Datenbank-Spektrum
https://doi.org/10.1007/s13222-026-00559-3

FACHBEITRAG

Toward Disaggregated Analytical Database Systems in the AI Hardware Era

Jigao Luo[1] · **Nils Boeschen**[1,4] · **Muhammad El-Hindi**[2] · **Carsten Binnig**[1,3,4]

Received: 9 June 2026 / Accepted: 7 August 2026

Abstract

The AI hardware boom has driven modern data centers toward HPC-style architectures centered on GPU clusters, RDMA-capable networks, and high-throughput NVMe storage. While designed primarily for training and inference, this infrastructure also creates new opportunities for designing the next generation of scalable database systems for analytical workloads on top of such data centers. In particular, the combination of GPU-centric computation, fast networking, and fast storage enables disaggregated architectures that extend beyond single-node, GPU-memory-resident execution.

This paper discusses the challenges and design considerations of analytical query processing on such disaggregated GPU-centric systems. We examine how modern networks and storage enable distributed execution and out-of-memory processing, and how their interaction shapes end-to-end performance. Our recent results show that naïve use of existing I/O abstractions can underutilize both compute and I/O bandwidth due to insufficient overlap between computation and data movement. We therefore identify effective I/O-computation overlap as a key requirement for fully exploiting the AI data center stack, and outline future research directions for next-generation analytical database architectures.

1 Introduction

Data Center Hardware in an AI Era The recent AI boom has triggered substantial investment in data centers for large-scale distributed training and inference [1, 2]. These AI data centers increasingly follow HPC-style designs, consisting of large GPU clusters connected via high-bandwidth, low-latency RDMA-capable networks and backed by high-throughput disaggregated NVMe storage systems [3, 4]. The scale of these investments is expected to increase the availability and affordability of earlier-generation GPU clusters for workloads beyond large-scale AI model training and inference. Although no longer considered frontier hardware, these clusters can still substantially accelerate data-processing workloads. Indeed, cloud providers such as Microsoft are investing in scalable database systems designed natively for distributed GPU processing on this emerging hardware ecosystem [5]. Our research group has been building disaggregated database systems for such data centers, which form the focus of this paper.

Fast Networks Support Distributed GPU Execution Early GPU-accelerated database systems for analytical OLAP-style processing primarily targeted single-node deployments and assumed that table data resides entirely in GPU memory. This assumption no longer holds for modern OLAP workloads, whose working sets frequently exceed the memory capacity of a single GPU. Recent work has therefore explored low-latency, high-bandwidth networks, including InfiniBand and 400G Ethernet, to enable distributed GPU query processing and to expand the effective memory capacity available to GPU workloads [5, 6].

Fast Storage Enables Large-Scale Query Processing However, distributed GPU query processing remains constrained by the assumption that the active workload fits into the aggregated GPU memory. Modern AI infrastructure relaxes this constraint through high-throughput NVMe storage attached to GPUs locally and remotely. Together with technologies such as GPUDirect Storage [7], GPUs can access storage with minimal CPU intervention. These capabilities

✉ Jigao Luo
jigao.luo@tu-darmstadt.de

1 Systems Group, Technical University of Darmstadt, Darmstadt, Germany

2 Technical University of Munich, Munich, Germany

3 hessian.AI, Darmstadt, Germany

4 DFKI, German Research Center for Artificial Intelligence, Darmstadt, Germany

Published online: 07 September 2026

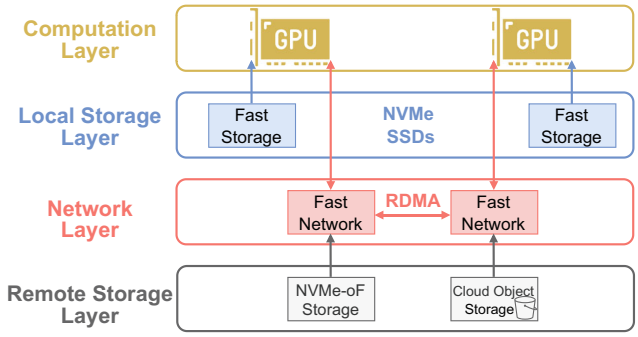

Fig. 1 A disaggregated architecture for GPU-centric data processing. The architecture leverages fast networking and storage hardware in modern AI data centers and spans four layers: computation, local storage, network, and remote storage

enable GPU-centric query engines to support out-of-memory execution by efficiently streaming data between storage and GPU memory [8, 9].

Toward Disaggregated GPU-Centric Data Processing Beyond exploiting modern storage and networking technologies in isolation, we argue that the full AI data center stack enables a disaggregated architecture for GPU-centric data processing, as illustrated in Fig. 1. Unlike traditional CPU-GPU co-processing approaches, this architecture treats the GPU as the primary execution engine in the *computation layer*. Through mechanisms such as GPUDirect, GPUs can directly interact with both the *storage layer* and the *network layer*. These layers provide two complementary forms of disaggregation: RDMA-capable networks enable inter-node communication between GPUs, while high-bandwidth NVMe SSDs enable efficient out-of-core execution within each node. Combining storage and networking further introduces a fourth layer, the *remote storage layer*, where shared data is accessed over the network by distributed compute nodes. For instance, NVMe over Fabrics (NVMe-oF) provides access to remote storage over RDMA-capable networks, analogous to RDMA access to remote memory.

Challenges for GPU-Centric Disaggregation Despite these hardware advances, fully exploiting GPU-centric systems atop such architectures remains challenging. Low-level GPU programming primitives for storage and network I/O are difficult to use directly, while tensor computation runtimes (TCRs) such as PyTorch [10] typically expose a blocking execution model in which data are loaded in bulk before computation begins. Although this model is well-suited to compute-intensive iterative training workloads, it is inefficient for I/O-intensive OLAP workloads, where bulk loading can reduce throughput and lead to out-of-memory (OOM) errors [11]. Remote storage further introduces additional challenges. While NVMe-oF exposes remote storage over RDMA-capable networks, its integration into GPU-centric query processing remains largely unexplored. Moreover, CPU-mediated access paths can reintroduce bottlenecks and diminish the computation benefits of GPUs.

Contributions of this Paper This paper provides an overview of our previously published results in the area of building such disaggregated GPU data processing and synthesizes them into a unified system-level vision. Rather than introducing a single new system, we connect these individual contributions to explain the overall architecture and design principles that underpin our work in this space. In addition, we discuss future research directions we believe are critical to fully realize GPU-centric, disaggregated data processing in modern AI data centers. Our central observation is that overlapping I/O with GPU computation is essential to saturate modern storage and network hardware and to enable efficient distributed query processing. The next section introduces the layers of our disaggregated architecture. We then discuss techniques for efficient query processing within individual layers and across layer boundaries. Finally, we outline promising directions for future work in fully disaggregated GPU-centric data processing.

2 Architecture Overview

The main idea of the proposed architecture, shown in Fig. 1, is to view modern GPU data centers as disaggregated data processing systems: computation is performed primarily on GPUs, while data resides in storage layers at varying distances from the compute layer and is accessed via a high-performance network layer. In this section, we outline how this architecture can be translated into a disaggregated database system for analytical query processing.

Computation Layer With the rise of AI, both GPU compute capabilities and efforts to leverage them for data systems have increased further. GPU-accelerated analytical processing has been extensively studied for in-memory OLAP workloads, as GPU parallelism matches the data-parallel nature of analytical processing. Recent work [12] even uses AI software stacks, i.e., tensor computation runtimes (TCRs) such as PyTorch [10], for general-purpose analytical query processing on AI hardware. However, simply moving operators to GPUs is not enough: larger working sets cannot be kept fully in GPU memory and require efficient access through storage or network layers. Our prior work shows how to make GPU query processing effective by tightly integrating computation with local storage, remote storage, and fast networks [8, 11, 13–17].

Local Storage Layer NVMe SSDs play a key role in AI-centric data centers due to their high bandwidth and favorable cost per capacity compared to main memory. This layer is complemented by GPUDirect Storage (GDS), which transfers data directly between SSDs and GPU memory while bypassing intermediate copies in CPU memory. As such, local storage provides an effective mechanism to overcome limited GPU memory and enable out-of-memory execution. In our previous work [8, 11, 15], we show how to efficiently integrate direct access to local storage into GPU-centric query processing, proving that compute-intensive heavyweight compression and data pruning techniques are highly beneficial for such systems, allowing them to inflate the effective data loading speeds past the raw read bandwidth of SSDs. We also show that these advantages can be achieved by optimizing existing file formats, such as Parquet [18], leading to robust high access speeds for data on the local storage layer.

Network Layer The network layer leverages high-speed RDMA-capable networks in modern AI data centers to support large-scale data movement during distributed query execution. It enables partitioning and coordination of work across GPU nodes, thereby increasing the effective aggregate memory and compute capacity of the system. Systems can exploit this layer using technologies such as GPUDirect RDMA [19], which allows NICs to directly read from and write to GPU memory without staging through host memory, reducing CPU overhead and data movement costs. In our previous works, we show how to effectively exploit these opportunities to realize distributed GPU joins in a co-processor [13] and a fully GPU-centric setting [11]. The core mechanism for achieving high processing bandwidth in these approaches lies in maximizing the overlap between network communication and local computation.

Remote Storage Layer The remote storage layer extends local storage across the network, decoupling data placement from compute nodes and enabling access to externally hosted data. NVMe-oF provides block-level access to remote storage over fast network fabrics, while cloud object storage exposes data through cloud service interfaces. Our previous work [14] provides an initial performance study of NVMe-oF, but leaves its system integration as future work. Together, these technologies open a promising design space for storage disaggregation across local and remote resources.

Towards Building a Full Engine Our recent contribution [11] demonstrates how to realize a system that can exploit the layers of this disaggregated architecture for query processing. This work shows that TCRs are a promising substrate for cross-layer query processing, as compute, storage, and network primitives are built into these AI-oriented software libraries. Our work also proves that query-processing-specific optimizations on each and across all layers are necessary to achieve saturating end-to-end performance. The following sections describe our contributions to integrating these layers into a GPU-centric, disaggregated database system in more detail, from storage and network layers to combining them into a single engine. Unless otherwise stated, all experiments are conducted on NVIDIA A100 GPUs, Mellanox ConnectX-6 NICs, and Samsung PM173X SSDs.

3 Storage Layer: Processing Beyond GPU Memory

The storage layer addresses a central limitation of GPU-centric query processing: GPU memory capacity. Although this layer does not yet provide full storage disaggregation, because data remains on locally attached SSDs, it extends each GPU node with a high-throughput storage tier. Its architectural role is therefore to make NVMe storage (local and remote) an active part of the GPU data path rather than a passive persistence layer.

This section examines the mechanisms required to stream data efficiently from local SSDs into GPU memory. We focus on GDS, compression, pruning, and GPU-aware file-format configuration. Together, these techniques reduce data movement, avoid OOM failures, and sustain high GPU utilization when datasets exceed GPU memory. In the overall architecture, the local storage layer is the first step from GPU-memory-resident execution toward disaggregated GPU-centric data processing. At the end, we briefly discuss how remote storage can be integrated into the overall architecture, but leave this as an important direction of future work.

3.1 Storage Engine for Local Storage

A storage engine for local SSDs moves data from the storage layer into GPU memory for processing. In a GPU-centric architecture, it maximizes read bandwidth and preserves GPU utilization by matching I/O granularity, compression, and synchronization to the execution pipeline. Based on our recent work [8, 11], we summarize five design lessons for storage engines in GPU data systems.

Synchronous GDS Reads Asynchronous I/O is often expected to improve overlap, but synchronous GDS reads can be more effective for high-throughput reading. We compare synchronous and asynchronous GDS reads across different chunk sizes and degrees of CPU-side I/O parallelism. The results show that multi-threaded synchronous reads saturate

local SSD bandwidth more reliably in practical settings [8, 9]. Synchronous GDS reads, therefore, provide a simple and robust interface for building high-throughput GPU storage engines.

Compression GPU data processing is often I/O-bound because data must be moved into GPU memory before it can be processed. Without compression, the loading rate is capped by SSD bandwidth. Compression can increase the effective read bandwidth by reducing the number of transferred bytes, at the cost of decompression during loading. GPUs are well suited for this trade-off: their high compute throughput can often absorb decompression overhead while the system remains I/O-bound. Compression is therefore widely used in GPU data systems [8, 9, 11, 20–24].

Overlapping Decompression can be overlapped with I/O to further improve end-to-end throughput. Once pipelined with data reads, the decompression cost can be hidden behind I/O time. Our overlapped reader uses multiple CPU threads to issue synchronous GDS calls for compressed chunks, while the GPU performs decompression concurrently with ongoing I/O. In contrast, the blocking reader completes all reads before starting decompression. As shown in Fig. 2, an overlapped reader improves performance with a single SSD and also outperforms a naively parallelized blocking reader in the four-SSD setting. Overlapping I/O and GPU computation has become a central design principle in many GPU systems [8, 9, 11, 21, 22, 25].

Minimal Synchronization After chunking and overlapping, synchronization becomes an important but often overlooked bottleneck. CPU–GPU synchronization is necessary for correctness, but implicit synchronization can serialize the pipeline and reduce throughput. Storage engines from existing systems introduced unnecessary synchronization points [26]. Removing these points eliminates pipeline hiccups and allows data to stream from SSDs to GPUs without stalls [11, 27].

Pruning An efficient storage engine should also avoid unnecessary data loading. Lightweight statistics, such as zonemaps or histograms, can be evaluated before issuing reads to skip irrelevant chunks [28]. Because GPU data processing is often I/O-bound, reducing the amount of data transferred directly improves ingestion efficiency. GPUs provide sufficient compute throughput to apply such pruning at low cost. In our experience, pruning can deliver more than 2× speedup [8].

3.2 Efficient File Formats

File formats define the physical representation of data before it is moved from the storage layer to the computation layer [28, 29]. Their configuration determines how data is transferred, decoded, and compressed along the I/O path. This perspective motivated our recent work on file formats [15], using Parquet as a case study. The main lesson is that poor GPU scan performance is often not caused by Parquet itself, but by file configurations chosen for CPU-oriented systems. Figure 3 summarizes the effect of these configurations on GPU scan performance. The results yield four practical insights for improving file formats on GPUs.

Page Count GPU Parquet readers use pages as units of parallel decoding work. If a column chunk contains too few pages, as in the baseline with only one page [30, 31], the GPU has too little parallel work to schedule, causing decoding kernels to underutilize the device. A higher page count exposes more parallelism during decoding and improves GPU utilization.

Row-Group Size Default Parquet files are often written with row-group (RG) sizes that are reasonable for CPU systems. With GDS, however, these defaults produce small I/O requests that do not saturate SSDs in the GPU I/O stack [8,

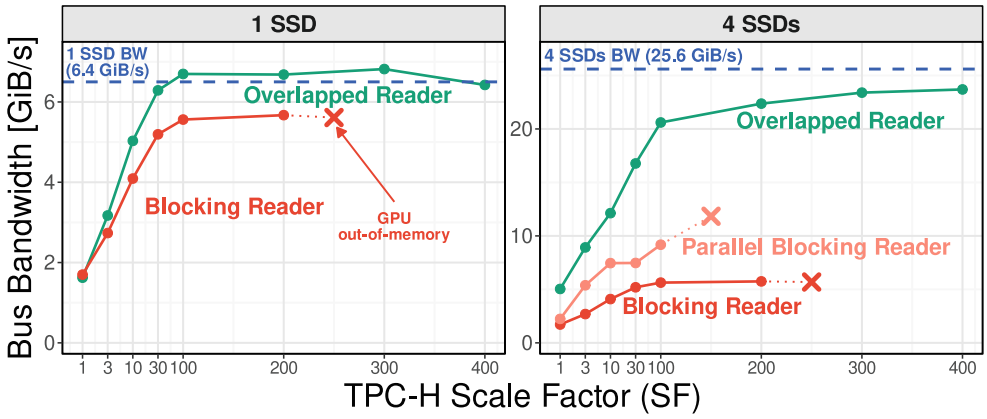

Fig. 2 GPU overlapped reader scan on TPC-H SF300 lineitem: scalability across multiple datasets, compared with the blocking reader and a 4-thread parallel blocking reader

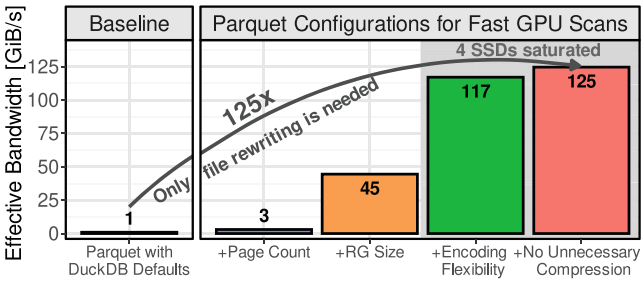

Fig. 3 GPU Parquet scan on TPC-H SF300 lineitem with 4 local NVMe SSDs: file configuration impact on effective read bandwidth. Default Parquet configurations are poorly matched to GPU scans, while GPU-aware Parquet configurations substantially improve read performance

32]. Larger RGs produce larger column chunks and therefore larger I/O units, which better match GDS.

Encoding Flexibility Many Parquet writers restrict themselves to conservative V1 encodings for compatibility, leaving unused opportunities for size reduction. Our rewriter explores valid Parquet encodings and selects the encoding that produces the smallest size for each column chunk. This reduces the number of transferred bytes without changing the file's logical contents.

Compression Compression is beneficial only when the saved I/O outweighs the additional decompression work [33–36]. Our rewriter skips compression when it provides insufficient size reduction. The key point is that compression should be applied selectively: it should reduce I/O enough to justify its GPU decompression cost.

Together, these lessons show that local storage performance depends on the entire GPU-facing I/O path, not only on raw SSD bandwidth. Storage engines must schedule reads, decompression, pruning, and synchronization as part of the GPU execution pipeline, while file formats must be configured with enough parallelism and I/O granularity to keep both SSDs and GPUs utilized during reads. Although this layer still relies on locally attached SSDs, it establishes the external-memory data path that later sections extend across the network and toward full disaggregation.

3.3 Remote Storage

An important direction for future work is extending the data path to the remote storage layer. In modern AI data centers, cloud object storage [37] and network-attached storage, such as NVMe-oF [14], are increasingly accessed through high-speed NICs. While these remote-storage paths promise low overhead in principle, their practical impact remains to be verified in real systems [38]. Moreover, emerging GPU-initiated I/O to remote storage is another direction to explore [39–42].

4 Network Layer: Beyond a Single Machine

The network layer moves beyond single-node GPU processing and enables disaggregation in our architecture. By connecting GPU nodes via high-bandwidth RDMA-capable networks, this layer allows the query engine to distribute computation and use aggregate GPU memory across machines. However, distributed execution improves performance only if communication is integrated into the GPU execution pipeline rather than treated as a separate blocking transfer phase.

We study this layer through distributed repartitioning joins, which are the most communication-intensive operators in analytical query processing. A repartitioning join stresses the interaction between the computation and network layers: tuples must be partitioned, exchanged across nodes, and consumed by GPU-resident join operators. We focus on GPUDirect RDMA, pipelined execution, TCR-based collective communication, and synchronization-aware scheduling. Together, these techniques reduce CPU-mediated data movement, overlap communication with GPU computation, and sustain high network utilization during distributed query processing. In the overall architecture, the network layer is the first step from single-node GPU execution toward disaggregated GPU-centric query processing across multiple compute nodes.

Design Space We discuss two types of GPU-accelerated distributed joins in the compute layer of a disaggregated architecture. First, we summarize CPU-GPU co-processing, in which CPU memory stores the input tables, and GPUs accelerate the local join phase. Second, we discuss GPU-only execution, in which GPU memory and compute resources serve as the primary execution substrate, with CPUs used only for coordination and control flow.

4.1 Distributed CPU-GPU Co-Processing

In our previous work, we studied direct-to-GPU RDMA transfers for distributed joins over tables that reside in CPU main memory [13]. In this setting, the CPU partitions and shuffles the data, while the GPU executes the compute-intensive local join phase. The goal is to combine the higher CPU memory capacity with the high parallel compute throughput of GPUs. The established CPU-based approach uses radix partitioning and an initial histogram phase to compute exclusive write offsets for shuffling. As a baseline, the data path transfers data from the source CPU through a remote CPU before reaching the destination GPU.

Direct GPU Communication A first optimization is to decouple GPU execution from CPU-mediated data movement. Destination GPUs can receive data directly from sender

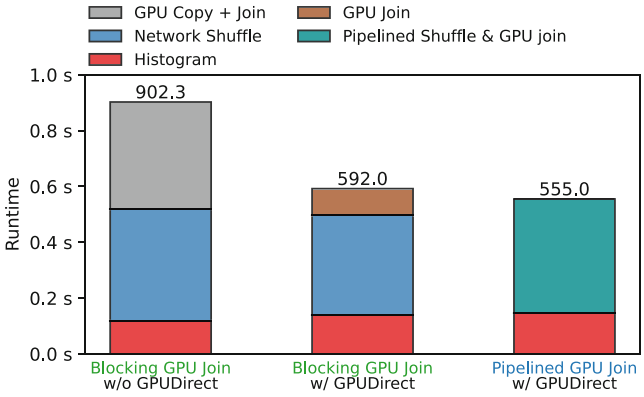

Fig. 4 Runtime of CPU-GPU co-processing distributed joins. Starting with a fully blocking variant, decoupling GPU from data transfer and kernel overheads, towards a fully pipelined GPU join that overlaps shuffling and local join, effectively hiding the latter phase. Here, two V100 GPUs and two Mellanox ConnectX-5 NICs are used

CPUs by using GPUDirect RDMA. This avoids an additional copy through the remote CPU and shortens the network data path. The same design can reduce CPU-initiated kernel-launch overhead by using an active, persistent GPU kernel that polls in-memory receive buffers. As shown by the left and middle bars in Fig. 4, these optimizations improve the blocking baseline by 1.5×.

Chunking and Pipelined Overlap Join runtimes can be further reduced by overlapping local computation with network shuffling. This overlap is achieved by pipelining the processing of horizontal data chunks: while the CPU partitions and shuffles one chunk, the GPU concurrently executes hash-table operations on received chunks. The pipeline hides much of the local join time behind network transfer and improves end-to-end throughput. Pure CPU systems cannot exploit this opportunity, since partitioning and shuffling already consume CPU resources.

As shown by the right bar in Fig. 4, the pipelined GPU join improves only modestly over the optimized blocking variant in this experiment, where only the join operator is considered. However, for full queries, where additional downstream operators increase local computation, the benefit of pipelining grows because more computation can be hidden behind network transfers. Another benefit of the pipelined join is its robustness to large probe-side inputs: the system can process arbitrarily large probe tables without requiring them to fit fully in GPU memory.

4.2 Distributed GPU-Only Processing

Our current system explores GPU-only distributed query processing using the PyTorch TCR [11]. This design builds on TCRs since they have shown strong performance for query processing [12] and provide distribution abstractions for RDMA-capable networks. These abstractions include collective communication primitives from the NCCL library [43], which can be used to realize the shuffling phase of the repartitioning join. The previous partitioning and the local join can be implemented via remainder/select and scatter/gather primitives in TCRs, respectively.

Pipelining with TCR Primitives TCR primitives can deliver high performance when executed in isolation, but they are

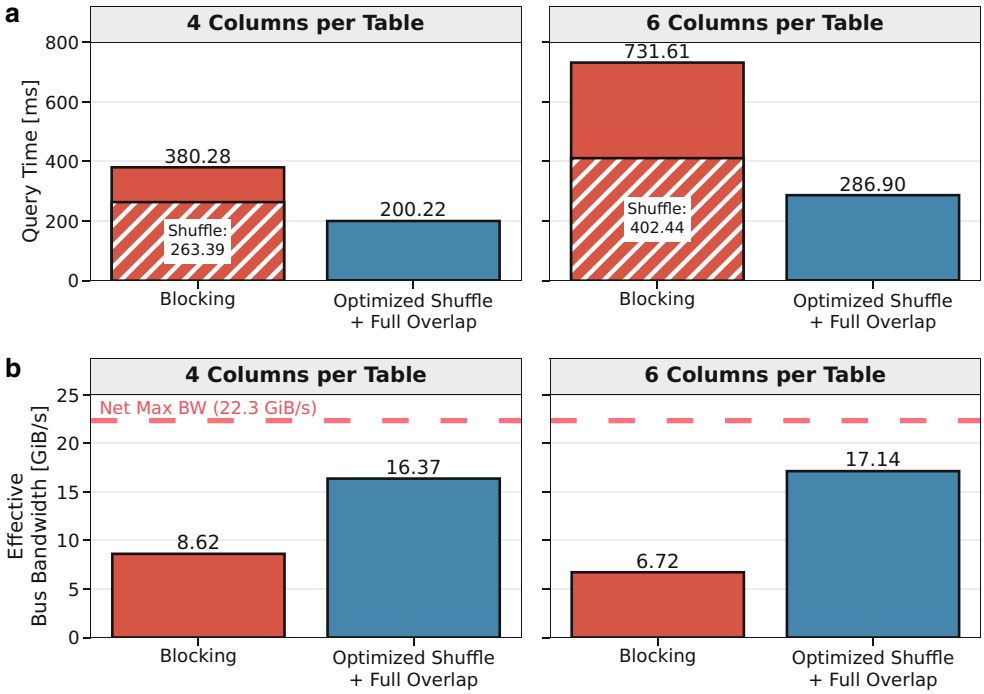

Fig. 5 Distributed join performance in a GPU-only system using 2 nodes (120M/320M build/probe tuples). Fully overlapping partitioning, shuffling, and local join computation allows the system to hide computation time and achieve high network bandwidth. The blue variant combines optimized shuffling with full overlap, whereas the red blocking baseline uses unoptimized shuffling. In **a**), the hatched bars show network shuffle time for the blocking variant. **a** Runtime of blocking versus overlapped join variants. **b** Effective network bandwidth of joins (total data sent/received per node divided by runtime)

not designed primarily for concurrent execution. In AI workloads, computation and communication often occur sequentially, and communication overhead is amortized by substantially larger computation costs. In analytical query processing, however, computation and communication phases often have comparable durations, as shown by the blocking variant in Fig. 5. This creates a significant opportunity for overlapping these phases.

A naive chunked pipeline, however, does not provide sufficient overlap since many required primitives synchronize CUDA streams and thereby enforce blocking execution. These synchronization points follow from the TCR execution model: returned tensors must be correctly sized, which limits asynchronous execution when result sizes are data-dependent, as in query processing. Our system mitigates this problem by scheduling chunk operations so that synchronizing partitioning and join primitives occur while asynchronous shuffling is already in progress. This scheduling strategy enables sustained overlap and hides computation behind data shuffling, as shown by the fully overlapped variant in Fig. 5.

Trade-off Analysis A fully overlapped GPU-only design must balance GPU memory and compute resources for concurrently performing partitioning, network shuffling, and local join processing. It must also choose chunk sizes that amortize scheduling overheads. Figure 6 evaluates this trade-off by varying the sizes of raw transfers and gradually adding concurrent partitioning and local join computation. The results show that GPUs can fully utilize high-speed networks for fixed-size transfers using AllToAll collectives. Adding GPU-based partitioning and local join operations introduces only modest overhead relative to raw data transfer, so end-to-end performance remains close to the network's maximum effective throughput. Without tuned network primitives, the naive overlapped variant performs substantially worse. The remaining gap between the optimized fully overlapped variant and the peak bus bandwidth is due to a known issue with NCCL collective kernels competing for GPU memory bandwidth.

Together, these lessons show that network performance depends on the entire GPU-facing communication path, not only on raw network bandwidth. Distributed query engines must schedule partitioning, shuffling, and local join processing within a single GPU execution pipeline. Direct GPU communication reduces CPU-mediated data movement, while chunked execution and synchronization-aware scheduling enable concurrent communication and computation. Although this layer focuses on distributed computation, it establishes the distributed execution path that the next section combines with local storage I/O to approach full cross-layer execution.

5 Putting all Together in One Disaggregated GPU Engine

The previous sections considered the local storage and network layers separately. The local storage layer extends each GPU node with a high-throughput external-memory path, while the network layer enables distributed execution across GPU nodes. However, a GPU-centric query engine cannot achieve high end-to-end performance by simply stacking these independently optimized layers. If storage first loads all input data, the system risks exhausting GPU memory; if network shuffling begins only after loading completes, the network remains idle during storage I/O; and if storage

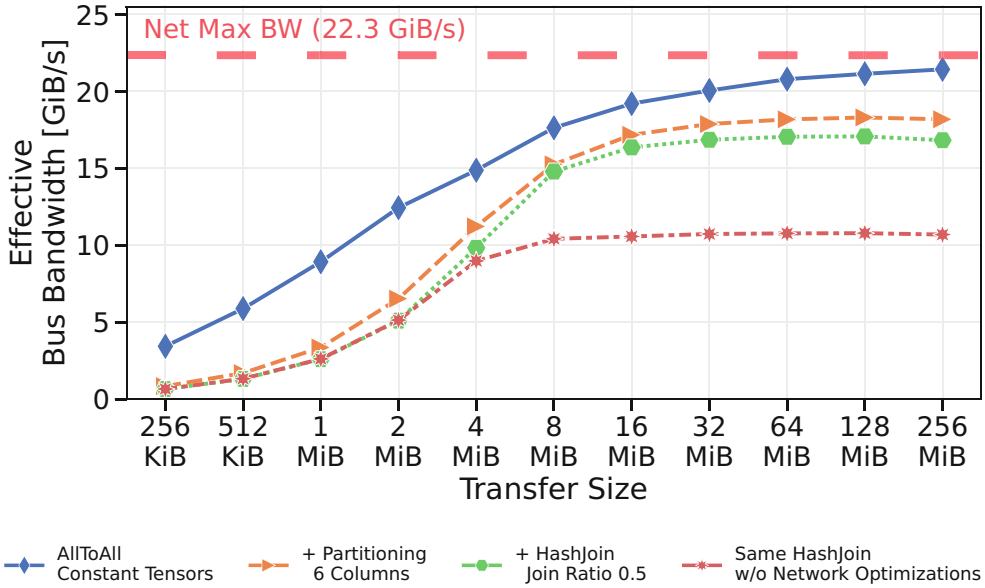

Fig. 6 Bus bandwidth for distributed joins between GPUs using network primitives. Starting from raw collective performance, adding concurrent partitioning, and finally local join computation. The red line shows naive, overlapped performance and demonstrates the necessity of optimizing the use of the collectives for performant concurrent communication and computation

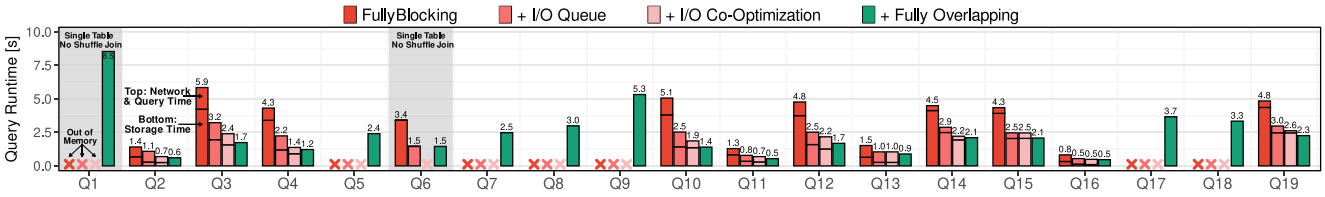

Fig. 7 TPC-H SF500 runtime breakdown with 2 GPUs. Blocking variants are shown in three shades of red, with each bar decomposed into network and query time (top) and storage table scan time (bottom)

and network transfers are not jointly balanced, neither layer reaches its peak bandwidth.

This section examines the mechanisms required to coordinate storage, network, and computation in a single GPU execution pipeline, moving the system closer to fully disaggregated GPU-centric data processing. We focus on I/O queues, storage-network co-optimization, full overlap, and memory regulation through backpressure. Together, they regulate the flow of data from local SSDs through GPU memory and across the network, avoid OOM failures, and sustain high utilization across both storage and network devices.

5.1 Query Engine Design

The query engine coordinates data movement and computation across the storage, network, and computation layers. It reads data from local storage, feeds GPU-resident operators, and invokes the network layer when operators must repartition data across GPUs. Simply connecting these layers sequentially leaves substantial performance on the table. We therefore summarize four design lessons for building query engines that efficiently coordinate storage and network I/O.

I/O Queue Once storage and network are optimized individually, the first challenge is to coordinate data movement between them. We introduce an I/O queue as an abstraction that connects the local storage layer with the computation layer. In the basic design, execution is still sequential: the storage engine first loads and enqueues all data chunks, and the query engine starts dequeuing only after the loading phase completes. The query engine then drives the network layer to repartition data across GPUs for distributed join processing. In our end-to-end TPC-H evaluation, this design provides the first mechanism to combine optimized storage and network I/O within the query engine, shown as +I/O Queue in Fig. 7. FullyBlocking is the baseline in which both storage and network execution remain unoptimized.

I/O Co-Optimization The I/O queue allows the query engine to leverage fast storage and networking, but it does not, by itself, guarantee efficient coordination across the two layers. We observe two bottlenecks. First, queries often read multiple tables, and sequentially scheduled readers introduce bandwidth drops at table boundaries: the tail of one table is read with limited bandwidth before the next table starts. Second, predicate filtering reduces chunk sizes before shuffling, producing chunks that are too small to saturate the network. To address these bottlenecks, we co-optimize storage and network I/O.

On the storage side, we merge table readers to avoid bandwidth hiccups at table boundaries and keep storage access continuous. On the network side, we re-buffer filtered chunks before shuffling, aggregating them into larger messages that better match the network (cf. Fig. 6). This also alleviates the effects of reduced transfer sizes resulting from scaling to larger GPU counts. As shown in Fig. 7, +I/O Co-Optimization further improves over the queue-only design.

Fully Overlapping Even with I/O co-optimization, storage and networked query processing are still executed sequentially: the storage layer must finish loading data before the query engine starts processing and repartitioning. This design can easily lead to OOM failures since base tables are fully materialized in GPU memory before downstream operators consume them. To avoid this problem, we use the I/O queue to overlap storage, network, and computation: While the storage layer continues to enqueue new chunks, the query engine already dequeues them and starts network shuffling. This avoids full materialization of base tables and leaves GPU memory primarily for operator state, such as hash tables. As shown by +Fully Overlapping in Fig. 7, full overlap reduces the risk of OOM failures and improves runtime.

Memory Regulation Full overlap improves utilization, but also makes GPU memory consumption a new scheduling concern. OOM errors occur when the storage layer loads data into GPU memory faster than downstream operators consume it. The I/O queue provides a natural control point for memory regulation through a maximum queue size. Once full, storage blocks before producing more chunks, creating backpressure across storage, network, and computation. As shown by the gray areas in Fig. 8, Adaptive Regulation on I/O Queues avoids OOM errors by throttling data

ingestion. This slightly slows reading but prevents memory pressure from terminating the query.

5.2 Scalability, System Comparison, and Cost

We further evaluate the cross-layer design from three perspectives. First, we study whether the system scales with increasing dataset size and GPU count. Second, we evaluate whether memory regulation enables larger workloads to complete without OOM failures. Third, we compare runtime and cost with CPU- and GPU-based baselines for storage-resident workloads.

Scalability We study scalability using TPC-H Query 3 as a representative workload. The left panel of Fig. 8 reports end-to-end runtime as the dataset size increases from TPC-H SF100 to SF1000 on 2 GPUs. With all optimizations enabled, the system scales nearly linearly with the scale factor and remains close to the theoretical minimum, computed as the maximum of ideal storage time and ideal network time under perfect overlap. The right panel of Fig. 8 reports GPU scalability on TPC-H SF1000 using up to all 8 GPUs in the DGX system. This experiment shows how many GPUs are required for different variants to avoid OOM failures and complete the query. Across both experiments, the optimized configuration handles larger scale factors and runs with fewer GPUs, as indicated by the gray regions.

System Comparison We extend the evaluation to additional queries and compare our system against DuckDB [44] and ClickHouse [45], two CPU-based systems with strong OLAP performance on storage-resident datasets. We also compare against Sirius [46], a GPU system that relies on CPU-based DuckDB for Parquet ingestion. We restrict this comparison to single-node query processing over SSD-resident data, since these systems do not support distributed repartition joins. As shown in the top part of Fig. 9, our system outperforms all baselines, mainly due to the efficiency of its GPU-native storage engine.

Cost Beyond runtime, we also evaluate cost per query across the compared systems. The lower part of Fig. 9 converts the measured runtimes into total-cost-of-ownership (TCO) estimates using cloud VM prices for hardware comparable to our setup. Our system achieves at least 4× lower TCO on the evaluated queries.

These results show that GPU data processing can provide both performance benefits and substantial cost efficiency

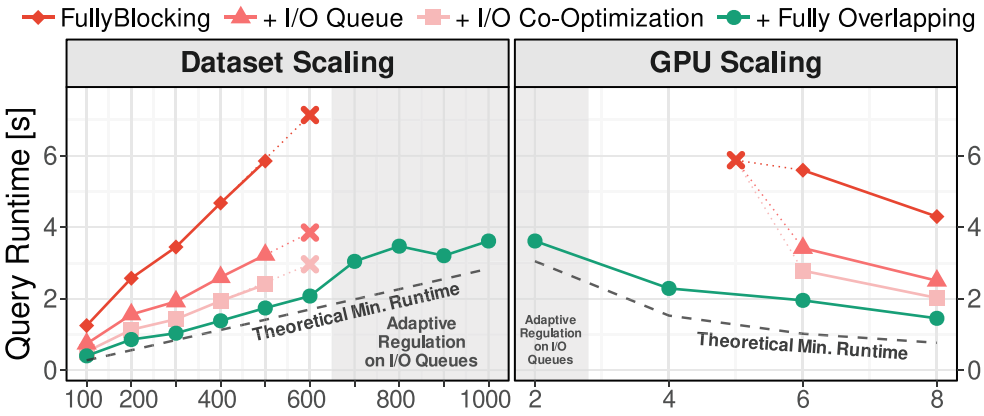

Fig. 8 Scalability experiments with TPC-H Query 3. Left: dataset scalability with 2 GPUs. Right: GPU scalability on SF1000

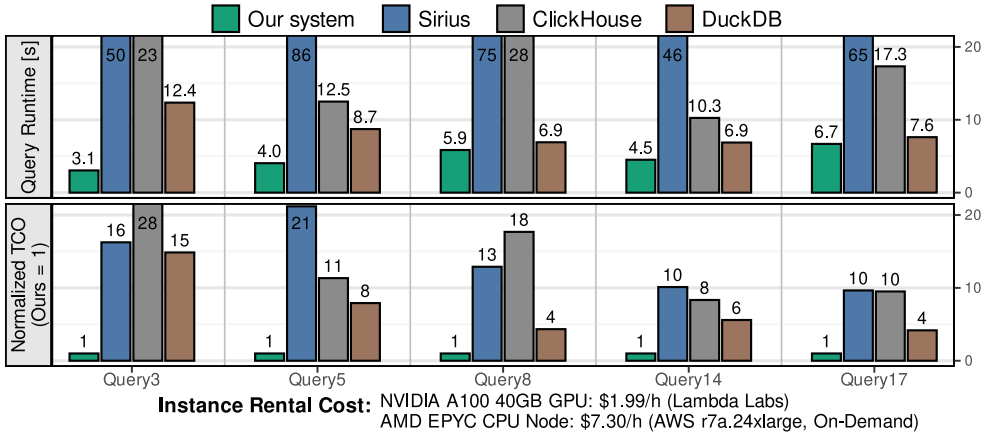

Fig. 9 System comparison on TPC-H SF500 runtime and normalized TCO in a single-node setting with fully storage-resident data

when the storage engine is efficient enough to keep GPUs well-utilized.

Together, these results show that cross-layer query performance depends on the entire GPU-facing data path, not only on the efficiency of individual storage or network components. The query engine must schedule loading, repartitioning, and local query processing as one coordinated pipeline. I/O queues provide the abstraction for connecting storage and computation, co-optimization preserves I/O device bandwidth across layer boundaries, and backpressure prevents storage from overwhelming GPU memory. Although this section still focuses on local SSDs and distributed compute nodes, it establishes the cross-layer execution model needed for future systems that incorporate remote storage and more advanced forms of disaggregation.

6 Future Research Directions

This section discusses future directions and open challenges for disaggregated GPU data processing.

File Formats for GPUs Several encodings for GPUs [47–50] have been proposed to make decoding highly parallelizable. These schemes report extremely high in-GPU-memory decoding throughput [33, 51]. However, it remains unclear whether these encodings can perform well in storage- and disaggregation-oriented settings.

Memory Pooling Instead of relying on explicit network primitives to manage memory in the nodes of the compute layer, modern interconnects such as NVLink [6, 52] or CXL [53, 54] allow for pooling memory to scale working set sizes across aggregate memory. However, how to optimally utilize these opportunities remains unclear.

Workloads Beyond OLAP Although this work focuses on analytical workloads, the proposed architecture is not limited to them. The compute layer on top of a disaggregated data system can evolve toward other GPU-friendly workloads, such as machine learning [10, 25, 55], vector search [56–58], and transaction processing [59–61].

7 Conclusion

Emerging AI hardware in data centers presents opportunities for future-proof database systems to profit from advancements in compute power and I/O capabilities. This paper explored the layers of the future data processing stack and summarized our recent work on integrating GPU acceleration with high-bandwidth storage and fast networks to build high-performance disaggregated analytical data systems for the AI era.

Acknowledgements This research was partially funded by the state of Hesse as part of the NHR program (NHR4CES), the LOEWE Spitzenprofessur of the state of Hesse (III 5-519/05.00.003-(0005)), the project “hessian.AI AISC” (Grant No. 01IS22091) by the Federal Ministry of Education and Research (BMBF), as well as by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy (EXC3057/1 “Reasonable Artificial Intelligence”, Project No. 533677015). We also thank DFKI Darmstadt and hessian.AI for their support.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability All data supporting the findings of this work are available at: https://github.com/DataManagementLab/PystachIO and https://github.com/DataManagementLab/ParquetRewriter.

References

1. Noffsinger, J., Goodpaster, M., Patel, M., Chang, H., Sachdeva, P., Bhan, A.: The cost of compute: A 7 trillion race to scale data centers. McKinsey & Company (2025). https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-cost-of-compute-a-7-trillion-dollar-race-to-scale-data-centers
2. Amazon (2025) Amazon to invest up to 50 billion to expand AI and supercomputing infrastructure for US Government agencies. https://www.aboutamazon.com/news/company-news/amazon-ai-investment-us-federal-agencies
3. Ye Z, Gao W, Hu Q, Sun P, Wang X, Luo Y, Zhang T, Wen Y (2024) Deep Learning Workload Scheduling in GPU Datacenters: A Survey. ACM Comput. Surv 56(6):146–114638. https://doi.org/10.1145/3638757
4. Gupta T (2024) Evolution of Data Center Design to Handle AI Workloads. 34th International Telecommunication Networks and Applications Conference, ITNAC 2024, Sydney, Australia, November 27–29, 2024. https://doi.org/10.1109/ITNAC62915.2024.10815309
5. Interlandi M, Bruno N, Haynes B, Curino C, Sen R, Li Y, Rajan K, Ding B, Maas LM, Cui W, Gaffney K, Hong M, Kroth B, Rajenda S, Cheng P, Chaudhuri S, Gehrke J, Ramakrishnan R, Zhou L, Al-Ghosien M, Peeper C, Dumitru M, Cunningham C, Bocksrocker K, Sundar P, Boskovic R, Kwon Y, Zivanovic M, Shi R, Sakowski K, Lamtsov D, Aguilar Saborit J, Srinivasan K, Kapil A, Putnam A, Kambapu A, Konduri A, Landy A, Moore A,

Pitman A, Oks A, Gosalia A, Kandimalla B, Pelton B, Crivat B, Galindo-Legaria C, Kulkarni C, Camacho Rodriguez J, Babin E, Lehnert Marino H, Konev I, Arroyo I, Penaranda L, Datar M, Borup Petersen M, Simmons N, Poonian P, Danda P, Rydberg R, Calvo E, Bajaj R, Sumanth Kammal Shetty S, Bhat S, Iracheta Z (2026) CoddSpeed: Hardware Accelerated Query Processing in Microsoft Fabric. Companion of the International Conference on Management of Data. SIGMOD Companion '26. https://doi.org/10.1145/3788853.3803077

6. Wu B, Cui CuiW, Curino C, Interlandi M, Sen R (2025) Terabyte-Scale Analytics in the Blink of an Eye. Proc. VLDB Endow 19:141–155
7. NVIDIA (2025) NVIDIA GPUDirect Storage. https://docs.nvidia.com/gpudirect-storage/. Accessed 4 Aug 2025
8. Boeschen N, Ziegler T, Binnig C (2024) GOLAP: A GPU-in-Data-Path Architecture for High-Speed OLAP. Proc. ACM Manag. Data 2(6):237–123726. https://doi.org/10.1145/3698812
9. Nicholson H, Chasialis K, Boffa A, Ailamaki A (2025) The Effectiveness of Compression for GPU-Accelerated Queries on Out-of-Memory Datasets. Proceedings of the 21st International Workshop on Data Management on New Hardware, DaMoN 2025, Berlin, Germany, June 22–27, 2025. https://doi.org/10.1145/3736227.3736240
10. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8–14, 2019, Vancouver, BC, Canada, pp. 8024–8035 (2019). https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html
11. Luo J, Boeschen N, El-Hindi M, Binnig C (2026) PystachIO: Efficient Distributed GPU Query Processing with PyTorch over Fast Networks & Fast Storage. Proc VLDB Endow 19(9):2494–2507. https://doi.org/10.14778/3819518.3819566
12. He D, Nakandala SC, Banda D, Sen R, Saur K, Park K, Curino C, Camacho-Rodríguez J, Karanasos K, Interlandi M (2022) Query Processing on Tensor Computation Runtimes. Proc. VLDB Endow 15(11):2811–2825. https://doi.org/10.14778/3551793.3551833
13. Thostrup L, Doci G, Boeschen N, Luthra M, Binnig C (2023) Distributed GPU Joins on Fast RDMA-capable Networks. Proc. ACM Manag. Data 1(1):29–12926. https://doi.org/10.1145/3588709
14. Luo J, Boeschen N, Ziegler T, Binnig C (2025) An Evaluation of NVMe-over-Fabrics for Disaggregated Databases over Fast Networks. In: Klettke M, Schenkel R, Henrich A, Nicklas D, Schüle ME, Meyer-Wegener K (eds) Datenbanksysteme Für Business, Technologie und Web (BTW 2025), 21. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 03.-07, März 2025, Bamberg, Germany, Proceedings. LNI. Gesellschaft für Informatik e.V., Bonn, Germany, pp 113–125 https://doi.org/10.18420/BTW2025-05
15. Luo J, Chen Q, Binnig C (2026) Do GPUs Really Need New Tabular File Formats? Proceedings of the 22nd International Workshop on Data Management on New Hardware, DaMoN 2026, Bengaluru, India. https://doi.org/10.1145/3789237.380912
16. Jasny M, Thostrup L, Tamimi S, Koch A, István Z, Binnig C (2024) Zero-sided RDMA: Network-driven Data Shuffling for Disaggregated Heterogeneous Cloud DBMSs. Proc. ACM Manag. Data 2(1):36–13628. https://doi.org/10.1145/3639291
17. Jasny M, Thostrup L, Binnig C (2023) Zero-sided RDMA: Network-driven Data Shuffling. Proceedings of the 19th International Workshop on Data Management on New Hardware, DaMoN 2023, Seattle, WA, USA, June 18–23, 2023. https://doi.org/10.1145/3592980.3595302
18. Vohra D (2016) Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools, 1st edn edn. Apress, USA https://doi.org/10.1007/978-1-4842-2199-0
19. NVIDIA (2025) NVIDIA GPUDirect RDMA. https://docs.nvidia.com/cuda/gpudirect-rdma/. Accessed 4 Aug 2025
20. Patterson J, Mostak T (2026) NVIDIA GTC 2026: The Era of GPU Data Processing: From SQL to Search and Back Again. https://www.nvidia.com/en-us/on-demand/session/gtc26-s81769/. Accessed 19 Apr 2026
21. Yeo G, Shen Z, Cui W, Interlandi M, Sen R, Ding B, Chen Q, Rhu M (2026) ZipFlow: a Compiler-based Framework to Unleash Compressed Data Movement for Modern GPUs. CoRR. https://doi.org/10.48550/ARXIV.2602.08190
22. Jiang, Y., Nicholson, H., Sanca, V., Ailamaki, A.: Data Movement-Aware GPU Sharing for Data-Intensive Systems. In: 16th Conference on Innovative Data Systems Research, CIDR 2026, Chaminade, CA, USA, January 18–21, 2026. www.cidrdb.org, Chaminade, CA, USA (2026). https://vldb.org/cidrdb/2026/data-movement-aware-gpu-sharing-for-data-intensive-systems.html
23. Li Y, Ding B, Wei Z, Maas LM, Al-Ghosien M, Blanas S, Bruno N, Curino C, Interlandi M, Peeper C, Rajan K, Chaudhuri S, Gehrke J (2025) Scaling GPU-Accelerated Databases beyond GPU Memory Size. Proc. VLDB Endow 18(11):4518–4531. https://doi.org/10.14778/3749646.3749710
24. Huang Z, Sakowski K, Lehnert H, Cui W, Curino C, Interlandi M, Dumitru M, Sen R (2025) GPU Acceleration of SQL Analytics on Compressed Data. Proc. VLDB Endow 19:320–333
25. Yuan Y, Iyer A, Ma L, Talati N (2024) Vortex: Overcoming Memory Capacity Limitations in GPU-Accelerated Large-Scale Data Analytics. Proc. VLDB Endow 18(4):1250–1263. https://doi.org/10.14778/3717755.3717780
26. Luo, J.: cuDF Issue: Unstable pipelining performance in Parquet reading due to mis-sync. https://github.com/rapidsai/cudf/issues/18967. Accessed: 2025-11-26 (2025)
27. Luo J (2025) [Story] Towards a faster Parquet reader with pipelining and multistream optimization. https://github.com/rapidsai/cudf/issues/18892. Accessed 27 Nov 2025
28. Zimmerer A, Dam D, Kossmann J, Waack J, Oukid I, Kipf A (2025) Pruning in Snowflake: Working Smarter, Not Harder. Companion of the 2025 International Conference on Management of Data, SIGMOD/PODS 2025, Berlin, Germany, June 22–27, 2025. https://doi.org/10.1145/3722212.3724447
29. Dageville B, Cruanes T, Zukowski M, Antonov V, Avanes A, Bock J, Claybaugh J, Engovatov D, Hentschel M, Huang J, Lee AW, Motivala A, Munir AQ, Pelley S, Povinec P, Rahn G, Triantafyllis S, Unterbrunner P (2016) The Snowflake Elastic Data Warehouse. Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016. https://doi.org/10.1145/2882903.2903741
30. DuckDB (2026) DuckDB Performance Guide: File Formats. https://duckdb.org/docs/stable/guides/performance/file_formats. Accessed 1 Feb 2026
31. (2026) DuckDB: DuckDB Parquet Tips. https://duckdb.org/docs/stable/data/parquet/tips. Accessed 1 Feb 2026
32. Torp KB, Lund SAF, Tözün P (2025) Path to GPU-Initiated I/O for Data-Intensive Systems. Proceedings of the 21st International Workshop on Data Management on New Hardware, DaMoN 2025, Berlin, Germany, June 22–27, 2025. https://doi.org/10.1145/3736227.3736232
33. Lamb A, Dem JL (2026) Talk: Column Storage for the AI Era. https://hpi.de/rabl/teaching/winter-term-2025-26/joint-database-systems-seminar-with-tu-darmstadt/andrew-lamb.html

34. Kuschewski M, Sauerwein D, Alhomssi A, Leis V (2023) Btr-Blocks: Efficient Columnar Compression for Data Lakes. Proc. ACM Manag. Data 1(2):118–111826. https://doi.org/10.1145/3589263
35. Zeng X, Hui Y, Shen J, Pavlo A, McKinney W, Zhang H (2023) An Empirical Evaluation of Columnar Storage Formats. Proc. VLDB Endow 17(2):148–161. https://doi.org/10.14778/3626292.3626298
36. Zeng X, Meng R, Prammer M, McKinney W, Patel JM, Pavlo A, Zhang H (2025) F3: The Open-Source Data File Format for the Future. Proc. ACM Manag. Data 3(4):245–124527. https://doi.org/10.1145/3749163
37. Durner D, Leis V, Neumann T (2023) Exploiting Cloud Object Storage for High-Performance Analytics. Proc. VLDB Endow 16(11):2769–2782. https://doi.org/10.14778/3611479.3611486
38. Aramburú F, Malpica W, Abrougui K, Aramoon A, Auccapuclla R, Brisson C, Brobbel M, Farrell C, Garigipati P, Hoozemans J, Kamburugamuve S, Nair A, Ocsa A, Peltenburg J, López RQ, Sihag D, Uyar A, Vats D, Wendt M, Patel JM, Aramburú R (2025) Theseus: A Distributed and Scalable GPU-Accelerated Query Processing Platform Optimized for Efficient Data Movement. CoRR. https://doi.org/10.48550/ARXIV.2508.05029
39. Newburn CJ, Prabhu P, Mailthody VS (2025) Speed-of-Light Data Movement Between Storage and the GPU. https://www.nvidia.com/en-us/on-demand/session/gtc25-s73012/. Accessed 8 June 2026
40. Mellor C (2025) Nvidia SCADA offloads storage control path to the GPU. https://www.blocksandfiles.com/ai-ml/2025/11/25/nvidia-scada-offloads-storage-control-path-to-the-gpu/1711995. Accessed 8 June 2026
41. Qureshi Z, Mailthody VS, Gelado I, Min S, Masood A, Park JB, Xiong J, Newburn CJ, Vainbrand D, Chung I, Garland M, Dally WJ, Hwu WW (2023) GPU-Initiated On-Demand High-Throughput Storage Access in the BaM System Architecture. Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems 2:325–339. https://doi.org/10.1145/3575693.3575748
42. Markussen J, Kristiansen LB, Halvorsen P, Kielland-Gyrud H, Stensland HK, Griwodz C (2020) SmartIO: Zero-overhead Device Sharing through PCIe Networking. ACM Trans. Comput. Syst 38(1-2):2–1278. https://doi.org/10.1145/3462545
43. NVIDIA (2025) NCCL. https://developer.nvidia.com/nccl. Accessed 11 Aug 2025
44. Raasveldt M, Mühleisen H (2019) DuckDB: an Embeddable Analytical Database. Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019. https://doi.org/10.1145/3299869.3320212
45. Schulze R, Schreiber T, Yatsishin I, Dahimene R, Milovidov A (2024) ClickHouse - Lightning Fast Analytics for Everyone. Proc. VLDB Endow 17(12):3731–3744. https://doi.org/10.14778/3685800.3685802
46. Yogatama, B., Yang, Y., Kristensen, K., Sarda, D., Kim, A., Cockcroft, A., Teng, Y., Patterson, J., Kimball, G., McKinney, W., Gong, W., Yu, X.: Rethinking Analytical Processing in the GPU Era. In: 16th Conference on Innovative Data Systems Research, CIDR 2026, Chaminade, CA, USA, January 18–21, 2026. www.cidrdb.org, Chaminade, CA, USA (2026). https://www.cidrdb.org/cidr2026/keynotes.html
47. Afroozeh A, Boncz P (2025) The FastLanes File Format. Proc. VLDB Endow 18(11):4629–4643. https://doi.org/10.14778/3749646.3749718
48. Hepkema S, Afroozeh A, Felius C, Boncz P, Manegold S (2025) G-ALP: Rethinking Light-weight Encodings for GPUs. Proceedings of the 21st International Workshop on Data Management on New Hardware, DaMoN 2025, Berlin, Germany, June 22–27, 2025. https://doi.org/10.1145/3736227.3736242
49. Afroozeh A, Felius L, Boncz P (2024) Accelerating GPU Data Processing using FastLanes Compression. Proceedings of the 20th International Workshop on Data Management on New Hardware, DaMoN 2024, Santiago, Chile. https://doi.org/10.1145/3662010.3663450
50. Vonk R, Hoozemans J, Al-Ars Z (2025) GSST: Parallel string decompression at 191 GB/s on GPU. ACM SIGOPS Oper. Syst. Rev 59(1):55–61. https://doi.org/10.1145/3759441.3759450
51. Manning W (2025) FutureData: Vortex: LLVM for File Formats. https://db.cs.cmu.edu/events/futuredata-vortex/. Accessed 1 Feb 2026
52. Lutz C, Breß S, Zeuch S, Rabl T, Markl V (2020) Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects. Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, Online Conference [Portland, OR, USA], June 14–19, 2020. https://doi.org/10.1145/3318464.3389705
53. Lerner A, Alonso G (2024) CXL and the Return of Scale-Up Database Engines. Proc. VLDB Endow 17(10):2568–2575. https://doi.org/10.14778/3675034.3675047
54. Weisgut M, Ritter D, Tözün P, Benson L, Rabl T (2025) CXL Memory Performance for In-Memory Data Processing. Proc. VLDB Endow 18(9):3119–3133. https://doi.org/10.14778/3746405.3746432
55. Paganelli M, Sottovia P, Park K, Interlandi M, Guerra F (2023) Pushing ML Predictions Into DBMSs. IEEE Trans Knowl Data Eng 35(10):10295–10308. https://doi.org/10.1109/TKDE.2023.3269592
56. Johnson J, Douze M, Jégou H (2021) Billion-Scale Similarity Search with GPUs. IEEE Trans Big Data 7(3):535–547. https://doi.org/10.1109/TBDATA.2019.2921572
57. Krishnaswamy R, Manohar MD, Simhadri HV (2024) The DiskANN library: Graph-Based Indices for Fast, Fresh and Filtered Vector Search. IEEE Data Eng. Bull 48:20–42
58. Subramanya, S.J., Devvrit, Kadekodi, R., Krishaswamy, R., Simhadri, H.V.: DiskANN: fast accurate billion-point nearest neighbor search on a single node. Curran Associates Inc., Red Hook, NY, USA (2019). https://proceedings.neurips.cc/paper_files/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf
59. Boeschen N, Binnig C (2022) GaccO - A GPU-accelerated OLTP DBMS. SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022. https://doi.org/10.1145/3514221.3517876
60. Qian, S., Goel, A.: Massively Parallel Multi-Versioned Transaction Processing. In: Gavrilovska, A., Terry, D.B. (eds.) 18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10–12, 2024, pp. 765–781. USENIX Association, Berkeley, CA, USA (2024). https://www.usenix.org/conference/osdi24/presentation/qian
61. Boeschen N, Binnig C (2021) GalOP: Towards a GPU-accelerated OLTP DBMS. Proceedings of the 17th International Workshop on Data Management on New Hardware, DaMoN 2021, 21 June 2021, Virtual Event, China. https://doi.org/10.1145/3465998.3466007