Disaggregated Database Systems

Jianguo Wang
Purdue University
csjgwang@purdue.edu

Qizhen Zhang
University of Toronto
qz@cs.toronto.edu

ABSTRACT

Disaggregated database systems achieve unprecedented excellence in elasticity and resource utilization at the cloud scale and have gained great momentum from both industry and academia recently. Such systems are developed in response to the emerging trend of disaggregated data centers where resources are physically separated and connected through fast data center networks. Database management systems have been traditionally built based on monolithic architectures, so disaggregation fundamentally challenges the designs. On the other hand, disaggregation offers benefits like independent scaling of compute, memory, and storage. Nonetheless, there is a lack of systematic investigation into new research challenges and opportunities in recent disaggregated database systems.

To provide database researchers and practitioners with insights into different forms of resource disaggregation, we take a snapshot of state-of-the-art disaggregated database systems and related techniques and present an in-depth tutorial. The primary goal is to better understand the enabling techniques and characteristics of resource disaggregation and its implications for next-generation database systems. To that end, we survey recent work on storage disaggregation, which separates secondary storage devices (e.g., SSDs) from compute servers and is widely deployed in current cloud data centers, and memory disaggregation, which further splits compute and memory with Remote Direct Memory Access (RDMA) and is driving the transformation of clouds. In addition, we mention two techniques that bring novel perspectives to the above two paradigms: persistent memory and Compute Express Link (CXL). Finally, we identify several directions that shed light on the future development of disaggregated database systems.

CCS CONCEPTS

• **Information systems** → **DBMS engine architectures**; **Relational parallel and distributed DBMSs**.

KEYWORDS

Databases, Storage Disaggregation, Memory Disaggregation

ACM Reference Format:
Jianguo Wang and Qizhen Zhang. 2023. Disaggregated Database Systems. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3555041.3589403

SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-9507-6/23/06.
https://doi.org/10.1145/3555041.3589403

1 INTRODUCTION

Resource disaggregation has recently emerged as a promising architecture in modern data centers and is evolving the clouds towards *disaggregated data centers* (DDCs) [1, 3, 22, 53]. In DDCs, resources such as compute (e.g., CPUs and GPUs), memory (e.g., DRAM), and storage (e.g., SSDs and NVMs) are no longer aggregated in monolithic servers as in the traditional setting. Rather, they are physically separated from each other, managed in independent resource pools [12, 47], and connected via fast networking, e.g., Remote Direct Memory Access (RDMA). While pools hosting each type of resource necessarily contain some amount of other resources (e.g., low-frequency CPUs in the memory/storage pools that manage local resources and process accesses, or a modest amount of DRAM in the compute pool that caches data), the expectation is that the amount is small. A typical data center application running on top of this architecture will require coordination across disaggregated pools to access resources of different types.

Resource disaggregation can achieve substantial operational advantages for cloud data centers. First, it enables higher resource utilization and less resource fragmentation due to hardware decoupling and pooling. This translates to lower total cost of ownership (TCO) and thus more efficient hardware investment. Second, resource disaggregation provides independent elasticity for scaling compute, memory, and storage, which is becoming increasingly critical in the cloud. Users can request instances and scale them up and down with arbitrary combinations of compute, memory, and storage capacities in response to workload changes without overprovisioning. Existing monolithic servers fail to provide this feature neatly. Independent elasticity also matches the needs of the emerging serverless computing. In addition, resource disaggregation facilitates higher reliability and lowers operational cost because each type of resource can fail and be upgraded without interfering each other. Finally, resource disaggregation provides users the illusion of having a near-infinite pool of resources for applications, which simplifies programming in the cloud. Due to these benefits, disaggregated designs have been widely adopted by cloud providers, e.g., Amazon [40], Microsoft [6, 27], Alibaba [11], Google [16, 29], and IBM [1].

This novel architecture has significant implications on database systems. The first is about mitigating the overhead incurred by disaggregation. In DDCs, what used to be local accesses now become network communication. Despite rapid advances in data center networking in recent years [9, 19, 33], communication remains the performance bottleneck in database systems compared to computation and accesses to memory and storage devices due to various reasons. Hence, database systems need to be aware of the underlying hardware changes and optimize data movement across resource components to reduce potential performance degradation. In particular,

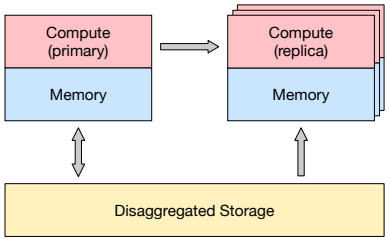

Figure 1: Shared-storage design atop disaggregated storage.

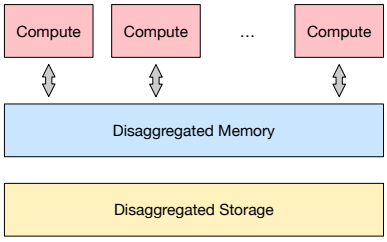

Figure 2: Shared-memory design atop disaggregated memory.

disaggregated database systems usually perform software-level disaggregation to decouple their storage engine—logging and storage—from the compute engine that encompasses SQL layer, caching, and transactions to better align with the underlying hardware-level resource disaggregation [6, 15, 40]. The software-level disaggregation enables a wide range of techniques to minimize the network I/O overhead, e.g., log-as-the-database [6, 15, 40], offloading [48, 55], caching [14, 41], and better exploitation of hardware characteristics [10, 11].

Another implication is about leveraging the benefits enabled by disaggregation. The resource pooling (storage pooling [40, 47] and memory pooling [11, 12]) makes a good case for disaggregated databases to share storage or even share memory. Figure 1 shows the shared-storage architecture adopted in disaggregated-storage database systems, e.g., Amazon Aurora [40], Azure SQL Hyperscale [8], and Google AlloyDB [16]. Figure 2 illustrates an emerging shared-memory design on top of disaggregated memory [44]. Traditional distributed databases (e.g., MySQL Cluster, PostgreSQL Citus [13], Teradata, MemSQL, VoltDB, SQL Server PDW, and Greenplum) have been considering distributed shared-nothing as the "gold standard" [37, 38]. However, in the cloud the shared architectures are more compelling because compute servers now become stateless and thus can achieve better scalability and elasticity, ease of migration, and fast crash recovery.

Scope and overview. In this tutorial we survey disaggregated architectures and systems related to *relational databases* under two disaggregation paradigms: storage disaggregation and memory disaggregation, and we involve both OLTP and OLAP in each category. In addition to secondary storage devices, e.g., SSDs, we also discuss persistent memory (PM) in storage disaggregation. For memory disaggregation, the discussion mainly centers around RDMA-based memory disaggregation, but we also cover CXL. Finally, we conclude this tutorial by discussing several promising future directions on disaggregated databases. Overall, we believe that this is an exciting moment to reflect on the evolution of database system designs with the decoupling of compute, memory, and storage.

Tutorial outline (90 min in total)

(1) **Introduction and motivation (5 min)**.
(2) **Storage disaggregation (30 min)**.
(3) **Additional discussions on PM (10 min)**.
(4) **Memory disaggregation (30 min)**.
(5) **Additional discussions on CXL (10 min)**.
(6) **Future directions (5 min)**.

Target audience and prerequisites. The target audience for this tutorial includes students, researchers, and practitioners who want to understand the enabling techniques of storage and memory disaggregation and learn the challenges and solutions in building disaggregated database systems from both industry and academia. The tutorial is self-contained in providing necessary background on resource disaggregation, so no prior knowledge of disaggregated architectures is required. However, basic knowledge of relational database systems is helpful to understand the materials.

Related tutorials. This tutorial has not been given by the authors in other venues, so SIGMOD 2023 will be the first venue where this tutorial will be offered. Disaggregation, especially that of memory, is an emerging cloud design, and to the best of our knowledge there is no prior tutorial holistically investigating disaggregated databases. Although there have been tutorials and books on cloud databases [25, 26, 31, 32], their main focus was on storage disaggregation. This tutorial significantly expands the horizon of disaggregation with recent materials on memory disaggregation, persistent memory disaggregation, and CXL-based database disaggregation.

2 STORAGE DISAGGREGATION

In this section, we present databases that are optimized for storage disaggregation. Traditional databases are ill-suited for the underlying storage disaggregation to support important features such as elasticity, independent scaling, cost efficiency, and fast recovery. Database systems for storage disaggregation embrace two innovations: (1) **Software-level disaggregation**: Instead of using the legacy monolithic database architecture where all the database components are tightly coupled together, they decouple the storage engine and compute engine to enable independent scaling and support more performance-related optimizations. (2) **Shared-storage**: Instead of sticking to shared-nothing, they adopt the shared-storage architecture to support extreme elasticity and scalability, as well as fast recovery. Figure 1 shows the brief architecture of Aurora [40], which is an exemplary disaggregated database system.

Next, we will present the main ideas of different disaggregated database systems for storage disaggregation (OLTP in Sec. 2.1 and OLAP in Sec. 2.2) that differ in implementation details and the optimizations developed.

2.1 OLTP Databases

Aurora [40] is a popular OLTP database that leverages the underlying storage disaggregation. It separates the storage engine and compute engine. To reduce the expensive network I/O cost, Aurora only sends logs rather than the actual data pages over the network to the storage engine. Aurora redesigns the storage engine in order to generate data pages based on logs asynchronously. The

storage engine is also designed to be highly available where each data segment (of size 10GB) is six-way replicated over three AZs (available zones). To support elasticity and scalability, Aurora can add replica nodes (a.k.a reader nodes) in addition to the primary node (a.k.a writer node). All the compute nodes share the same storage but only the primary node can process write requests (read-write transactions) in order to avoid write conflicts and distributed transactions.

PolarDB [10, 25] has a similar architecture with Aurora that separates the compute layer from the storage layer. But it has different implementations. PolarDB sends both data pages and logs to the storage instead of just sending logs. In order to improve performance, PolarDB leverages emerging hardware such as RDMA and 3D Xpoint SSDs to reduce the network overhead and transaction commit time. The storage layer is also different from Aurora in the sense that PolarDB relies on a lightweight POSIX-like file system called PolarFS to guarantee persistence. PolarFS provides durability through a three-way replication with an optimized Raft protocol. PolarDB is elastic and high-performant but may suffer from high cost because of using advanced hardware.

Socrates [6] also embraces the ideas of disaggregation, shared-storage, and log shipping following Aurora. But Socrates separates the notion of durability and availability by decoupling the database into four tiers: compute tier, log tier (XLOG service), storage tier, and the Azure Storage Service (XStore). The new design allows Socrates to have customized optimizations for durability and availability, e.g., durability does not require copies in fast storage while availability does not require a fixed number of replicas. But it may potentially introduce more data movement since it needs to write pages from the storage tier to XStore over the network.

Taurus [15] takes the ideas from Socrates further with fine-tuned optimizations by observing that the data access patterns for the database logs and database pages are different. Thus, Taurus uses different replication and consistency techniques for logs and pages to obtain high performance, high availability, and low storage cost. However, Taurus uses the writer node to send logs that may add significant burden to the writer node (considering there is only one writer) while Socrates relies on the XLOG service to disseminate logs. To mitigate the overhead, the writer node only guarantees that the updated pages are propagated to one page store (instead of three) and leverages the gossip protocol to achieve consistency among different page stores.

2.2 OLAP Databases

Snowflake [14, 41] is a widely known disaggregated OLAP database system. It is based on the shared-storage (rather than shared-nothing) architecture. It separates storage from compute to allow independent scaling and elasticity. In the storage layer, data is partitioned into large immutable files that are stored in the cloud storage, e.g., S3. In the compute layer, there are many virtual machines (e.g., EC2 instances) organized into Virtual Warehouses (VWs) that can be scaled up and down independently of data storage to support elasticity. The execution engine in the VW is columnar, vectorized, and push-based to achieve high performance. Besides that, there is a cloud service layer that manages VWs and metadata and also performs query optimization and transactions.

AnalyticDB [49] is the disaggregated OLAP database developed by Alibaba that has a similar architecture with Snowflake but it has different optimizations. AnalyticDB uses the hybrid storage format (instead of purely column store) to efficiently support both OLAP queries and point-lookup queries. It also maintains all-column indexes and develops customized optimizations to leverage indexes to support complex ad-hoc queries at the expense of maintaining expensive indexes, while Snowflake only maintains light-weight indexes (e.g., min-max index [30]). Besides that, AnalyticDB decouples reads from writes to allow independent read/write optimizations.

Polaris [4] is a distributed OLAP database that powers Azure Synapse. Similar to Snowflake, Polaris embraces the concepts of storage-compute separation and shared-storage into the design. But the difference is that the compute cluster in Polaris (called Polaris Pool) is responsible for the whole query lifetime (e.g., compilation, optimization, and execution) while the compute cluster in Snowflake only executes the query plan. Also, Polaris introduces the concept of "cell" to abstract heterogeneous data and develops a flexible task orchestration framework and global workload task graph to efficiently execute distributed queries.

Redshift [7] is Amazon's fully managed OLAP database service optimized for storage disaggregation. Unlike Snowflake, AnalyticDB, and Polaris, Redshift initially adopts the shared-nothing MPP-style (massively parallel processing) architecture to achieve fast performance [20]. But recently, it shifts the architectural design to incorporate the disaggregated storage (called Redshift Managed Storage [2, 7]) to allow independent scaling. Redshift scales compute nodes via multi-cluster autoscaling (called Concurrency Scaling). It also introduces many optimizations, e.g., compression, query compilation, offloading, and FPGA accelerations (called AQUA). Besides that, Redshift develops many ML-based optimizations, e.g., automated workload management and physical tuning.

2.3 Persistent Memory Disaggregation

Next, we present database optimizations for persistent memory (PM) disaggregation.

Persistent memory (a.k.a non-volatile main memory) is an emerging memory technology that achieves memory-like speed while guaranteeing persistence. Examples include PCM and Intel Optane PM. Compared with DRAM, PM has higher density and lower cost per GB in addition to persistence. Compared with SSDs, PM is significantly faster and is byte-addressable. There are recent studies advocating PM disaggregation to support elasticity and independent scaling [21, 34, 50]. Another advantage of PM disaggregation is that it can even improve performance [36].

There are unique challenges for PM disaggregation. First, some types of PM require high-end servers to host. For example, Intel Optane PM needs at least 2nd generation Intel Xeon or Platinum CPUs. Thus, different from the fact that a memory or disk server has weak computing capability, a PM server may have much stronger computing capability. As a result, it becomes critical on how to best leverage the remote computing power in the PM servers. Second, it is also non-trivial to guarantee remote persistence in the presence of RDMA. Directly using one-sided RDMA write may not necessarily guarantee remote persistence [21] because data is stored in the CPU cache of the remote PM server.

Kalia et al. present an initial evaluation of accessing remote persistent memory through RDMA [21] based on Intel Optane PM. It discovers many interesting results. For example, it finds that directly writing data to remote PM using one-sided RDMA cannot guarantee persistence because the written data may be in the remote server's NIC or PCIe buffers. It requires one more RDMA read to flush the prior writes. It also compares the one-sided and two-sided approach (i.e., RPC) to guarantee persistence, which turns out that the two-sided approach is even faster.

Oracle Exadata has also deployed PM disaggregation [36]. A straightforward solution to use PM is to directly attach it to a local server. However, it turns out that existing I/O stack introduces significant software overhead that can squander the fast I/O performance provided by PM. That work finds that accessing PM (based on Intel Optane PM) remotely via RDMA can be even faster than accessing PM locally due to the heavy-weight software overhead involved. This is somehow counter-intuitive and it is different from prior memory disaggregation or storage disaggregation.

PilotDB [34] is a recent disaggregated database for PM disaggregation. It builds a disaggregated PM layer to provide a more cost-effective memory layer compared to main memory disaggregation. It stores the log in the PM layer to achieve faster transaction persistence. The main challenge in building PilotDB is the low write bandwidth of PM. To solve this problem, PilotDB applies the idea of log-as-the-database. Two optimizations are introduced to guarantee efficiency and correctness. The first one is compute-node-driven logging in which the log is conducted by the compute node through one-sided RDMA. The other optimization is optimistic page reads, in which the compute node aggressively reads pages via one-sided RDMA. To provide correctness guarantees, the compute node validates a page by log sequence number and replays the log locally if the page is outdated.

3 MEMORY DISAGGREGATION

We now discuss memory disaggregation, a hardware architecture that elevates resource disaggregation to the next level by separating main memory from compute to accomplish the complete compute and data decoupling. Enabling memory disaggregation is the advancement of fast networking technologies. In particular, recent generations of RDMA achieve sub-microsecond latency and hundreds of Gbps throughput. While this is still lower than the performance of buses, the boundary between local and remote machines is becoming blurred.

The first and perhaps the most dramatic impact of memory disaggregation on database systems is about performance. Database systems have been heavily relying on the high speed of random accesses to main memory to achieve good performance. Memory disaggregation means that the majority of these accesses now become network communication. Managing remote memory accesses is thus a key design challenge. The second impact is introduced by sharing remote disaggregated memory between a large number of compute nodes. Data consistency in disaggregated memory must be made available for concurrent accesses from different compute nodes, which also have their own local caches. One-sided RDMA that is often employed for efficient remote memory accesses pushes this responsibility back to the compute nodes. In addition, RDMA fundamentally differs from socket programming and has a large configuration space for tuning. Hence, it is important to expose remote memory with ease of use and high performance. Moreover, since memory disaggregation disables fate-sharing between compute and memory, reliability designs must consider partial failures.

In the rest of this section we present recent investigations on memory disaggregation that cover these aspects. Unlike storage disaggregation, "right" database system designs for memory disaggregation are still under active exploration. This is largely because the architecture of memory disaggregation itself is fast-evolving.

3.1 OLTP Databases

A solution to building database systems on top of memory disaggregation is to run existing ones over transparent disaggregated memory with the aid of the operating system, e.g., LegoOS [35] and Infiniswap [18], which provide an unmodified Linux ABI for applications to utilize disaggregated memory with no modifications. However, recent studies have shown that the performance degradation of this approach offsets the operational benefits of disaggregation and calls for new cloud-native database designs [54, 56]. LegoBase [56] employs two-level designs to manage local and remote memory and handle failures. Specifically, the system utilizes the limited compute-local memory to cache pages from remote memory pool and adopts two LRU lists (one for local cache and the other for remote memory pool) to maximize the cache hit ratios. In addition, it proposes a two-tier ARIES protocol to take checkpoints to remote memory and storage respectively and allow compute nodes to recover from remote memory for fast recovery.

The aforementioned PolarDB system has been extended to support memory disaggregation with its serverless variant. More specifically, PolarDB Serverless [11] inherits the storage-disaggregated architecture of PolarDB but adds a remote memory layer that implements an elastic buffer pool and shares it between all compute nodes. This shared memory pool enables several key benefits for serverless workloads: memory usage is more efficient since compute nodes no longer own private buffers, secondary nodes have the up-to-date view of the data without replaying logs, (re)sizing database instances becomes easy and flexible, and pause/resume and failure recovery are made faster because of compute and memory decoupling. However, coherence and concurrency support, partial failures, and caching must be handled carefully for the same reason.

Another line of work on database designs for memory disaggregation targets indexing, e.g., hash indexing [58], B-tree [43, 57], and LSM [45]. Unlike PolarDB Serverless and LegoBase that only support a single primary database node, these systems support concurrent index updates from multiple compute nodes. They take different approaches to handle write conflicts. RACE hashing [58] proposes a lock-free design to manage concurrent accesses to remote hash index through RDMA compare-and-swap. It also leverages one-sided RDMA to process index requests without involving computation on the memory nodes. Recent work [57] has also adopted a lock-based RDMA approach to extend optimistic-lock-coupling [24] for concurrent accesses to the B-tree index in remote memory. In comparison, Sherman [43] leverages in-order RDMA writes to batch update operations and the SRMA on the RDMA NIC to build two-level lock tables to handle lock contention. It also

optimizes the index layout to reduce write amplification. dLSM [45] is an LSM-based indexing system that adopts a sharding-based approach to supporting concurrent accesses with a suite of performance optimizations, such as software overhead reduction and remote LSM-tree compaction.

Memory disaggregation also has implications on distributed database systems. Recent work has shown that distributed shared-memory database, which did not receive great attention in the past, would be more promising with disaggregated memory [44]. In particular, multiple compute nodes can share the same remote memory pool as in recent database architectures for memory disaggregation. However, many challenges remain. For instance, concurrency control is still challenging due to the lack of hardware-supported cache coherence between compute nodes. It is also non-trivial to build a distributed shared-memory layer that provides high durability and availability with easy-to-use APIs offered to database applications. We must also rethink buffer and index management to adapt the designs to leverage memory disaggregation because existing local memory- or disk-oriented mechanisms are ill-suited for this new setup [44].

Memory disaggregation has influenced the designs for other workloads and architectures as well. FlexChain [46] separates the world state in permissioned blockchain systems with a tiered key-value store based on disaggregated memory to overcome imbalanced consumption of compute and memory resources across blockchain workloads. To optimize the validation phase in XOV blockchains that becomes the new bottleneck in the disaggregated architecture, FlexChain adopts a dependency-graph-based approach that parallelizes validations.

3.2 OLAP Databases

Memory disaggregation has more profound performance implications on OLAP workloads [23, 29, 53–55]. A visionary work [53] first identified the challenges and opportunities of memory disaggregation for OLAP database systems. It discussed the potential benefits of disaggregated memory but also explained how significant performance degradation can happen due to excessive network communication when large working sets reside in remote memory. The authors then took a further step to evaluate the impact of memory disaggregation on production database systems with two representative systems (PostgreSQL, a disk-based DBMS, and MonetDB, an in-memory DBMS) using the TPC-H benchmark [54]. Through extensive experiments, the study shows that both the benefits and overhead of memory disaggregation are substantial. On one hand, a large disaggregated memory pool can prevent the processing of memory-intensive queries from being spilled to secondary storage. On the other hand, network communications for remote memory accesses are confirmed to be expensive for large queries. In addition, this work provides insights about how to design memory management in DBMSs for a disaggregated environment. If the memory is managed by the application, as is the case with MonetDB, some of the data can be kept in local memory. In comparison, if memory management is delegated to the OS, as is the case with PostgreSQL, the disk cache may be kept in the remote memory pool where most memory resides. This design hurts performance as cached data would still need to be moved across the network.

To overcome the overhead of memory disaggregation and unlock its full benefits, TELEPORT [55] is introduced as a new feature for optimizing memory-hungry analytics query processing performance. Specifically, TELEPORT is a compute pushdown framework that enables DBMSs to offload expensive operations close to data. It is based on disaggregated operating systems that emulate traditional OS interfaces to provide backward compatibility such that current applications can immediately harvest the benefit of resource disaggregation. With TELEPORT, DBMSs are capable of executing light-weight but memory-intensive operators on the remote memory side. In doing so they eliminate expensive data movement cost and hence achieve better query processing performance. TELEPORT is unique in its generality and efficiency. With a new system call, it allows applications to offload arbitrary pieces of computation by wrapping them as functions. Pushing a function down is as simple as providing the pointers of the function and its arguments to the memory pool. This is possible because applications' stack, heap, and code pages all live in the memory pool as a byproduct of disaggregated OSes. Data synchronization is critical for TELEPORT: the compute pool caches part of the main memory, so data copies in different pools can diverge before, during, and after pushdown. Without proper synchronization, concurrent threads in two pools may access the same memory pages without observing each other's updates. TELEPORT employs efficient synchronization mechanisms to provide memory coherence. It only synchronizes data on applications' demands, which outperforms application-agnostic alternatives. TELEPORT shows the necessity of providing compute offloading capabilities in disaggregated memory to optimize memory-intensive analytical query processing.

Farview [23] is a memory disaggregation framework that supports operator offloading using FPGAs. It consists of three components: a network stack, a memory stack, and an operator stack in between. The network stack is responsible for accepting requests from the query processing engine in the compute pool with reliable RDMA connections, and the memory stack performs address translation and serves memory accesses with local physical memory or pages/blocks from the storage. The operator stack, the core piece of Farview, implements database operators (e.g., projection, selection, groupby, and aggregation) by configuring the memory-attached FPGA to process the read data from the memory stack. In particular, Farview supports pipelining in the operator stack such that DBMSs can offload complex analytical sub-queries to the disaggregated memory to minimize data movement.

Redy [51] is a new cloud service that uses stranded memory as remote caches. It offers a lower-latency alternative to SSDs, using disaggregated memory resources that would otherwise go to waste Its use of RDMA for caching leads to two challenges. The first is performance. Tuning RDMA requires complex, low-level optimizations to trade off network latency, throughput, and resource cost. Second, stranded memory resources are highly dynamic. Their availability can be as short as a few minutes. Redy addresses the first challenge with SLO-based RDMA configuration and the dynamic challenge with a memory manager that migrates a cache to new stranded memory when the old memory is reclaimed by the cloud VM allocator. To further reduce cost in remote caching, CompuCache [52] hosts cache servers with spot instances. To reduce data movement overhead, it supports near-data processing with stored procedures

that achieve high performance by single-round-trip offloading with server-side pointer-chasing.

In addition, Dremel [29] leverages disaggregated memory to decouple compute and intermediate shuffle state for distributed joins. The tight coupling between these two was becoming a scalability bottleneck in Dremel because (1) shuffles scale quadratically with the number of producers and consumers, and (2) the coupling introduces resource fragmentation and stranding, exacerbating resource underutilization in cloud data centers. To address these issues, Dremel adopts a disaggregated intermediate shuffle layer that improves the performance and scalability of joins by an order of magnitude. It also significantly reduces the cost of the service.

3.3 CXL for Disaggregation

So far we have been assuming that memory and compute servers are connected via RDMA. However, there is still a noticeable performance gap between local memory and RDMA-accessible remote memory. Compute Express Link (CXL) [39] recently emerged as a promising interconnect technology that is faster than RDMA and thus achieves closer performance to that of local memory. CXL is an open industry standard that provides fast interconnections between host processors, accelerators, and I/O devices with cache coherence. Recent studies show that CXL can be adopted for efficient resource disaggregation [5, 17, 28].

Ahn et al. [5] investigated the impact of CXL on main memory databases, specifically Type 3 CXL devices using the CXL.memory protocol for memory expansion. Similar to persistent memory, there are two approaches to utilizing CXL memory devices in database systems. First, CXL-connected memory can be treated the same as host local memory with a unified memory space. This approach requires no application modifications. However, it incurs performance degradation because of the performance difference between CXL and local memory. The other approach is to distinguish CXL memory from local memory such that database systems can explicitly manage data between the two tiers to tune performance. The authors took the latter, i.e., using local memory for delta storage and operational data and CXL memory for main storage in SAP HANA. They evaluated the performance drop due to CXL-based memory disaggregation using TPC-C and TPC-DS. The preliminary results show that there is virtually no performance drop on TPC-C due to prefetching, but there is 7% to 27% performance drop on TPC-DS. More sophisticated evaluation and analysis are left to future work.

DirectCXL [17] is a disaggregated memory solution based on CXL and FPGA. It provides a transparent programming environment that unifies local and CXL memory without changing applications. Compared to RDMA, it improves the raw latency by 6.2× and the performance of real applications by 3×. Pond [27] is a cloud memory pooling framework using CXL. There are two key insights behind its designs: (1) pooling memory across a small number of sockets suffices to improve memory utilization, and (2) machine learning models can predict how to allocate local and remote memory to VMs to minimize performance disruption. To reduce pooling overhead, the framework places the memory of workloads that are less latency-sensitive and memory that is likely untouched by VMs in the remote CXL memory pool.

4 FUTURE DIRECTIONS

Comprehensive performance evaluation of disaggregated databases. Given different hardware platforms (e.g., RDMA and CXL), different workloads (e.g., OLTP, OLAP, and HTAP), and different forms of disaggregation (e.g., memory and storage disaggregation), having an experimental platform that integrates all these ingredients and supports extensive performance evaluation of database systems on resource disaggregation would be helpful for future investigations on disaggregated database designs.

Scalable transactions in disaggregated databases. Existing cloud databases usually have a single compute node that processes write workloads, with other nodes only processing read-only queries. It is interesting to support multiple writers to improve write scalability, which would be more feasible with memory disaggregation and distributed shared-memory databases.

Automatic resource provisioning. As the clouds are becoming increasingly disaggregated, it is critical to investigate automatic resource provisioning to decide the right amount of resources for different applications. Recent advances in machine learning techniques can be leveraged for this problem.

CXL-optimized databases. CXL is projected to be widely available in 2023, and different generations of CXL have different properties. However, it remains unclear how this new interconnect will change the architectures of different disaggregated database systems.

5 BIOGRAPHY

Jianguo Wang is a tenure-track assistant professor in the Department of Computer Science at Purdue University. Prior to that, he was a full-time software engineer working on Amazon Aurora [40], a cloud-native database system optimized for storage disaggregation. He received his Ph.D. degree in Computer Science from the University of California, San Diego. His research interests include building novel database systems for the cloud, new hardware, and emerging applications. He is a co-author of [44, 45] that are highly relevant to disaggregated database systems. He offered a graduate seminar course (CS592) on "Disaggregated Database Systems" in Spring 2023 at Purdue University [42].

Qizhen Zhang is a tenure-track assistant professor in the Department of Computer Science at the University of Toronto starting August 2023. Before that, he is spending a year in industry at Microsoft Research, Redmond. He received his Ph.D. degree in Computer and Information Science at the University of Pennsylvania. His research has been bridging cloud data processing systems and data center networks to address emerging challenges in hyperscale data processing. He is broadly interested in data management and computer systems and networking, and he researches across the data processing stack. His recent work [51, 53–55] has been pioneering data processing in disaggregated data centers.

REFERENCES

[1] Advancing Cloud with Memory Disaggregation, https://www.ibm.com/blogs/research/2018/01/advancing-cloud-memory-disaggregation/.
[2] Amazon Redshift RA3 Instances with Managed Storage, https://aws.amazon.com/redshift/features/ra3/.
[3] Intel RSD, https://www.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design-overview.html.

[4] J. Aguilar-Saborit and R. Ramakrishnan. POLARIS: The Distributed SQL Engine in Azure Synapse. *Proceedings of the VLDB Endowment (PVLDB)*, 13(12):3204–3216, 2020.

[5] M. Ahn, A. Chang, D. Lee, J. Gim, J. Kim, J. Jung, O. Rebholz, V. Pham, K. T. Malladi, and Y. Ki. Enabling CXL Memory Expansion for In-Memory Database Management Systems. In *International Conference on Management of Data (DaMoN)*, pages 8:1–8:5, 2022.

[6] P. Antonopoulos, A. Budovski, C. Diaconu, A. H. Saenz, J. Hu, H. Kodavalla, D. Kossmann, S. Lingam, U. F. Minhas, N. Prakash, V. Purohit, H. Qu, C. S. Ravella, K. Reisteter, S. Shrotri, D. Tang, and V. Wakade. Socrates: The New SQL Server in the Cloud. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 1743–1756, 2019.

[7] N. Armenatzoglou, S. Basu, N. Bhanoori, M. Cai, N. Chainani, K. Chinta, V. Govindaraju, T. J. Green, M. Gupta, S. Hillig, E. Hotinger, Y. Leshinksy, J. Liang, M. McCreedy, F. Nagel, I. Pandis, P. Parchas, R. Pathak, O. Polychroniou, F. Rahman, G. Saxena, G. Soundararajan, S. Subramanian, and D. Terry. Amazon Redshift Re-invented. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 2205–2217, 2022.

[8] Azure SQL Database. Hyperscale service tier. https://learn.microsoft.com/en-us/azure/azure-sql/database/service-tier-hyperscale?view=azuresql, 2023.

[9] W. Bai, S. S. Abdeen, A. Agrawal, K. K. Attre, P. Bahl, A. Bhagat, G. Bhaskara, T. Brokhman, L. Cao, A. Cheema, R. Chow, J. Cohen, M. Elhaddad, V. Ette, I. Figlin, D. Firestone, M. George, I. German, L. Ghai, E. Green, A. Greenberg, M. Gupta, R. Haagens, M. Hendel, R. Howlader, N. John, J. Johnstone, T. Jolly, G. Kramer, D. Kruse, A. Kumar, E. Lan, I. Lee, A. Levy, M. Lipshteyn, X. Liu, C. Liu, G. Lu, Y. Lu, X. Lu, V. Makhervaks, U. Malashanka, D. A. Maltz, I. Marinos, R. Mehta, S. Murthi, A. Namdhari, A. Ogus, J. Padhye, M. Pandya, D. Phillips, A. Power, S. Puri, S. Raindel, J. Rhee, A. Russo, M. Sah, A. Sheriff, C. Sparacino, A. Srivastava, W. Sun, N. Swanson, F. Tian, L. Tomczyk, V. Vadlamuri, A. Wolman, Y. Xie, J. Yom, L. Yuan, Y. Zhang, and B. Zill. Empowering Azure Storage with RDMA. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

[10] W. Cao, Z. Liu, P. Wang, S. Chen, C. Zhu, S. Zheng, Y. Wang, and G. Ma. PolarFS: An Ultra-Low Latency and Failure Resilient Distributed File System for Shared Storage Cloud Database. *Proceedings of the VLDB Endowment (PVLDB)*, 11(12):1849–1862, 2018.

[11] W. Cao, Y. Zhang, X. Yang, F. Li, S. Wang, Q. Hu, X. Cheng, Z. Chen, Z. Liu, J. Fang, B. Wang, Y. Wang, H. Sun, Z. Yang, Z. Cheng, S. Chen, J. Wu, W. Hu, J. Zhao, Y. Gao, S. Cai, Y. Zhang, and J. Tong. PolarDB Serverless: A Cloud Native Database for Disaggregated Data Centers. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 2477–2489, 2021.

[12] H. Craddock, L. P. Konudula, K. Cheng, and G. Kul. The Case for Physical Memory Pools: A Vision Paper. In *International Conference on Cloud Computing (CLOUD)*, pages 208–221, 2019.

[13] U. Cubukcu, O. Erdogan, S. Pathak, S. Sannakkayala, and M. Slot. Citus: Distributed PostgreSQL for Data-Intensive Applications. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 2490–2502, 2021.

[14] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, A. W. Lee, A. Motivala, A. Q. Munir, S. Pelley, P. Povinec, G. Rahn, S. Triantafyllis, and P. Unterbrunner. The Snowflake Elastic Data Warehouse. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 215–226, 2016.

[15] A. Depoutovitch, C. Chen, J. Chen, P. Larson, S. Lin, J. Ng, W. Cui, Q. Liu, W. Huang, Y. Xiao, and Y. He. Taurus Database: How to be Fast, Available, and Frugal in the Cloud. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 1463–1478, 2020.

[16] Google Cloud. AlloyDB for PostgreSQL Under the Hood: Intelligent, Database-aware Storage. https://cloud.google.com/blog/products/databases/alloydb-for-postgresql-intelligent-scalable-storage, 2022.

[17] D. Gouk, S. Lee, M. Kwon, and M. Jung. Direct Access, High-Performance Memory Disaggregation with DirectCXL. In *USENIX Annual Technical Conference (ATC)*, pages 287–294, 2022.

[18] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin. Efficient Memory Disaggregation with Infiniswap. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 649–667, 2017.

[19] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn. RDMA over Commodity Ethernet at Scale. In *Proceedings of the ACM Conference on Data Communication (SIGCOMM)*, pages 202–215, 2016.

[20] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, and V. Srinivasan. Amazon Redshift and the Case for Simpler Data Warehouses. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 1917–1923, 2015.

[21] A. Kalia, D. G. Andersen, and M. Kaminsky. Challenges and Solutions for Fast Remote Persistent Memory Access. In *ACM Symposium on Cloud Computing (SoCC)*, pages 105–119, 2020.

[22] K. Keeton. Memory-Driven Computing. https://www.usenix.org/sites/default/files/conference/protected-files/fast17_slides_keeton.pdf. In *USENIX Conference on File and Storage Technologies (FAST)*, 2017.

[23] D. Korolija, D. Koutsoukos, K. Keeton, K. Taranov, D. S. Milojicic, and G. Alonso. Farview: Disaggregated Memory with Operator Off-loading for Database Engines. In *Conference on Innovative Data Systems Research (CIDR)*, 2022.

[24] V. Leis, F. Scheibner, A. Kemper, and T. Neumann. The ART of Practical Synchronization. In *Proceedings of the International Workshop on Data Management on New Hardware (DaMoN)*, pages 3:1–3:8, 2016.

[25] F. Li. Cloud Native Database Systems at Alibaba: Opportunities and Challenges. *Proceedings of the VLDB Endowment (PVLDB)*, 12(12):2263–2272, 2019.

[26] G. Li, H. Dong, and C. Zhang. Cloud Databases: New Techniques, Challenges, and Opportunities. *Proceedings of the VLDB Endowment (PVLDB)*, 15(12):3758–3761, 2022.

[27] H. Li, M. Hao, S. Novakovic, V. Gogte, S. Govindan, D. R. K. Ports, I. Zhang, R. Bianchini, H. S. Gunawi, and A. Badam. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023.

[28] P. Mehra and T. Coughlin. Taming Memory With Disaggregation. *Computer*, 55(9):94–98, 2022.

[29] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, H. Ahmadi, D. Delorey, S. Min, M. Pasumansky, and J. Shute. Dremel: A Decade of Interactive SQL Analysis at Web Scale. *Proceedings of the VLDB Endowment (PVLDB)*, 13(12):3461–3472, 2020.

[30] G. Moerkotte. Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing. In *Proceedings of International Conference on Very Large Data Bases (VLDB)*, pages 476–487, 1998.

[31] V. R. Narasayya and S. Chaudhuri. Cloud Data Services: Workloads, Architectures and Multi-Tenancy. *Foundations and Trends in Databases*, 10(1):1–107, 2021.

[32] V. R. Narasayya and S. Chaudhuri. Multi-Tenant Cloud Data Services: State-of-the-Art, Challenges and Opportunities. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 2465–2473, 2022.

[33] L. Poutievski, O. Mashayekhi, J. Ong, A. Singh, M. M. B. Tariq, R. Wang, J. Zhang, V. Beauregard, P. Conner, S. D. Gribble, R. Kapoor, S. Kratzer, N. Li, H. Liu, K. Nagaraj, J. Ornstein, S. Sawhney, R. Urata, L. Vicisano, K. Yasumura, S. Zhang, J. Zhou, and A. Vahdat. Jupiter Evolving: Transforming Google's Datacenter Network via Optical Circuit Switches and Software-Defined Networking. In *Proceedings of the ACM Conference on Data Communication (SIGCOMM)*, pages 66–85, 2022.

[34] C. Ruan, Y. Zhang, C. Bi, X. Ma, H. Chen, F. Li, X. Yang, C. Li, A. Aboulnaga, and Y. Xu. Persistent Memory Disaggregation for Cloud-Native Relational Databases. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 498–512, 2023.

[35] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 69–87, 2018.

[36] J. Shi. Exadata with Persistent Memory: An Epic Journey, https://www.snia.org/educational-library/exadata-persistent-memoryan-epic-journey-2020, 2020.

[37] M. Stonebraker. The Case for Shared Nothing. *IEEE Data Engineering Bulletin*, 9(1):4–9, 1986.

[38] M. Stonebraker. Shared-nothing vs Shared-disk, https://www.youtube.com/watch?v=G-o2bFd91Sw. In *Extremely Large Databases Workshop (XLDB)*, 2011.

[39] The CXL Consortium. Compute Express Link: The Breakthrough CPU-to-Device Interconnect. https://www.computeexpresslink.org, 2022.

[40] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 1041–1052, 2017.

[41] M. Vuppalapati, J. Miron, R. Agarwal, D. Truong, A. Motivala, and T. Cruanes. Building An Elastic Query Engine on Disaggregated Storage. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 449–462, 2020.

[42] J. Wang. Purdue CS592: Disaggregated Database Systems (Spring 2023), https://www.cs.purdue.edu/homes/csjgwang/CS592DisaggregatedDB/, 2023.

[43] Q. Wang, Y. Lu, and J. Shu. Sherman: A Write-Optimized Distributed B+Tree Index on Disaggregated Memory. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 1033–1048, 2022.

[44] R. Wang, J. Wang, S. Idreos, M. Özsu, and W. Aref. The Case for Distributed Shared-Memory Databases with RDMA-Enabled Memory Disaggregation. *Proceedings of the VLDB Endowment (PVLDB)*, 16(1):15–22, 2022.

[45] R. Wang, J. Wang, P. Kadam, M. T. Özsu, and W. G. Aref. dLSM: An LSM-Based Index for Memory Disaggregation. In *IEEE International Conference on Data Engineering (ICDE)*, 2023.

[46] C. Wu, M. J. Amiri, J. Asch, H. Nagda, Q. Zhang, and B. T. Loo. FlexChain: An Elastic Disaggregated Blockchain. *Proceedings of the VLDB Endowment (PVLDB)*, 16(1):23–36, 2022.

[47] S. Xue, S. Zhao, Q. Chen, G. Deng, Z. Liu, J. Zhang, Z. Song, T. Ma, Y. Yang, Y. Zhou, K. Niu, S. Sun, and M. Guo. Spool: Reliable Virtualized NVMe Storage Pool in Public Cloud Infrastructure. In *USENIX Annual Technical Conference*

(ATC), pages 97–110, 2020.

[48] Y. Yang, M. Youill, M. E. Woicik, Y. Liu, X. Yu, M. Serafini, A. Aboulnaga, and M. Stonebraker. FlexPushdownDB: Hybrid Pushdown and Caching in a Cloud DBMS. *Proceedings of the VLDB Endowment (PVLDB)*, 14(11):2101–2113, 2021.

[49] C. Zhan, M. Su, C. Wei, X. Peng, L. Lin, S. Wang, Z. Chen, F. Li, Y. Pan, F. Zheng, and C. Chai. AnalyticDB: Real-time OLAP Database System at Alibaba Cloud. *Proceedings of the VLDB Endowment (PVLDB)*, 12(12):2059–2070, 2019.

[50] M. Zhang, Y. Hua, P. Zuo, and L. Liu. FORD: Fast One-sided RDMA-based Distributed Transactions for Disaggregated Persistent Memory. In *USENIX Conference on File and Storage Technologies (FAST)*, pages 51–68, 2022.

[51] Q. Zhang, P. A. Bernstein, D. S. Berger, and B. Chandramouli. Redy: Remote Dynamic Memory Cache. *Proceedings of the VLDB Endowment (PVLDB)*, 15(4):766 – 779, 2022.

[52] Q. Zhang, P. A. Bernstein, D. S. Berger, B. Chandramouli, V. Liu, and B. T. Loo. CompuCache: Remote Computable Caching using Spot VMs. In *Conference on Innovative Data Systems Research (CIDR)*, 2022.

[53] Q. Zhang, Y. Cai, S. Angel, V. Liu, A. Chen, and B. T. Loo. Rethinking Data Management Systems for Disaggregated Data Centers. In *Conference on Innovative Data Systems Research (CIDR)*, 2020.

[54] Q. Zhang, Y. Cai, X. Chen, S. Angel, A. Chen, V. Liu, and B. T. Loo. Understanding the Effect of Data Center Resource Disaggregation on Production DBMSs. *Proceedings of the VLDB Endowment (PVLDB)*, 13(9):1568–1581, 2020.

[55] Q. Zhang, X. Chen, S. Sankhe, Z. Zheng, K. Zhong, S. Angel, A. Chen, V. Liu, and B. T. Loo. Optimizing Data-intensive Systems in Disaggregated Data Centers with TELEPORT. In *ACM International Conference on Management of Data (SIGMOD)*, pages 1345–1359, 2022.

[56] Y. Zhang, C. Ruan, C. Li, J. Yang, W. Cao, F. Li, B. Wang, J. Fang, Y. Wang, J. Huo, and C. Bi. Towards Cost-Effective and Elastic Cloud Database Deployment via Memory Disaggregation. *Proceedings of the VLDB Endowment (PVLDB)*, 14(10):1900–1912, 2021.

[57] T. Ziegler, S. Tumkur Vani, C. Binnig, R. Fonseca, and T. Kraska. Designing Distributed Tree-based Index Structures for Fast RDMA-capable Networks. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 741–758, 2019.

[58] P. Zuo, J. Sun, L. Yang, S. Zhang, and Y. Hua. One-sided RDMA-Conscious Extendible Hashing for Disaggregated Memory. In *USENIX Annual Technical Conference (ATC)*, pages 15–29, 2021.