feifei Li, and Eric Lo. 2023. Knock Out 2PC with Practicality Intact: a High-performance and General Distributed Transaction Protocol. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2317–2331. https://doi.org/10.1109/ICDE55515.2023.00179

[27] Per-Åke Larson, Spyros Blanas, Cristian Diaconu, Craig Freedman, Jignesh M. Patel, and Mike Zwilling. 2011. High-Performance Concurrency Control Mechanisms for Main-Memory Databases. *Proc. VLDB Endow.* 5, 4 (2011), 298–309. https://doi.org/10.14778/2095686.2095689

[28] 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 *Proceedings of the 19th International Workshop on Data Management on New Hardware, DaMoN 2023, Seattle, WA, USA, June 18-23, 2023*, Norman May and Nesime Tatbul (Eds.). ACM, 35–43. https://doi.org/10.1145/3592980.3595311

[29] Sangjin Lee, Alberto Lerner, Philippe Bonnet, and Philippe Cudré-Mauroux. 2024. Database Kernels: Seamless Integration of Database Systems and Fast Storage via CXL.. In *CIDR*.

[30] Alberto Lerner and Gustavo Alonso. 2024. CXL and the Return of Scale-Up Database Engines. *CoRR* abs/2401.01150 (2024).

[31] Huaicheng Li, Daniel S. Berger, Stanko Novakovic, Lisa Hsu, Dan Ernst, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2022. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. arXiv:2203.00241

[32] Qian Lin, Pengfei Chang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Zhengkui Wang. 2016. Towards a Non-2PC Transaction Management in Distributed Database Systems. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 1659–1674. https://doi.org/10.1145/2882903.2882923

[33] Yi Lu, Xiangyao Yu, Lei Cao, and Samuel Madden. 2021. Epoch-based Commit and Replication in Distributed OLTP Databases. *Proc. VLDB Endow.* 14, 5 (2021), 743–756. https://doi.org/10.14778/3446095.3446098

[34] Yi Lu, Xiangyao Yu, and Samuel Madden. 2019. STAR: Scaling Transactions through Asymmetric Replication. *Proc. VLDB Endow.* 12, 11 (2019), 1316–1329. https://doi.org/10.14778/3342263.3342270

[35] Hatem A. Mahmoud, Vaibhav Arora, Faisal Nawab, Divyakant Agrawal, and Amr El Abbadi. 2014. MaaT: Effective and scalable coordination of distributed transactions in the cloud. *Proc. VLDB Endow.* 7, 5 (2014), 329–340. https://doi.org/10.14778/2732269.2732270

[36] C. Mohan, Bruce G. Lindsay, and Ron Obermarck. 1986. Transaction Management in the R* Distributed Database Management System. *ACM Trans. Database Syst.* 11, 4 (1986), 378–396. https://doi.org/10.1145/7239.7266

[37] Andrew Pavlo, Carlo Curino, and Stanley B. Zdonik. 2012. Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 61–72. https://doi.org/10.1145/2213836.2213844

[38] Niklas Riekenbrauck, Marcel Weisgut, Daniel Lindner, and Tilmann Rabl. 2024. A Three-Tier Buffer Manager Integrating CXL Device Memory for Database Systems. In *40th International Conference on Data Engineering, ICDE 2024 - Workshops, Utrecht, Netherlands, May 13-16, 2024*. IEEE, 395–401. https://doi.org/10.1109/ICDEW61823.2024.00063

[39] Daniel J. Scales and Kourosh Gharachorloo. 1997. Design and Performance of the Shasta Distributed Shared Memory Protocol. In *Proceedings of the 11th international conference on Supercomputing, ICS 1997, Vienna, Austria, July 7-11, 1997*, Steven J. Wallach and Hans P. Zima (Eds.). ACM, 245–252. https://doi.org/10.1145/263580.263643

[40] Marco Serafini, Rebecca Taft, Aaron J. Elmore, Andrew Pavlo, Ashraf Aboulnaga, and Michael Stonebraker. 2016. Clay: Fine-Grained Adaptive Partitioning for General Database Schemas. *Proc. VLDB Endow.* 10, 4 (2016), 445–456. https://doi.org/10.14778/3025111.3025125

[41] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2023. Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 105–121. https://doi.org/10.1145/3613424.3614256

[42] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, Paul Bardea, Amruta Ranade, Ben Darnell, Bram Gruneir, Justin Jaffray, Lucy Zhang, and Peter Mattis. 2020. CockroachDB: The Resilient Geo-Distributed SQL Database. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David

Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1493–1509. https://doi.org/10.1145/3318464.3386134

[43] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 1–12. https://doi.org/10.1145/2213836.2213838

[44] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 1041–1052. https://doi.org/10.1145/3035918.3056101

[45] Tianzheng Wang and Ryan Johnson. 2014. Scalable Logging through Emerging Non-Volatile Memory. *Proc. VLDB Endow.* 7, 10 (2014), 865–876. https://doi.org/10.14778/2732951.2732960

[46] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. 2015. Fast in-memory transaction processing using RDMA and HTM. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*, Ethan L. Miller and Steven Hand (Eds.). ACM, 87–104. https://doi.org/10.1145/2815400.2815419

[47] Emmett Witchel. 2024. Challenges and Opportunities for Systems Using CXL Memory. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 2–2.

[48] Xinjun Yang, Yingqiang Zhang, Hao Chen, Feifei Li, Bo Wang, Jing Fang, Chuan Sun, and Yuhui Wang. 2024. PolarDB-MP: A Multi-Primary Cloud-Native Database via Disaggregated Shared Memory. In *Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago AA, Chile, June 9-15, 2024*, Pablo Barceló, Nayat Sánchez Pi, Alexandra Meliou, and S. Sudarshan (Eds.). ACM, 295–308. https://doi.org/10.1145/3626246.3653377

[49] Richard M. Yoo and Hsien-Hsin S. Lee. 2008. Adaptive transaction scheduling for transactional memory systems. In *SPAA 2008: Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures, Munich, Germany, June 14-16, 2008*, Friedhelm Meyer auf der Heide and Nir Shavit (Eds.). ACM, 169–178. https://doi.org/10.1145/1378533.1378564

[50] Xiangyao Yu, George Bezerra, Andrew Pavlo, Srinivas Devadas, and Michael Stonebraker. 2014. Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores. *Proc. VLDB Endow.* 8, 3 (2014), 209–220. https://doi.org/10.14778/2735508.2735511

[51] Xiangyao Yu, Yu Xia, Andrew Pavlo, Daniel Sánchez, Larry Rudolph, and Srinivas Devadas. 2018. Sundial: Harmonizing Concurrency Control and Caching in a Distributed OLTP Database Management System. *Proc. VLDB Endow.* 11, 10 (2018), 1289–1302. https://doi.org/10.14778/3231751.3231763

[52] Erfan Zamanian, Carsten Binnig, Tim Kraska, and Tim Harris. 2017. The End of a Myth: Distributed Transaction Can Scale. *Proc. VLDB Endow.* 10, 6 (2017), 685–696. https://doi.org/10.14778/3055330.3055335

[53] Ming Zhang, Yu Hua, Pengfei Zuo, and Lurong Liu. 2022. FORD: Fast One-sided RDMA-based Distributed Transactions for Disaggregated Persistent Memory. In *20th USENIX Conference on File and Storage Technologies, FAST 2022, Santa Clara, CA, USA, February 22-24, 2022*, Dean Hildebrand and Donald E. Porter (Eds.). USENIX Association, 51–68. https://www.usenix.org/conference/fast22/presentation/zhang-ming

[54] Wenting Zheng, Stephen Tu, Eddie Kohler, and Barbara Liskov. 2014. Fast Databases with Fast Durability and Recovery Through Multicore Parallelism. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, Jason Flinn and Hank Levy (Eds.). USENIX Association, 465–477. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/zheng_wenting

[55] Xinjing Zhou, Xiangyao Yu, Goetz Graefe, and Michael Stonebraker. 2022. Lotus: Scalable Multi-Partition Transactions on Single-Threaded Partitioned Databases. *Proc. VLDB Endow.* 15, 11 (2022), 2939–2952. https://doi.org/10.14778/3551793.3551843

[56] Zhiting Zhu, Newton Ni, Yibo Huang, Yan Sun, Zhipeng Jia, Nam Sung Kim, and Emmett Witchel. 2024. Lupin: Tolerating Partial Failures in a CXL Pod. In *Proceedings of the 2nd Workshop on Disruptive Memory Systems* (Austin, TX, USA) *(DIMES '24)*. Association for Computing Machinery, New York, NY, USA, 41–50. https://doi.org/10.1145/3698783.3699377

[57] Tobias Ziegler, Philip A. Bernstein, Viktor Leis, and Carsten Binnig. 2023. Is Scalable OLTP in the Cloud a Solved Problem?. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023*. www.cidrdb.org. https://www.cidrdb.org/cidr2023/papers/p50-ziegler.pdf