

Terabyte-Scale Analytics in the Blink of an Eye

Bowen Wu^{*}[†]

ETH Zurich

bowen.wu@inf.ethz.ch

Wei Cui[†]

Microsoft Research Asia

weicu@microsoft.com

Carlo Curino

Gray Systems Lab, Microsoft

carlo.curino@microsoft.com

Matteo Interlandi

Gray Systems Lab, Microsoft

matteo.interlandi@microsoft.com

Rathijit Sen

Gray Systems Lab, Microsoft

rathijit.sen@microsoft.com

ABSTRACT

For the past two decades, the DB community has devoted substantial research to take advantage of cheap clusters of machines for distributed data analytics—we believe that we are at the beginning of a paradigm shift. The scaling laws and popularity of AI models lead to the deployment of incredibly powerful GPU clusters in commercial data centers. Compared to CPU-only solutions, these clusters deliver impressive improvements in per-node compute, memory bandwidth, and inter-node interconnect performance.

In this paper, we study the problem of scaling analytical SQL queries on distributed clusters of GPUs, with the stated goal of establishing an upper bound on the likely performance gains. To do so, we build a prototype designed to maximize performance by leveraging ML/HPC best practices, such as group communication primitives for cross-device data movements. This allows us to conduct thorough performance experimentation to point our community towards a massive performance opportunity of at least 60 \times . To make these gains more relatable, before you can blink twice, our system can run all 22 queries of TPC-H at a 1TB scale factor!

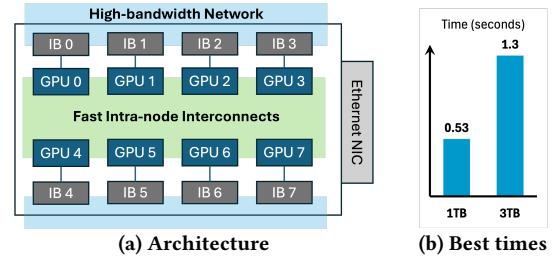
PVLDB Reference Format:

Bowen Wu, Wei Cui, Carlo Curino, Matteo Interlandi, and Rathijit Sen. Terabyte-Scale Analytics in the Blink of an Eye. *PVLDB*, 19(2): 141 - 155, 2025.

doi:10.14778/3773749.3773754

1 INTRODUCTION

The massive parallelism and multi-TB/sec memory bandwidths offered by modern GPUs are hugely beneficial for accelerating SQL analytics queries. Consequently, GPU acceleration for SQL analytics continues to receive much attention, and numerous recent studies have extended the state of the art to accelerate query processing on GPUs [32, 69, 88, 104]. At the same time, driven by the enormous potential and need for supporting the training and inferencing of GenAI models, GPU technology has been rapidly evolving, both within the GPU hardware itself, as well as at the system level with the introduction of multi-GPU machines and



(a) Architecture

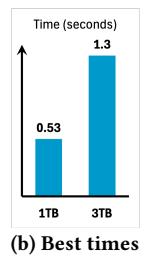


Figure 1: (a) Multi-GPU machine architecture. IB: InfiniBand. (b) Best total run time for all 22 queries of TPC-H using 5 machines (40 GPUs).

high-bandwidth interconnects [3, 19, 20]. We believe that such advances have opened new opportunities for SQL acceleration, at scales and speeds unimaginable just a few years ago. Multi-GPU machines will drive the next wave of acceleration for analytics.

Single-GPU acceleration for SQL analytics has traditionally faced two performance bottlenecks [80]: (1) limited high-bandwidth memory (HBM) capacity on the GPU; and (2) low CPU-GPU data movement bandwidth (over PCIe) compared to available CPU-memory bandwidths. The recent availability of multi-GPU machines from NVIDIA and AMD has alleviated these concerns by offering opportunities for resource aggregation. For example, the NVIDIA A100/H100 and AMD MI300X eight-GPU machines have an aggregate HBM capacity of 640 GiB and 1.5 TiB per machine, respectively, thereby enabling larger datasets to remain resident in GPU HBM without requiring frequent CPU-GPU transfers. With a separate PCIe link per GPU, the aggregate CPU-GPU PCIe bandwidth in multi-GPU machines is comparable to the CPU main memory bandwidths available in high-end dual-socket servers today. Additionally, multi-GPU machines support high-bandwidth data exchange between GPUs, within and across machines, thereby enabling high scale-up and scale-out performance for distributed query processing—data exchange operations for shuffle and broadcast play a critical role in running analytical queries at scale and are often limited by the network bandwidth. But not anymore!

Figure 1a shows the architecture of a scale-up 8-GPU machine, which we use as a building block for our scale-out cluster setups. The GPUs are interconnected by a high-bandwidth backplane network, made up of NVLinks and NVSwitches for NVIDIA multi-GPU machines, and Infinity Fabric for AMD multi-GPU machines. The bandwidth for both outgoing and incoming traffic to each GPU is in the order of several hundred GB/sec (e.g., up to 450 GB/sec on the NVIDIA 8-H100 DGX machines, which is more than 7 \times

^{*}Work done during an internship at the Microsoft Gray Systems Lab.

[†]Equal contribution.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 19, No. 2 ISSN 2150-8097.
doi:10.14778/3773749.3773754

the CPU-GPU PCIe gen5 bandwidth), and also exceeds the single-socket CPU-main memory bandwidths of most modern servers. The inter-GPU backplane network within these machines offers up to 3.6 TB/sec of aggregate bandwidth for shuffle operations, far exceeding what is available across NUMA CPU sockets [59]. These multi-GPU machines also have impressive scale-out performance. Our highest-performing clusters provide up to 400 GB/sec bandwidth between machines with eight Mellanox NICs, one per GPU. As we will show, this can speed up SQL query performance by more than an order of magnitude compared to using traditional lower-bandwidth Ethernet connectivity (100 Gbits/sec) between machines. Finally, to make things practical, such machines are available in the cloud today [2, 17, 54, 68, 84], thereby expanding their access and affordability to the community with on-demand pricing models.

In this work, we show that distributed SQL query processing at very high speeds is possible using a cluster of multi-GPU machines with high-bandwidth interconnects. While prior systems such as HeavyDB [33] and Theseus [5, 25] have explored SQL acceleration with multiple GPUs, this paper pushes the boundaries of SQL analytics performance on powerful clusters of GPU machines. To achieve this, we implemented a distributed version of Tensor Query Processor (TQP) [32]. With TQP, we took the nonconventional decision of leveraging high-performance ML frameworks (i.e., PyTorch) for running SQL analytics on GPUs. The key idea was to leverage any system improvement in the ML space also for SQL analytics [52]. In this paper, we follow a similar philosophy by implementing data exchange operations using the core group communication primitives used in AI training. We leverage NVIDIA and AMD high-performance implementations of such primitives (i.e., NCCL [18] and RCCL [4]), and natively use their proprietary backend networks, as well as leveraging their algorithms for efficient multi-GPU cross-machine data transfers.

Starting from input data partitioned and loaded in GPU HBMs, we run all 22 TPC-H [96] queries at 1 TB scale in 0.53 seconds in total using 40 GPUs (H100, 5 machines) and 3 TB scale in 1.3 seconds (Figure 1b). We can also complete all 22 queries at 1 TB scale in 1.06 seconds on a single machine with 8 GPUs (MI300X). This beats custom scale-up CPU machines fitting the workload in RAM by over 60x [36, 37] and more for commodity scale-out cloud-based hardware. This is just a snapshot in time of the achievable performance with the hardware that we have access to. In fact, and as we write, next-generation GPUs with even faster network interconnect are being deployed in the cloud [12]. To address how future-generation hardware and interconnects can impact end-to-end performance, we also developed analytical models to get insights into the expected performance as we continue to scale and run TQP on new hardware. We expect the performance that we can achieve on GPU clusters with fast interconnects to drastically outpace CPU equivalents.

In summary, this paper makes the following contributions:

- (1) We show how off-the-shelf group communications libraries that are developed primarily for AI and HPC applications can be adapted to be applied to the SQL analytics domain.
- (2) We show how TQP can be easily extended to take advantage of multiple GPUs across a cluster of machines coming from different vendors, thereby enjoying ease of portability along with highly competitive scale-up and scale-out performance.
- (3) We demonstrate, for the first time, TPC-H 1 TB total query performance of less than a second, and < 1.5 seconds at a 3 TB scale using a cluster of multi-GPU machines in the cloud.
- (4) To gain insights about scalability and extrapolate performance to future hardware, we present analytical performance models of the time-consuming shuffle and broadcast operations.
- (5) We characterize TPC-H workload execution on a multi-GPU cluster and analyze the effect of various factors on workload performance, such as warm/cold run, broadcast implementation, data skew, and data placement. The analysis gives valuable insights into designing an efficient SQL analytics system for a multi-GPU cluster.

We believe that this paper firmly demonstrates the potential for accelerating analytics at scale over multi-GPU clusters in the cloud.

The rest of the paper is organized as follows. We describe our distributed SQL query acceleration approach using the Tensor Query Processor (TQP) and off-the-shelf collective communications libraries in Section 2. We present analytical performance models for data exchange operations in Section 3, with experimental evidence of scaling trends in Section 5. Section 4 lists our cluster configurations. Sections 6 and 7 analyze terabyte-scale TPC-H performance on standard (uniform) and skewed data, respectively, and explore other factors contributing to the performance. Section 8 summarizes related literature and Section 9 concludes the paper. Further details are available in the extended online version [102].

2 APPROACH: DISTRIBUTED TQP

We extend and use TQP to accelerate SQL queries using multiple GPUs. Our approach is to add data exchange operators to tensor programs generated by TQP and use primitives provided by the NVIDIA Collective Communications Library (NCCL) API to implement them. We describe each of these components in this section.

2.1 Background: Tensor Query Processor

Tensor Query Processor (TQP) [7, 32] accelerates analytical queries by implementing relational operators using PyTorch’s tensor API. By leveraging PyTorch, TQP can execute queries on specialized hardware such as GPUs [29, 32] and APUs [23]. The advantage of using PyTorch’s tensor API not only relies on the ease of portability to diverse and rapidly evolving hardware platforms but also on taking advantage of any algorithm improvement coming from the ML community, while maintaining a familiar programming interface.

TQP is composed by a set of utilities for (1) converting input data into tensor format; and (2) a query compiler transforming input queries into tensor programs. For the former, TQP supports Pandas DataFrame [65], Parquet [28], Numpy [97], and CSV as input formats. TQP also support integers, floating point as well as ASCII string (in dictionary and value-encoded formats), and date inputs¹. For the latter, given an input query, TQP uses Apache Spark’s [109] query optimizer (Catalyst) to generate an initial physical query plan for single-node execution. TQP then tensorizes [32] the plan, and compiles it into a tensor program that is fed with the input data (in tensor format) to generate the query results. We reuse TQP’s compilation stack in this work but add additional capabilities for supporting distributed SQL query processing, as we describe next.

¹Notably, decimals are not supported yet. Decimal columns are currently loaded as floating-point numbers.

2.2 Background: Data Exchange for Distributed Query Processing

Data exchange is a costly operation, because it happens over networks having maximum bandwidths that can be an order of magnitude (or more) lower than device-local HBM bandwidths. Thus, distributed analytical systems strive to minimize data exchanges and maximize local computations. Shuffle and broadcast are the most common kinds of data exchange operations. In shuffle, each process first applies the same partitioning function (based on one or more keys/attributes of the table) to partition its data. The shuffle operation then redistributes the partitions among the processes so that each process has all partitions corresponding to the same set of key values. In broadcast, each process scatters its part of the distributed table to all other processes. At the end of the operation, the entire table is replicated in all processes. While shuffle redistributes data while keeping the total data across all participants fixed, broadcast replicates data thereby increasing the total data size across all participants. Broadcasts are useful for small tables since they can enable local joins without needing the other table to be partitioned on the join keys. Generating optimal query plans for distributed execution requires knowledge of how the data is partitioned across the processes [8, 11, 66, 109]. For example, a join between two partitioned tables could proceed in one of following ways: (1) If both are partitioned on the join keys, then join locally; (2) Broadcast one of the tables and then join locally; (3) Partition and shuffle both tables and then join locally.

2.3 Collective Communications Library

The open-source NVIDIA collective communication library (NCCL) provides a variety of communication primitives between multiple GPUs. These include collective communication involving all participating GPUs (e.g., all-reduce) as well as point-to-point (p2p) communication for send and receive operations. We use NCCL primitives to implement the data exchange operations in TQP. This approach has the following advantages.

- NCCL transparently handles various available interconnect technologies for GPU-GPU communications, such as NVLink, GPUDirect RDMA over InfiniBand (IB), TCP/IP over Ethernet, etc. These interconnects can be either intra-node (e.g., NVLink connects GPUs local to a VM) or inter-nodes (e.g., RDMA over IB connects remote GPUs not residing on the same node).
- NCCL optimizes data routing for the network topology and chooses different routing algorithms for the underlying platform as part of its bootstrapping process.
- NCCL’s APIs are supported by both NVIDIA and AMD. The AMD implementation is named RCCL (ROCm Communication Collectives Library) [4], but the APIs and semantics remain the same. Thus, adopting NCCL enables ease of portability across vendors, along with enjoying vendor-maintained improvements for new hardware generations, which is similar to TQP’s motivation of leveraging PyTorch’s API for implementing SQL operators.

We use a subset of NCCL operations to implement data exchange operations in TQP. This includes collective and p2p operations, and group calls (to optimize routing and reduce kernel invocations).

The reason why we need to use p2p operations is due to a mismatch between NCCL collective APIs, which were developed primarily for ML applications, and what is needed to implement data exchange for distributed query execution. For example, collective operations assume equal-sized data at each node (as is the case for ML workloads). For SQL queries, this is restrictive since partitioned data is not always perfectly balanced. This can happen both due to skew in data distributions and having different selectivities of filter operations at different nodes. One workaround could be to use data padding to make the data sizes equal, but that would involve a waste of GPU HBM capacity as well as network bandwidth.

To solve this mismatch between collective primitives, and distributed SQL processing, one can build them using the p2p NCCL operations `ncclSend` and `ncclRecv`. These enable the construction of more complex operations such as shuffle², and also allow each send-receive pair to have a different size, thereby offering more flexibility. We implement the shuffle operation among N GPUs using N^2 (including self-transfers) `ncclSend` and N^2 `ncclRecv` enclosed within a `ncclGroupStart` and `ncclGroupEnd`. Algorithm 1 shows the pseudo-code of the shuffle executing at each GPU.

Algorithm 1 Shuffle @ i

```

1: ncclGroupStart
2: for  $j = 0$  to  $N - 1$  do
3:   ncclSend:  $i \rightarrow j$ 
4:   ncclRecv:  $i \leftarrow j$ 
5: end for
6: ncclGroupEnd

```

Algorithm 2 Broadcast @ i

```

1: ncclGroupStart
2: for  $j = 0$  to  $N - 1$  do
3:   ncclBroadcast: recv
   from  $j$  and send if  $j == i$ 
4: end for
5: ncclGroupEnd

```

For broadcast operations, a similar approach of using a set of p2p exchanges, although functionally sufficient, is not optimal in performance since that sends the same data packets to the same remote machine multiple times. For example, GPU 0 in machine 1 needs to send the same data twice to GPU 0 and 1 in machine 2 separately. Using the *one-to-all* broadcast operation in the communication library, e.g., `ncclBroadcast`, on the other hand, may avoid or reduce repeated transfers of the broadcasted data across machines [76, 83]. As we will show in Section 7.1, the impact is severe, particularly for multi-machine deployments where the network bandwidth between machines is about an order of magnitude lower with InfiniBand and about two orders of magnitude lower with Ethernet compared to intra-machine inter-GPU (e.g., NVLink) bandwidths. Hence, we use the one-to-all `ncclBroadcast` to implement the broadcast operation. Algorithm 2 shows the pseudo-code of the broadcast executing at each GPU. The for loop is needed since NCCL uses the same operation for both sending and receiving: when $i == j$ GPU i will send, while all GPUs participating in the same `ncclBroadcast` will be receiving from GPU i .

The shuffle and broadcast operations are preceded by an information exchange where the data sizes are exchanged between senders and receivers. This allows the receivers to allocate receive buffers accurately. Furthermore, since we exchange one column at a time, knowledge of the incoming data sizes allows in-place construction of a contiguous tensor for each column, as needed by TQP for

²Note that shuffle could be expressed as MPI `AllToAll` operation. However, NCCL doesn’t implement the `AllToAll` collective operation.

performing operations on it. We achieve this by determining a starting offset, for data incoming from each sender, in the contiguous receive buffer for the column at the recipient. This information exchange is very light-weight compared to the shuffle or broadcast because only N integers are sent from each GPU, where N is the total number of GPUs participating in the data exchange.

2.4 Distributed TQP

Let us now combine everything and discuss how we implemented distributed query processing in TQP. As a distributed processing model, we use data-parallel computing, with GPUs as the computational backends. The input data is partitioned between the GPUs in the distributed system and loaded in their device memories (HBM). We run a TQP process for each GPU, with all the processes launched as MPI (Message Passing Interface) [67] jobs by a distributed job runner (e.g., `mpirun`). In distributed mode, each TQP process computes using its assigned GPU on its local data, and it performs data exchange with other TQP processes, when necessary, by moving data directly between GPU HBMs wherever possible. To achieve this, we modified the TQP compiler stack to automatically inject data exchange operations (implemented as described in the previous section) into their compiled tensor programs before group by, join operations, or for final aggregation.

Note that: (1) each TQP process executes the exact same tensor program, the only difference is that each GPU reads a different input data partition; (2) data exchange operations are managed by the processes themselves and not orchestrated by a separate “driver” process; (3) fault tolerance is based on re-execution (since queries run at interactive speed). *This is closer to how ML training runs are executed, rather than how distributed analytical systems work.* Fundamentally, we are advocating not only for leveraging ML frameworks (i.e., PyTorch and NCCL) for targeting hardware accelerators and fast network interconnect, but also that we can embrace the same computational model used for ML training.

3 ANALYTICAL PERFORMANCE MODELS

In this section, we will describe performance models that facilitate our analysis of shuffle and broadcast operations on multiple GPUs spanning one or more machines. The goal of these analyses is twofold—to gain insights about the performance scalability of data exchange operations, and to extrapolate performance impact with different interconnects, larger cluster sizes, and next-generation network technologies that are not yet available.

3.1 Model Description/Assumptions

Table 1: Notations.

k	Number of GPUs per machine (or node, VM)
V	Number of machines (or nodes, VMs)
N	Total number of GPUs = $k \times V$
B_g	Unidirectional inter-GPU bandwidth within each machine
B_n	Unidirectional network bandwidth at each machine
S	Total dataset size processed by the GPU cluster.
G_{ij}	The j -th GPU in the i -th machine.
$m_{ij \rightarrow pq}$	The message sent from G_{ij} to G_{pq} .

Table 1 describes the notations that we use in this paper. To begin with, we focus on the case where we exchange a large amount

of data in the shuffle and broadcast operations because frequent latency-bound small-sized data transfers are not common or expensive in databases. Later, we also discuss how we adapt our models for small message sizes. Moreover, we only model the data exchange among the GPUs—metadata exchanges prior to the shuffle and broadcast (Section 2.3) are ignored.

- The system has V machines and each machine has k GPUs. Machines are connected by the network (e.g., Ethernet, RDMA). Each node can send and receive data to/from any other nodes. Each node has outbound/inbound network bandwidth B_n , hence each GPU has a share of B_n/k network bandwidth.
- Within each machine, there are also pairwise connections between GPUs. Each GPU can send data to all other GPUs within the same machine (i.e., local peers) at an aggregated rate of B_g .
- The total dataset is S in size. It is evenly distributed across N GPUs, making the size of each GPU’s work set $S/(Vk)$. Later in this section, we will model the effect of the data skew without assuming uniform distribution.
- A *message* is what a GPU sends to another GPU during a shuffle or broadcast operation.
- We assume that intra-node data transfer and inter-node data transfer can happen concurrently.
- We define the throughput of the operation as the total dataset size S divided by the total time T used to finish the operation.

3.2 Broadcast Scalability

We model the broadcast by assuming it uses a ring-based algorithm, which is true for throughput-optimized applications [42]. To address the discrepancy in bandwidth between a GPU’s intra-VM link (300-450 GB/sec) and its network (6.25-50 GB/sec), NCCL forms multiple rings, each of which connects all GPUs and spans both inter- and intra-node links. For example, in our H100+IB cluster (Table 2), every two rings share every two NICs per node, and in total 8 rings are formed. The rings become the most efficient when $B_n = B_g$, which means neither the inter- nor intra-VM interconnect is underutilized. This approach essentially obliterates the heterogeneity of interconnects. Since the ring-based broadcast proceeds in $(N - 1)$ steps, and in each step, S/N data are transferred over each hop of the ring, we can calculate the broadcast time as follows.

$$T_{\text{broadcast}} = (N - 1) \frac{S/N}{\min(B_n, B_g)}, \text{ for } V > 1.$$

S/N is the message size in the broadcast, which is the same as that of each GPU’s work set. The throughput of the ring is bound by the minimum of inter-VM and intra-VM interconnects, i.e., $\min(B_n, B_g)$. The throughput of broadcast can be calculated as

$$\text{Thpt}_{\text{broadcast}} = \frac{N}{N - 1} \min(B_n, B_g), \text{ for } V > 1. \quad (1)$$

When $V = 1$, since only the intra-VM links matter, the throughput becomes $\frac{N}{N-1} B_g$. The above equation implies that when we add more nodes to the system ($V \uparrow$), the broadcast throughput will decrease. In other words, the broadcast operation does not scale with the number of nodes. Figure 2a shows the model-predicted throughput when we increase the number of nodes for different network bandwidths (B_n). The throughput decreases with the number of nodes and eventually converges. Notice how 800 GB/sec network

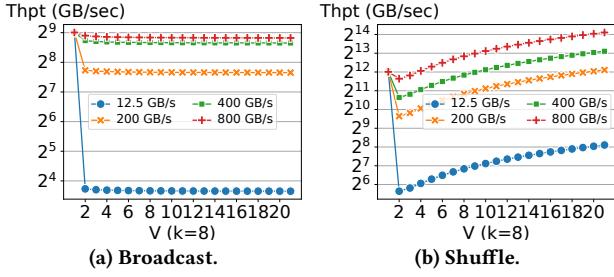


Figure 2: Model-predicted throughput

will not give much better broadcast performance than a 400 GB/sec because the intra-VM interconnects become the bottleneck.

3.3 Shuffle Scalability

The biggest difference between shuffle and broadcast is that the point-to-point message size in the shuffle operation is $S/(V^k)^2$ because each pair of nclSend/Recv primitives deals with only $1/(V^k)$ of the work set of a GPU whose size is $S/(V^k)$. Based on common cluster configurations, we assume that the local sends are faster than the remote sends. Therefore, the time of the shuffle operation can be calculated as

$$T = \frac{S(V-1)k^2}{(V^k)^2 B_n} = \frac{S(V-1)}{V^2 B_n}.$$

The throughput of the shuffle operation is therefore

$$\text{Thpt}_{\text{shuffle}} = \frac{V^2}{V-1} B_n, \text{ where } V \geq 2. \quad (2)$$

When $V = 1$, $\text{Thpt}_{\text{shuffle}} = \frac{N^2}{N-1} B_g$. For $V > 1$, $\text{Thpt}_{\text{shuffle}}$ is an increasing function of V , which means that with more nodes, the shuffle will become more efficient. Compared with $\text{Thpt}_{\text{broadcast}}$, shuffle is almost V times more efficient than the broadcast. Figure 2b shows the model-predicted throughput when we increase the number of nodes for different network bandwidths (B_n). For slower networks, the throughput significantly drops when V goes from 1 to 2. However, for $V \geq 2$, the throughput of shuffle consistently increases with the number of machines. Moreover, the throughput grows proportionally with the network bandwidth B_n . Although theoretically, the shuffle becomes more efficient with more nodes, the cost of partitioning the data and transmitting more network package headers could increase with more nodes.

3.4 Shuffle vs. Broadcast Competitiveness

To join the table R and table S , we can either broadcast R (assuming R is the smaller table) or shuffle both R and S . We represent the size of a table as $|\cdot|$. The time T_b for broadcasting R and the time T_s for shuffling both are then:

$$T_b = (N-1) \frac{|R|}{NB_n}, \quad T_s = \frac{V-1}{V^2 B_n} (|R| + |S|).$$

For $T_b < T_s$, we need:

$$\frac{|S|}{|R|} > \frac{N-1}{N-k} \cdot V - 1. \quad (3)$$

For $V = 1$, the condition for the broadcast-based join to outperform the shuffle-based join is $|S|/|R| > N - 1$. If we assume a fixed

k , then more GPUs make shuffle more favorable. On the other hand, when V is small and $|S| \gg |R|$, broadcast is preferred.

3.5 Modeling Skew

Data skew is omnipresent in database workloads, affecting not only local processing per GPU but also data exchange. We model how data skew could affect the broadcast and shuffle operations.

3.5.1 Broadcast. As mentioned in Section 3.2, NCCL form rings to unify the bandwidth profiles of different links as much as possible. In a ring, messages can be pipelined, which means a GPU can stream out the message to the next hop while it is being received. This brings the benefit that broadcast with a skewed initial data placement does not cause certain links to be idle or under-utilized. In conclusion, having data skew will *not* affect the broadcast as long as the broadcast is bandwidth-bound.

3.5.2 Shuffle. The effect of skew on the shuffle performance depends on the implementation of the communication library. An ideal implementation should be able to monitor the traffic of each NIC and leverage the high-bandwidth intra-node interconnect (e.g., NVLink) to route messages through idle local NICs. Consequently, each GPU can utilize all NICs of its residing node to send/receive messages to/from the network. This implies that data skew should only be visible on a per-node level instead of a per-GPU level. The total data sent and received through the network on any node i is

$$S_i = \sum_{j \in [0, k)} \sum_{p \in [0, V) \setminus \{i\}} \sum_{q \in [0, k)} m_{ij \rightarrow pq} \text{ (Send)} \\ R_i = \sum_{j \in [0, k)} \sum_{p \in [0, V) \setminus \{i\}} \sum_{q \in [0, k)} m_{pq \rightarrow ij} \text{ (Receive).}$$

The time of the shuffle operation is therefore

$$T_{\text{shuffle}} = \max\{S_0, \dots, S_{V-1}, R_0, \dots, R_{V-1}\} / B_n.$$

For brevity, we ignore potential data skew within the intra-node interconnect domain; however, it should be fairly easy to extend the model to account for it. In contrast to broadcast, data skew does have an impact on the shuffle, even in the ideal case. NCCL implementation's PXN [70] optimization (enabled by default) brings performance closer to the ideal, but falls short due to currently being sender-side only and lacking dynamic message scheduling [55].

3.6 Small Message Sizes

So far, we have been assuming that the efficiency of transferring a message (i.e., B_n, B_g) is independent of the message size m , and therefore the throughput of the shuffle and broadcast is also agnostic to message sizes. However, we observe in experiments that the message size does have an impact on performance of both. To address this, we extend our model by parameterizing B_n and B_g with the message size m . We follow the Hockney model [101] and assume the time to send a message via a link as $t = L + c \cdot m$, where c corresponds to the time gap between sending each byte and L is the latency. Therefore, we can represent B_n and B_g as

$$B_n(m) = \frac{m}{L_n + c_n \cdot m}, \quad B_g(m) = \frac{m}{L_g + c_g \cdot m}.$$

To find the model parameters c_n, c_g, L_n, L_g , we can fit our analytical models against our experiment measurements. Due to the

completely different implementations, shuffle and broadcast are fitted separately using the results from $V = 2$.

4 TESTBED AND WORKLOADS

4.1 Cluster Configuration

For this study, we use three clusters of multi-GPU Azure Virtual Machines (VMs). These clusters have GPUs from two vendors (NVIDIA, AMD) and use different interconnect technologies (NVLink, Infinity Fabric, Ethernet, InfiniBand) for communications between GPUs. Table 2 lists the cluster configurations. We report unidirectional bandwidths for all cases.

- **Intra-VM:** Communications between GPUs within the same VM can use high-bandwidth interconnects—NVIDIA NVLink for each VM in clusters 1 (A100) and 2 (H100), and AMD Infinity Fabric for each VM in cluster 3 (MI300X). For both the A100 and H100 clusters, the interconnect fabric is created with NVLinks and NVSwitches. In each VM of the A100 cluster, each GPU has 12 outgoing (and incoming) lanes, each supporting a bandwidth of up to 25 GB/sec, providing an aggregate max. outgoing (and incoming) bandwidth of 300 GB/sec. For H100, this increases to 18 outgoing (and incoming) lanes with an aggregate max. unidirectional bandwidth of 450 GB/sec. In the MI300X cluster, each GPU has 7 lanes, each providing up to 128 GB/sec bidirectional bandwidth, resulting in an aggregate max. unidirectional bandwidth of $7 \times 128/2 = 448$ GB/sec.
- **Inter-VM:** All VMs have an Ethernet NIC providing connectivity to other VMs. Additionally, every VM in the H100 and MI300X clusters has 8 Mellanox NICs, one per GPU, each supporting InfiniBand 4x NDR (Next-Generation Rate) connectivity of up to 400 Gbits/sec.

4.2 Profiling Setup

We report the average execution time of queries, taken over 10 runs, once each query’s input data is loaded into the HBM of all participating GPUs. We take measurements after a warm-up phase, in which each query has been run twice to warm up the device caches. Our analyses are based on the input, output, and intermediate data of the query execution fitting in the GPU HBM. This fits with our goal of demonstrating the potential speedup opportunities of multi-GPU clusters. This also represents scenarios of recurring queries, or when data loaded earlier into GPU memory may remain cached. We discuss the impact of data loading for cold runs in Section 7.4.

In the experiments, we break down the execution time into three parts: compute, shuffle, and broadcast. Compute time refers to the local execution time of each GPU. We measure the end-to-end query execution time, and for each shuffle/broadcast operation, we insert barriers before and after to obtain its time³. We calculate the compute time by subtracting the communication time from the query execution time. We use vendor-specific utilities (e.g., nvidia-smi) to monitor GPU memory occupancy during query runs.

4.3 Workloads

We use the 22 queries of the TPC-H benchmark in our study, on both default (uniform) and skewed data. We run these queries on the TPC-H dataset, which has a mostly uniform distribution of

keys, for Scale Factors (SF) of 1000 and 3000. To study the impact of data skew, we generate skewed data using the data generator for the JCC-H [10] dataset, but use the TPC-H queries for a close comparison with TPC-H query runs.

We pre-partition input tables based on keys to reduce shuffles of leaf-level inputs. Since the two largest tables, `lineitem` and `orders`, are joined in queries Q3, Q4, Q5, Q7, Q8, Q9, Q10, Q12, Q18, and Q21, we partition `lineitem` by the foreign key, `l_orderkey`, so that the distributed join can be performed locally on each GPU without first requiring a data exchange operation. For `partsupp`, we use `ps_partkey`, and for the remaining tables, their primary keys, as the partitioning keys. Some queries, e.g., Q1 and Q6, do not need or use key-based input partitioning.

4.4 TQP Setup

Selection of the optimal exchange operation requires knowledge of table cardinalities (Section 3.4), as well as how data is pre-partitioned (Section 4.3). Since in this work, we aim at showing the speedup potential with multi-GPU acceleration, once distributed TQP generates the tensor programs, we further optimize them manually (e.g., changing the data exchange operation, or the join ordering). We leave the integration with a statistic-aware distributed query optimizer as part of our future work.

The data exchanges involved both leaf tables (query inputs) and intermediate results. In these counts, we exclude the final gather operation for collecting partial results from the individual GPUs. Having partitioned input tables helps to reduce the number of shuffles. A different partitioning scheme would lead to a different number of data exchange operations in the resulting query plans: we discuss an example in Section 7.3.

5 DATA EXCHANGE PERFORMANCE

We develop microbenchmarks to measure the throughput of shuffle and broadcast on our platforms under various settings. These include varying the number of machines, data skew, and message sizes. In addition, we validate our models with the measured performance assuming that the message sizes are known.

Skew-free case. To demonstrate the power of our model, we compare our model-predicted throughput for $V = 4$ with the measured throughput in Figure 3. To get the prediction, we first find the parameters L_n, c_n (see Section 6.5) from $V = 2$ by fitting. The results show that our models have strong predictive power for both shuffle and broadcast, for varying message sizes, for different GPUs, and for different network types. The implication of this is that we can use our model to accurately predict the data exchange performance of real query workloads given a cluster configuration. The prediction provides insights into whether the user should scale out the cluster to gain more performance. As we will see in Section 6.3, although scaling out brings more parallelism, it also makes data exchange less efficient due to smaller message sizes.

Skew case. To validate our models in the skew case, we introduce skew to the initial data placement. Assuming GPUs G_0, G_1, \dots, G_{N-1} have $x, x+fx, \dots, x+(N-1)fx$ data, respectively. Here, we call f the “skew gradient”. Varying f produces different levels of skewness, and when $f = 0$, there is no skew. We fix the total dataset size to be $S = N \times 1$ GiB; therefore, for each f , we need to find out x . For

³We don’t explicitly measure the reduction operations (`ncclAllReduce`) since these have negligible overhead.

Table 2: Cluster Configurations. Eth: Ethernet. IB: InfiniBand.

Cluster	GPU Type	HBM (GiB)	k	V	GPU Interconnect		CPU type	CPU Cores	CPU Mem (GiB)	Price/hour (USD)
					Intra-VM	Inter-VM				
1	NVIDIA A100	80	8	7	NVLink 300 GB/sec	Eth: 1×50 Gbits/sec	AMD EPYC 7V12	96	1800	32.77*
2	NVIDIA H100	79.6		5	NVLink 450 GB/sec	Eth: 1×100 Gbits/sec IB: 8×400 Gbits/sec	Intel Xeon Platinum 8480C		1900	98.32
3	AMD MI300X	191.5		4	Infinity Fabric 448 GB/sec	Eth: 1×100 Gbits/sec IB: 8×400 Gbits/sec	Intel Xeon Platinum 8480C		1850	63.6

* This is the price for 8×200 Gbits/sec Infiniband. The Eth version, where we run our experiment, is not publicly listed.

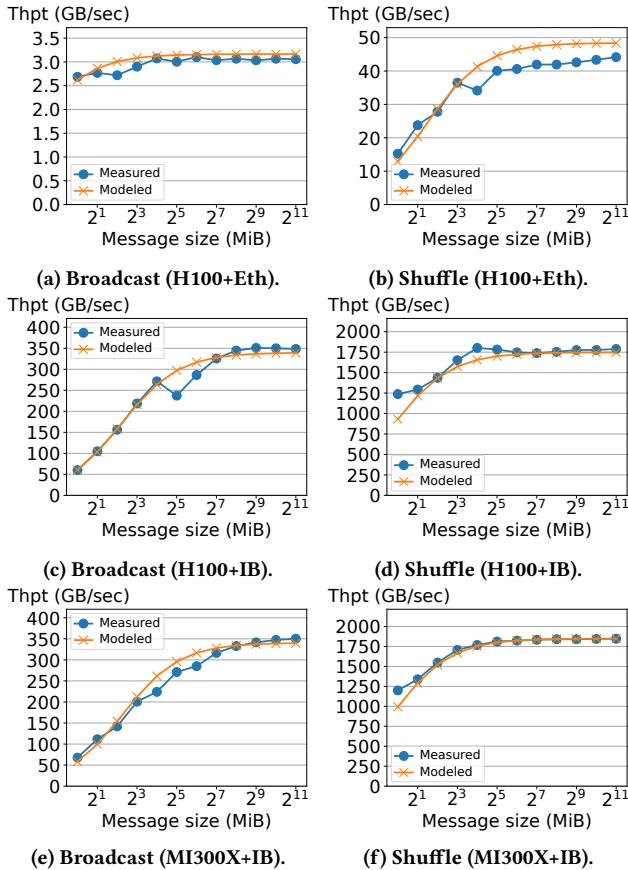


Figure 3: Model validation for the skew-free case. (V=4)

example, if $f = 1$, $V = 4$, $k = 8$, then the first GPU will have around 62 MiB data, whereas the last GPU will have around 1.94 GiB data. In contrast, if $f = 0$, every GPU will have 1 GiB of data.

We measure the performance of broadcast for different V and f . The result (Figure 4) shows that the throughput of broadcast is not affected by the data skew, which agrees with our model.

In the shuffle case, on top of the initial data placement, we let each GPU send the same amount of data to all of its receivers. In this case, the last node sends the largest amount of data, and the first node receives the largest amount of data. Figure 5a-5c shows our modeled shuffle throughput in this case in comparison with the actually measured throughput. We find the B_n parameter from fitting the $f = 0$ data point. The result shows that our model also captures the effect of skew very well. The result also implies that

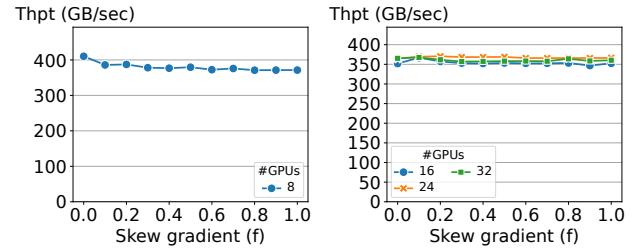


Figure 4: Broadcast + data skew.

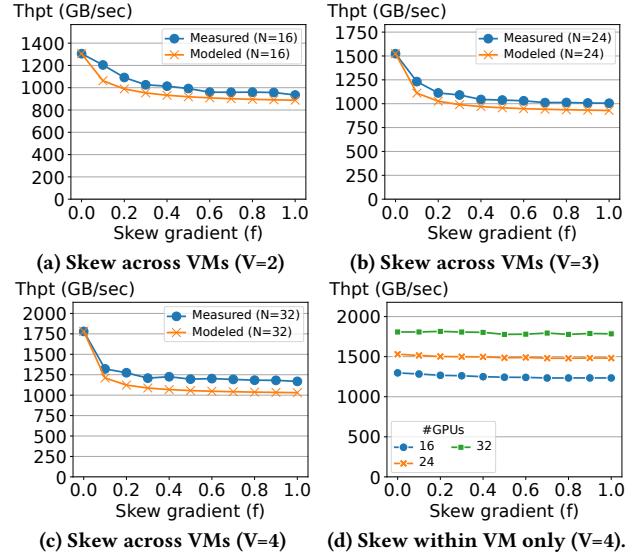


Figure 5: Shuffle + data skew.

the performance of shuffle degrades noticeably with the existence of skew across VMs. According to our model, if the skew only exists across GPUs but not VMs, the skew does not have an effect. We further validate this claim by letting each GPU in each VM contain $x, x + fx, \dots, x + 7fx$ data so that there is only skew intra-VM but not inter-VM. Results in Figure 5d show that the skew in this case does not affect the shuffle performance.

6 TPC-H PERFORMANCE ANALYSIS

In this section, we discuss the times for TPC-H queries on 1 TB and 3 TB datasets with distributed TQP. We analyze the results from various angles, including run times (Section 6.1), time breakdown (Section 6.2), price performance (Section 6.4), data exchange (Section 6.5), and memory utilization (Section 6.6). Furthermore,

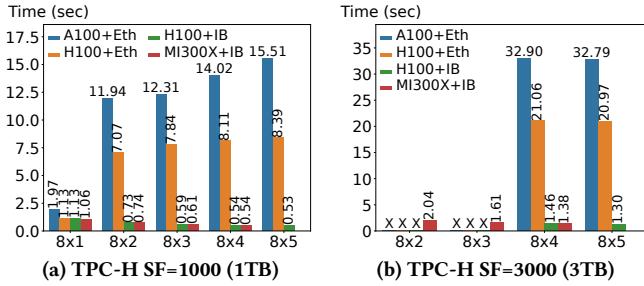


Figure 6: Workload performance (sum of run times for 22 queries) with different cluster configurations.

based on existing results, we project the performance for future networks and scale out with more VMs (Section 6.3). The goal is to characterize the execution of a classic SQL workload and offer insights into the efficiency and bottlenecks of multi-GPU clusters.

6.1 Workload performance

Figure 6 shows overall workload performance, which is the sum of the warm run times over all 22 TPC-H queries. We only show cluster configurations where all queries can be run. For SF=1000, this was possible on all cluster configurations, but SF=3000 required $V \geq 4$ for A100 and H100 VMs, and $V \geq 2$ for MI300X VMs.

We see that all queries with SF=1000 could be run in 1.1 seconds using 8 GPUs in a single VM. With multiple VMs and more GPUs, further speedups are possible. With 40 GPUs in 5 VMs with IB interconnects, the total run time reduces to 0.53 seconds for SF=1000 and 1.3 seconds for SF=3000, representing more than two orders of magnitude in speedup over published numbers for these scale factors using CPU-only machines [36, 37]. Interestingly, the run times are comparable between H100 and MI300X machines, even though they are from different vendors.

The above speedups with multiple VMs are possible only with high-bandwidth networks, such as the IB networks that we use. With Ethernet connectivity, the times are significantly larger for multi-VM configurations, e.g., 15.51 seconds for A100 and 8.39 seconds for H100 at $V = 5$. We omit the numbers for MI300X with Ethernet from the figure since the network configuration is similar to that of the H100 cluster, and the single-VM performances are also similar. Multi-VM configurations without high-bandwidth interconnects only increase run time and costs, and thus are not an efficient setup for these workloads.

6.2 Time breakdown

Figure 7 shows the breakdown of workload run time (sum of run times of the 22 queries) into compute, shuffle, and broadcast components. For the multi-VM InfiniBand configurations, we only show breakdowns for the H100 cluster since the network configuration and total run times for the MI300X cluster are similar.

If only a single VM is used (8x1 configuration), then the compute time dominates. As the number of VMs (V) increases, shuffle and broadcast times become major contributors to overall time. This is because of the relatively low inter-VM network bandwidths compared to intra-VM NVLink bandwidths—two orders of magnitude lower for Ethernet and one order of magnitude lower for

InfiniBand—which severely impact the performance of data exchange operations. For 5 VMs (8x5 configuration), they contribute 55.1%, 92.3%, and 94.8% of times for H100+IB, H100+Eth, and A100+Eth configurations. The significantly lower-bandwidth Ethernet NICs on the A100 VMs cause data exchanges to take the longest time on the A100 cluster. The time breakdowns highlight the critical importance of network bandwidth for the scale-out performance of distributed GPU-based analytical query processing.

Between shuffle and broadcast, the latter dominates as we scale out by increasing V . For example, for the A100+Eth, shuffle and broadcast contribute 34.5% and 61% respectively to the overall time for $V = 7$, but 36.4% and 55.4% for $V = 2$. For H100+IB configurations, they contribute 11.3% and 38.7% for $V = 5$, and 10.2% and 16.5% for $V = 2$. This trend is due to the poor scalability of broadcast with V compared to shuffle, as we discussed in Section 3.

Figure 8 shows a breakdown of total run times for TPC-H SF=3000, on the A100 and H100 clusters for values of $V \geq 4$ where all 22 queries were completed. Since the MI300X GPUs have more HBM, we can run all 22 queries on the MI300X cluster for $V \geq 2$. Similar to SF=1000 breakdowns, these breakdowns also show the very significant impact of network performance on shuffles and broadcasts, and consequently, on overall workload performance.

We also note the run time scales sub-linearly with the increase in scale factor. For the H100+IB configuration, the run time increased by 2.7 \times and 2.5 \times for $V = 4$ and $V = 5$ respectively, compared to the corresponding times for SF=1000.

6.3 Performance Projection

We project the performance of TPC-H 1TB for an increasing number of machines (V) of Cluster 2 with IB, using our analytical scalability models for shuffle and broadcast. One of the most important goals of our modeling effort in Section 3 is to help users build an expectation on how the TPC-H performance will change before they decide to scale out. We discuss two different methods of predicting the TPC-H performance, “best-accuracy” and “best-effort”. Both utilize our analytical models in the same way, and the only difference is how they predict the compute time. The “best-accuracy” method predicts the compute time by fitting the five measurements of compute time from $V = 1$ to 5 into a power model, $\text{Compute} = a \times N^b$ ($-1 < b < 0$), which captures the sub-linearity. However, this requires the user to run the TPC-H on more than one machine, which is potentially costly and unavailable. To solve this, the “best-effort” method only uses the compute time at $V = 1$ and assumes a perfect linear scaling of the compute time with respect to V .

To project the shuffle and broadcast time in both methods, we leverage (1) the workset size of each shuffle and broadcast and (2) the c_n, c_g, L_n, L_g constants obtained from the microbenchmarks ($V = 1, 2$), which are essential to account for small message sizes. We argue that these are reasonable and accessible inputs to our models: (1) can be obtained by running the queries for $V = 1$, and (2) can be obtained from fact sheets or published literature. **(Projection I)** This vanilla approach ignores the effect of message sizes and uses the B_n and B_g in Table 2 to calculate the throughput of shuffle and broadcast according to Equation 1 and 2. Since in real systems the theoretical peak bandwidth of a link cannot be attained, we normalize our predicted performance by making it

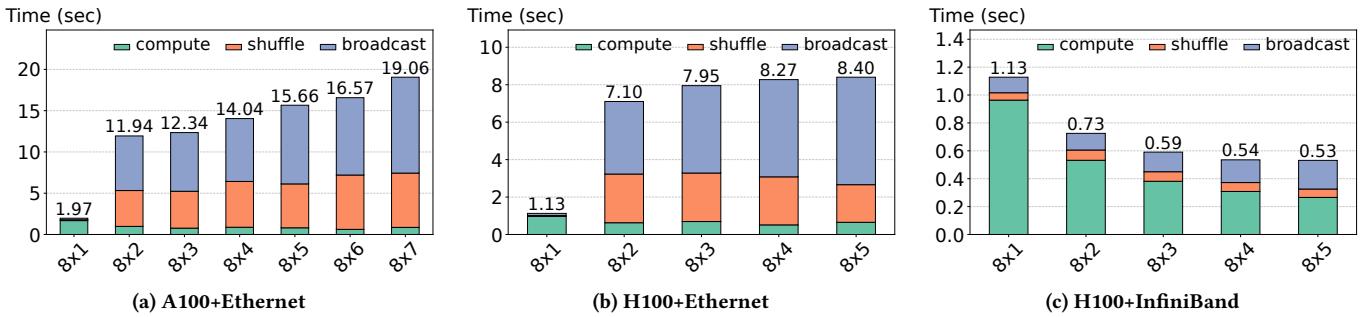


Figure 7: 22-query Total Time Breakdown. TPC-H, SF=1000. (Each bar from bottom up: compute, shuffle, and broadcast.)

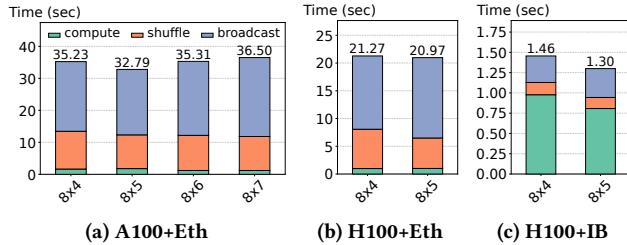


Figure 8: 22-query Total Time Breakdown. TPC-H, SF=3000.

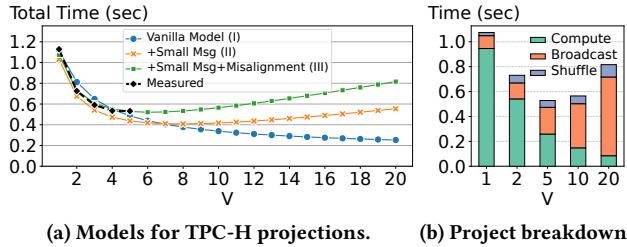


Figure 9: Project TPC-H performance with our models.

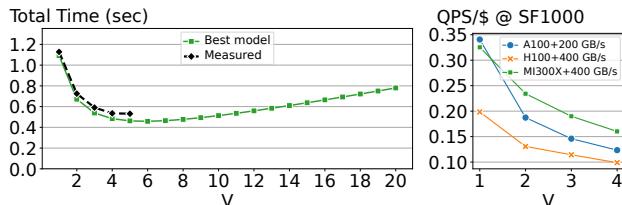


Figure 10: Project TPC-H from V = 1. Figure 11: QPS/USD.

agree with the measured performance at $V = 1$. **(Projection II)** This approach (“+Small Msg”) takes the average message size of each shuffle and broadcast as input to $B_n(m)$ and $B_g(m)$ and then applies Equation 1 and 2. **(Projection III)** This approach (“+Small Msg+Misalignment”) considers yet another influential factor that is often seen in real-world workloads, namely misaligned start addresses of send/receive buffers. As shown in Figure 12, if the starting addresses of the send or receive buffer are not aligned at 16 bytes, the performance will degenerate substantially in some cases.

Figure 9a shows the predicted time of TPC-H using “best-accuracy”. Due to a lack of fine-grained message-size-based modeling, **Projection I** does not capture the trend of the performance well and is expected to deviate further away as V continues to increase. However, **Projection I** does reflect how the performance would look if

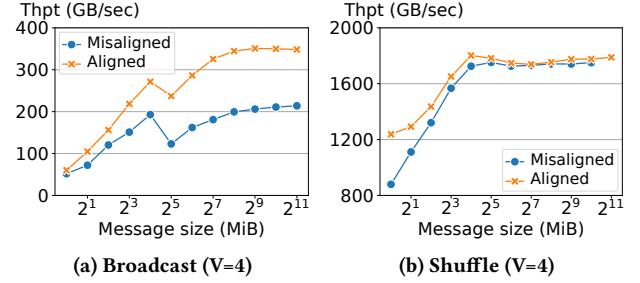


Figure 12: Effect of misalignment on shuffle and broadcast.

the data communication is less sensitive to message sizes. Another merit is that it also works for the interconnects that we have not studied with microbenchmarks. Although **Projection II** captures the trend better than the vanilla approach, it consistently underestimates performance due to its ignorance of misalignment. The best approach (**Projection III**) very accurately models the TPC-H performance. According to it, adding more machines will not improve the performance for $V > 6$. Figure 9b shows how the performance breakdown looks when increasing V according to **Projection III**. As compute time drops with more machines, broadcast time increases consistently due to two factors. First, broadcast becomes less efficient with more machines according to Equation 1. Second, messages become smaller and cannot efficiently utilize the links. Unlike broadcast, shuffle benefits from more machines (Equation 2); however, according to our prediction, the shuffle performance will eventually be dragged down by the small message sizes.

Figure 10 shows the projection by the “best-effort” (using **Projection III**). It slightly underestimates the performance because the compute cannot scale perfectly linearly, otherwise it provides a very close projection to “best-accuracy” (using **Projection III**). This means even with the performance of $V = 1$, we can reliably predict the performance of TPC-H for a large number of machines!

6.4 Price-Performance

Although the workload performance is lower on the A100 VMs compared to that on the H100 and MI300X VMs, they have an advantage from a price-performance perspective. Table 2 lists the hourly price of one machine of different clusters. Currently (as of this writing, March 2025), the Azure pay-as-you-go pricing [68] for single eight-GPU VMs is around 3 \times and 1.94 \times higher for the H100 and MI300X

VMs compared to that for the A100 VMs. Figure 11 shows the price performance of each cluster when running SF=1000. The price performance is calculated as the “query-per-second” (QPS) per dollar. Since our A100 cluster does not have InfiniBand connectivity between VMs, we use the performance projections (vanilla approach) discussed in Section 6.3 to estimate performance for that, assuming 8×200 Gbits/sec InfiniBand NICs per A100 VM. The result shows that runs for TPC-H SF=1000 on a single eight-GPU A100 VM have 1.72× and 1.05× better QPS/\$ respectively compared to the QPS/\$ on the H100 and MI300X VMs. For multi-VM price-performance, we find that for $V = 4$, A100+8×200 Gbits/sec has 1.25× better and 0.77× worse QPS/\$ compared to 4 VMs of H100 and MI300X respectively, both of which use 8×400 Gbits/sec InfiniBand NICs per VM. We can conclude that MI300X VMs are priced competitively, as they have performance similar to that of the H100 VMs, but are currently ~35% less expensive than them.

6.5 Message sizes for data exchange

Figure 13 shows the distribution of inter-GPU message sizes for broadcasts and shuffles, over all the 22 queries, along with the 80th percentile values. For these workloads, most of the messages are at most a few hundred MiBs and usually much smaller, e.g., for $V = 4$, 80% of shuffles and broadcasts are smaller than 7 MiB and 36 MiB, respectively for SF=1000, and 21 MiB and 108 MiB for SF=3000. The largest message sizes occur for $V = 1$. These are 179 MiB and 191 MiB for shuffles and broadcasts at SF=1000. The message sizes increase with dataset size, e.g., up to 537 MiB for shuffles in Q22 at SF=3000 ($V = 1$). They also increase with data skew, which we will discuss further in Section 7.2.

For both message types, increasing V reduces message sizes, but the reduction is larger for shuffles than for broadcasts due to their inverse quadratic scaling with $N = V \times k$ rather than the inverse linear scaling for broadcasts (as we discussed in Section 3). Although theoretically, this should result in lower network overheads with increasing V , smaller messages utilize the bandwidth less effectively, as we showed in Section 5, due to protocol and kernel launch overheads, thereby preventing a proportional reduction in per-message network overheads with scale out.

6.6 Memory Occupancy

Compared to main memory in high-end servers, single GPUs have much smaller HBM capacity, thereby limiting the largest dataset size that can be kept resident on each GPU. Using multiple GPUs per machine alleviates, but does not remove, this bottleneck. In our setup, we run queries with all data resident in the GPU HBMs, and do not spill or otherwise transfer data between the GPUs and main/external memory or storage. A query execution will fail if the peak memory consumption exceeds the HBM capacity at runtime. While all queries for SF=1000 finish with $V \geq 1$ on any cluster, SF=3000 requires $V \geq 4$ for the A100/H100 clusters, but $V \geq 2$ is sufficient for the MI300X cluster due to the larger HBM capacity.

Figure 14 shows the distributions, along with the 80th percentiles, of peak per-GPU HBM occupancy on the A100/H100 GPUs during runs of all the 22 queries. The peak occupancy is affected by the size of the input dataset and intermediate results, and the space needed by GPU kernels and the runtime scheduler. The peak increases with

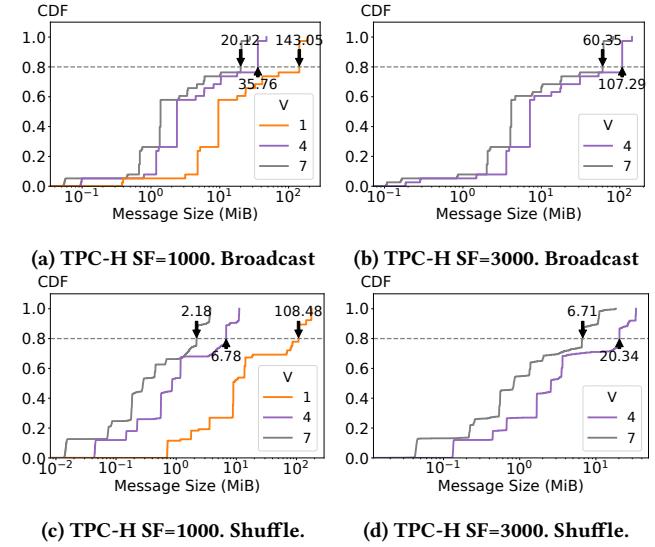


Figure 13: Message size distribution for all running queries. Max. message size decreases as V increases.

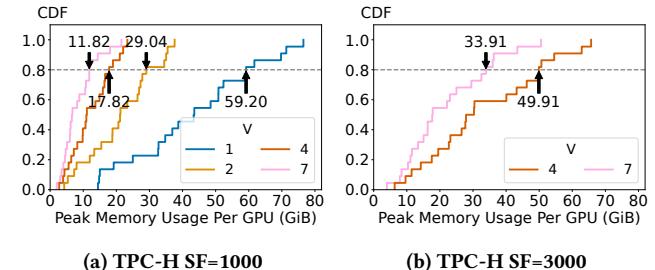


Figure 14: Distribution of peak GPU memory occupancy of all running queries. Occupancy decreases as V increases.

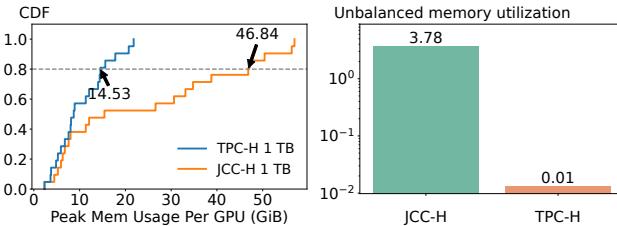
the scale factor, e.g., the 80th percentile increases from 17.8 GiB for SF=1000 to 49.8 GiB for SF=3000, and decreases with V , e.g., the 80th percentile for SF=1000 decreases from 59.2 GiB for $V = 1$ to 11.8 GiB for $V = 7$, both due to the associated changes in the per-partition data size for each GPU.

One difference in memory consumption between single-GPU and multi-GPU query execution is that additional space is needed for the latter to create partitioned inputs for shuffle operations. This is not much in our setup since, for this workload, the shuffle message sizes are small (as we discussed in Section 6.5) and the tables do not have too many columns. Memory needs for partitioning can be reduced by partitioning one column at a time for shuffling and reusing the memory for the next column.

6.7 Comparison with other Databases

In this section, we compare our system with two popular database systems, DuckDB (CPU-based) and HeavyDB (GPU-based).

We run DuckDB v1.3.2 on Cluster 2 (Table 2), which has 96 CPU cores and 1900 GiB memory. For each query, we do 10 warmup runs, then measure the median runtime of 10 subsequent runs. For TPC-H SF=1000, DuckDB takes 121 seconds, two orders of magnitude slower than our result for $V = 1$ on Cluster 2. Assuming perfect scaling with V , DuckDB would still need at least 24.2 seconds (plus



(a) Peak memory utilization CDF. (b) Imbalance across GPUs.

Figure 15: Memory utilization for TPC-H and JCC-H (1 TB).

communication overheads) with $V = 5$ compared to 0.53 seconds in our case. For TPC-H SF=100, DuckDB spends 10.7 seconds. In contrast, our system spends 0.13 seconds using eight H100 GPUs in a single VM (0.59 seconds using a single H100 GPU). On a 32x-cheaper CPU-only VM with 64 vCPUs (VM type: D64s v5, 32 cores, \$3.072/hr), DuckDB needs 13.1 seconds for SF=100, thus allowing us a performance advantage of over two orders of magnitude, and a QPS/\$ advantage of over 3x for SF=100 using eight H100 GPUs.

For HeavyDB⁴, we use their published TPC-H performance numbers [34]. On a GH200 machine, which contains a single H100 GPU, HeavyDB takes 3.9 seconds (without Q21) for SF=100. Assuming perfect scaling, it would need at least 0.49 seconds with eight GPUs, leaving us with a performance advantage of at least 3.75x.

7 PERFORMANCE SENSITIVITY

We now discuss how naive broadcast implementations (Section 7.1), skewed data distributions (Section 7.2), sub-optimally partitioned input data (Section 7.3), and data access from host memory (Section 7.4) can impact the query performance.

7.1 Collective vs point-to-point broadcasts

In Section 2, we emphasized that broadcast needs to be done with a collective operation rather than with a set of point-to-point (p2p) operations for better performance. Here, we detail the impact of p2p-based broadcast on the performance of TPC-H. Both implementations produce similar performance at $V = 1$. However, the negative impact of p2p-based broadcast increases with V . The slowdown is 1.6x, 1.2x at $V = 2$, but 2.2x, 1.3x at $V = 5$ for H100+Eth and H100+IB respectively.

7.2 Uniform vs Skewed inputs

The TPC-H dataset has a largely uniform distribution of data values to table keys [10]. This leads to the partitioned table sizes on the different GPUs being largely similar. To test scenarios involving skewed data, we use the JCC-H dataset with the TPC-H queries.

Memory utilization and message sizes. Figure 15a shows the distribution of per-GPU peak memory occupancies of 21 queries⁵ for SF=1000 with $V = 5$. In contrast, we also show the distribution for these 21 queries on TPC-H for $V = 5$. The imbalance caused by the data skew results in more HBM being used on some GPUs, while some other GPUs' HBM remains under-utilized. The 80th percentile and the max. peak memory used are 46.8 and 56.9 GiB, respectively,

⁴We were unable to download the enterprise version of HeavyDB, and found that the available version was slower than published numbers.

⁵We exclude Q18 since it currently produces incorrect results for JCC-H.

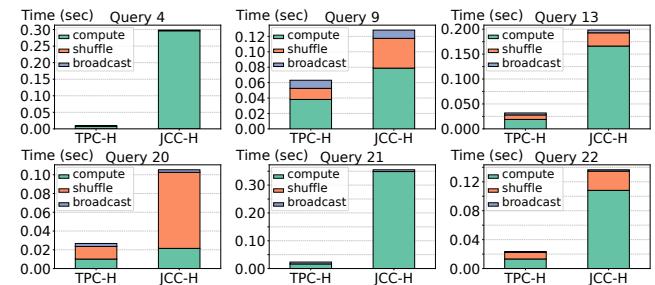


Figure 16: Time breakdown comparison. (V=5)

for JCC-H, but 14.5 GiB and 21.8 GiB for TPC-H 1TB. In Figure 15b, we show the imbalance of memory utilization among GPUs. For each query, we calculate the standard deviation of all GPUs' memory utilization and report the average of all queries. The results indicate that JCC-H causes a severely uneven workload distribution among GPUs. A query fails if the memory required exceeds the available HBM capacity on even a single GPU. The A100/H100 clusters need $V \geq 5$ to run these queries, while the MI300X cluster needs $V \geq 3$. In terms of the message size distribution (not shown), JCC-H 1 TB has a much longer tail than TPC-H 1 TB for the shuffle, whereas the two benchmarks have very similar distributions for the broadcast. The max. message size in broadcast operation for both benchmarks is 38.7 MiB. In contrast, the max. message size in the shuffle operation is 244 MiB and 10 MiB for JCC-H and TPC-H, respectively.

Per-query analysis. Several queries severely suffer from the skew introduced in JCC-H, as shown in Figure 16. The figure details where the time is spent in each benchmark. The two main contributing factors to slower execution are compute and shuffle. In Q4, the lineitem table is ill-partitioned based on 1_orderkey, leading to some GPUs processing almost 7x more data than others. Worse still, due to the imbalance, some GPUs need to build a hash table with around 450M keys, far exceeding the problem size that a hash join can efficiently handle [103]. In Q20, two GPUs in the same VM needs to send around 7x more data when shuffling lineitem to join with partsupp. Moreover, some GPUs also receive almost 11x more data than others due to a poor partition function. As discussed in Section 3 and Section 5, shuffle can be affected by such a skew, resulting in a sheer increase of shuffle time. Even after shuffling, the lineitem table is still distributed in an unbalanced manner, which causes the subsequent join to be slower. Q9 sees a significant increase in both compute and shuffle. The shuffle of partsupp takes longer due to the skew in initial data distribution and the partition function. The compute is slowed down by a series of joins that stress only some GPUs. In contrast, the broadcast is not affected by the skew even though the largest message size in the broadcast is almost 15x larger than the smallest.

DuckDB comparison. We run the same JCC-H benchmark (SF=1000) with DuckDB on a single machine of our Cluster 2. Similar to Section 6.7, we report the median time of warm runs. It takes DuckDB 108.34 seconds to finish 22 queries, 67x slower than our system (1.62 seconds, $V = 5$, excluding Q18⁵).

7.3 Partitioned vs non-partitioned inputs

Our performance results so far are for query runs on appropriately partitioned input data that reduces/avoids shuffles on large

input tables. Changing the partitioning scheme may require corresponding changes to the query plan for functional correctness and result in a different run time for the query. Here, we investigate its impact using Q12 as an example. This query filters lineitem by applying a predicate and joins with orders. Our default plan does not need any data exchange operations for this query since the input tables (lineitem, orders) are partitioned by their join keys. We consider the following alternative query plans when neither table is partitioned on these keys. **Pa**: Shuffle lineitem (filtered) on l_partkey and orders on o_partkey, then do the join. **Pb**: Broadcast lineitem (filtered), then join with orders.

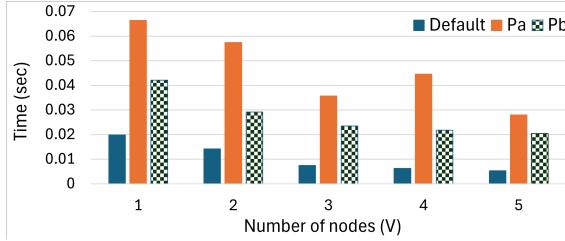


Figure 17: Q12 time using different plans (Cluster 2)

Figure 17 shows the run times of Q12 for TPC-H SF=1000 on the H100 cluster with a different number of nodes (V). The default plan, which assumes already-partitioned data on join keys and avoids data exchanges, was the best. Plan **Pa** was the worst, while **Pb** came second. As already discussed, broadcasts can be more efficient than shuffles, as is the case here. **Pa** gets more competitive with more nodes due to the faster (quadratic) reduction of message sizes compared to the linear reduction for broadcast (see Section 3).

7.4 Warm vs cold runs

The performance numbers that we presented so far are for warm runs with the input data already loaded in the GPU HBMs. For cold runs, data needs to be moved from the main memory over the CPU-GPU PCIe bus. On the H100 multi-GPU machines, this can be done with an aggregate bandwidth of 440 GB/s (8×55) over the 8 PCIe gen5 buses (theoretical max. of 8×63 GB/s). The total run time for the 22 queries for TPC-H 1TB using one H100 VM increases from 1.13s to 11.4s for cold runs, which is still more than $\times 7$ faster than CPU-only warm execution [37]. This is the worst-case scenario, as, in practice, some of the input columns may be cached from prior query executions, thereby not incurring the loading overhead. With multi-machine clusters, the data loading time drops linearly with the number of machines (V) assuming that the input data is uniformly partitioned and that the data loads on all machines are initiated in parallel. Other recent architectures [71, 72] replace the CPU-GPU PCIe with a high-bandwidth interconnect. Thus, like prior studies [47], we foresee data loading as not being the bottleneck anymore in the future. Additional overheads affecting cold run times are query compilation overheads, which are query dependent [32], but can be mitigated using a query plan cache.

8 RELATED WORK

Distributed analytical databases. There is a long history of research and development of distributed databases for data analytics [1, 6, 8, 24, 66, 79, 109]. All of these systems only use the CPUs

for data processing and many adopt the Massively Parallel Processing (MPP) paradigm for scalability. Our work adopts MPP to construct a *GPU-based* distributed analytical database.

Single-GPU query processing. Numerous works have studied different aspects of single-GPU-based query processing, for example, implementation and optimization [7, 26, 29, 32, 40, 41, 48, 49, 53, 58, 77, 81, 82, 88, 90, 92, 94, 103, 105], data placement and caching [106], fast interconnect technologies [61, 62], optimizing storage I/O [9, 95], data compression [35, 43, 89], system integration [45, 46, 51, 69], performance analysis and surveys [14, 47, 73, 80, 91, 108], and so on. These techniques are orthogonal to our work but are beneficial in improving the efficiency of individual GPUs in our system.

Multi-GPU query processing. Many works [30, 31, 63, 64, 74, 93] have studied the implementation of database operators on multiple GPUs. HetExchange [16] is a database execution model that can exploit parallelism across multi-core CPUs and multi-GPUs. Yuan et al. [107] uses multiple GPUs to address the CPU-GPU PCIe bottleneck. Yogatama et al. [104] designs a hybrid CPU and multi-GPU query engine. The above work either does not consider a multi-node GPU cluster or does not present full database systems that can complete large-scale TPC-H benchmarks. Our work studies the most general multi-GPU-multi-node case with different GPU models from different vendors and various network technologies. Some commercial systems [5, 25, 33, 78] also support data analytics on multi-GPU systems. Compared to them, we present an in-depth analysis of the TPC-H and JCC-H workloads in addition to end-to-end query runtime. In terms of system design, we advocate the use of ML-style processing with ML-driven high-performance libraries.

Multi-GPU communication. Many previous works [13, 22, 39, 44, 50, 87, 98] provide alternative communication libraries to NCCL and RCCL. Weingram et al. [100] compares the state-of-the-art collective communication libraries. Some works [38, 57, 75] evaluate GPU-interconnects and networks. Others [15, 27, 55, 56, 60, 85, 86, 99] optimize and/or model certain collective communication primitives, such as all-reduce. Universal Communication X (UCX) [87] and NVSHMEM [21] are other interconnect-agnostic frameworks for multi-GPU communications. To the best of our knowledge, we are the first to model and optimize the inter-GPU communication for database applications, show how these ML-oriented communication libraries can be used for databases, and demonstrate how much time inter-GPU communication takes up in query processing.

9 CONCLUSION

We present a distributed implementation of TQP leveraging group communication libraries to process tera-byte scale TPC-H workloads on multi-GPU clusters. Our approach allows for seamless portability across different GPU models (A100, H100, and MI300X) and network technologies (Ethernet, Infiniband, NVLink, Infinity Fabric). Our experimental evaluation shows that distributed TQP yields very competitive performance results for TPC-H 1 TB and 3 TB scale factors, up to two orders of magnitude faster than public CPU-server results. To better understand the performance, we develop analytical models for shuffle and broadcast operations, and conduct a detailed analysis of TPC-H workloads. We believe that this work unveils the potential of multi-GPU in the SQL analytics domain, and provides researchers and practitioners with valuable insights into how to build an efficient multi-GPU database system.

REFERENCES

[1] Josep Aguilar-Saborit, Raghu Ramakrishnan, Krish Srinivasan, Kevin Bock-srocker, Ioannis Alagiannis, Mahadevan Sankara, Moe Shafei, Jose Blakeley, Girish Dasarathy, Sumeet Dash, Lazar Davidovic, Maja Damjanic, Slobodan Djunic, Nemanja Djurkic, Charles Feddersen, Cesar Galindo-Legaria, Alan Halversen, Milana Kovacevic, Nikola Kicovic, Goran Lukic, Djordje Maksimovic, Ana Manic, Nikola Markovic, Bosko Mihic, Ugljesa Milic, Marko Milojevic, Tapas Nayak, Milan Potocnik, Milos Radic, Bozidar Radivojevic, Srikumar Rangarajan, Milan Ruzic, Milan Simic, Marko Sosic, Igor Stanko, Maja Stikic, Sasa Stanojkov, Vukasin Stefanovic, Milos Sukovic, Aleksandar Tomic, Dragana Tomic, Steve Toscano, Djordje Trifunovic, Veljko Vasic, Tomer Verona, Aleksandar Vujic, Nikoli Vujic, Marko Vukovic, and Marko Zivanovic. 2020. POLARIS: the distributed SQL engine in Azure Synapse. *Proc. VLDB Endow.* 13, 12 (Aug. 2020), 3204–3216. <https://doi.org/10.14778/3415478.3415545>

[2] Vast AI. (last accessed) 2025. Vast.AI Instance Types. [Online] Available from: <https://cloud.vast.ai/create/>.

[3] AMD. (last accessed) 2025. AMD Instinct MI300X Platform. [Online] Available from: <https://www.amd.com/en/products/accelerators/instinct/mi300/platform.html>.

[4] AMD. (last accessed) 2025. ROCm Communication Collectives Library (RCCL). [Online] Available from: <https://github.com/ROCM/rcc1>.

[5] Felipe Aramburú, William Malpica, Kaouthar Abrougui, Amin Aramoon, Romulo Auccapuclla, Claude Brisson, Matthijs Brobbel, Colby Farrell, Pradeep Garigipati, Joost Hoozemans, Supun Kamburugamuve, Akhil Nair, Alexander Ocsa, Johan Peltenburg, Rubén Quesada López, Deepak Sihag, Ahmet Uyar, Dhruv Vats, Michael Wendt, Jignesh M. Patel, and Rodrigo Aramburú. 2025. Theseus: A Distributed and Scalable GPU-Accelerated Query Processing Platform Optimized for Efficient Data Movement. *arXiv:2508.05029* [cs,DC] <https://arxiv.org/abs/2508.05029>

[6] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, Todd J. Green, Monish Gupta, Sebastian Hillig, Eric Hotinger, Yan Leshinsky, Jintian Liang, Michael McCready, Fabian Nagel, Ippokratis Pandis, Panos Parchas, Rahul Pathak, Orestis Polychroniou, Foyzur Rahman, Gaurav Saxena, Gokul Soundararajan, Sriram Subramanian, and Doug Terry. 2022. Amazon Redshift Re-invented. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 2205–2217. <https://doi.org/10.1145/3514221.3526045>

[7] Yuki Asada, Victor Fu, Apurva Gandhi, Aditya Gemawat, Lihao Zhang, Dong He, Vivek Gupta, Ehi Nosakhare, Dalitso Banda, Rathijit Sen, and Matteo Interlandi. 2022. Share the tensor tea: how databases can leverage the machine learning ecosystem. *PVLDB* (2022), 3598–3601.

[8] Alexander Behm, Shoumik Palkar, Utkarsh Agarwal, Timothy Armstrong, David Cashman, Ankur Dave, Todd Greenstein, Shant Hovsepian, Ryan Johnson, Arvind Sai Krishnan, Paul Leventis, Ala Luszczak, Prashant Menon, Mostafa Mokhtar, Gene Pang, Sameer Paranjpye, Greg Rahn, Bart Samwel, Tom van Bussel, Herman van Hovell, Maryann Xue, Reynold Xin, and Matei Zaharia. 2022. Photon: A Fast Query Engine for Lakehouse Systems. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 2326–2339. <https://doi.org/10.1145/3514221.3526054>

[9] Nils Boeschen, Tobias Ziegler, and Carsten Binnig. 2024. GOLAP: A GPU-in-Data-Path Architecture for High-Speed OLAP. *Proc. ACM Manag. Data* 2, 6, Article 237 (Dec. 2024), 26 pages. <https://doi.org/10.1145/3698812>

[10] Peter Boncz, Angelos-Christos Anatiotis, and Steffen Kläbe. 2018. JCC-H: Adding Join Crossing Correlations with Skew to TPC-H. In *Performance Evaluation and Benchmarking for the Analytics Era*. Raghunath Nambiar and Meikel Poess (Eds.). Springer International Publishing, Cham, 103–119.

[11] Nicolas Bruno, César Galindo-Legaria, Milind Joshi, Esteban Calvo Vargas, Kabita Mahapatra, Sharon Ravindran, Guoheng Chen, Ernesto Cervantes Juárez, and Beyisim Sezgin. 2024. Unified Query Optimization in the Fabric Data Warehouse. In *Companion of the 2024 International Conference on Management of Data* (Santiago AA, Chile) (SIGMOD/PODS '24). Association for Computing Machinery, New York, NY, USA, 18–30. <https://doi.org/10.1145/3626246.3653369>

[12] Georgia Butler. 2024. Microsoft becomes first cloud to offer Nvidia Blackwell system. [Online] Available from: <https://www.datacenterdynamics.com/en/news/microsoft-becomes-first-cloud-to-offer-nvidia-blackwell-system/>. Last accessed: 2025.

[13] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2021. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) (PPoPP '21). Association for Computing Machinery, New York, NY, USA, 62–75. <https://doi.org/10.1145/3437801.3441620>

[14] Jiashen Cao, Rathijit Sen, Matteo Interlandi, Joy Arulraj, and Hyeseon Kim. 2023. GPU Database Systems Characterization and Optimization. *Proc. VLDB Endow.* 17, 3 (nov 2023), 441–454. <https://doi.org/10.14778/3632093.3632107>

[15] M. Cho, U. Finkler, M. Serrano, D. Kung, and H. Hunter. 2019. BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. *IBM Journal of Research and Development* 63, 6 (Nov 2019), 1:1–1:11. <https://doi.org/10.1147/JRD.2019.2947013>

[16] Periklis Chrysogelos, Manos Karpathiotakis, Raja Appuswamy, and Anastasia Ailamaki. 2019. HetExchange: encapsulating heterogeneous CPU-GPU parallelism in JIT compiled engines. *Proc. VLDB Endow.* 12, 5 (Jan. 2019), 544–556. <https://doi.org/10.14778/3303753.3303760>

[17] Google Cloud. (last accessed) 2025. GPU machine types. [Online] Available from: <https://cloud.google.com/compute/docs/gpus>.

[18] NVIDIA Corporation. (last accessed) 2025. NVIDIA Collective Communication Library (NCCL) Documentation. [Online] Available from: <https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/index.html>.

[19] NVIDIA Corporation. (last accessed) 2025. NVIDIA DGX H100. [Online] Available from: <https://www.nvidia.com/en-gb/data-center/dgx-h100/>.

[20] NVIDIA Corporation. (last accessed) 2025. NVIDIA GB200 NVL72. [Online] Available from: <https://www.nvidia.com/en-us/data-center/gb200-nvl72/>.

[21] NVIDIA Corporation. (last accessed) 2025. NVSHMEM. [Online] Available from: <https://developer.nvidia.com/nvshmem>.

[22] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. 2023. MSCCLang: Microsoft Collective Communication Language. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) (ASPLOS 2023). Association for Computing Machinery, New York, NY, USA, 502–514. <https://doi.org/10.1145/3575693.3575724>

[23] Wei Cui, Qianxi Zhang, Spyros Blanas, Jesús Camacho-Rodríguez, Brandon Haynes, Yinan Li, Ravi Ramamurthy, Peng Cheng, Rathijit Sen, and Matteo Interlandi. 2023. Query Processing on Gaming Consoles. In *Proceedings of the 19th International Workshop on Data Management on New Hardware* (Seattle, WA, USA) (DaMoN '23). Association for Computing Machinery, New York, NY, USA, 86–88. <https://doi.org/10.1145/3592980.3595313>

[24] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 215–226. <https://doi.org/10.1145/2882903.2903741>

[25] Volttron Data. (last accessed) 2025. Theseus The Enterprise SQL Engine. [Online] Available from: <https://volttrondata.com/>.

[26] Yangshen Deng, Shiwen Chen, Zhaoyang Hong, and Bo Tang. 2024. How Does Software Prefetching Work on GPU Query Processing?. In *Proceedings of the 20th International Workshop on Data Management on New Hardware* (Santiago, AA, Chile) (DaMoN '24). Association for Computing Machinery, New York, NY, USA, Article 5, 9 pages. <https://doi.org/10.1145/3662010.3663445>

[27] Jiawei Fei, Chen-Yu Ho, Atal N. Sahu, Marco Canini, and Amedeo Sapi. 2021. Efficient sparse collective communication and its application to accelerate distributed deep learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) (SIGCOMM '21). Association for Computing Machinery, New York, NY, USA, 676–691. <https://doi.org/10.1145/3452296.3472904>

[28] Apache Software Foundation. (last accessed) 2025. Apache Parquet. [Online] Available from: <https://parquet.apache.org/>.

[29] Apurva Gandhi, Yuki Asada, Victor Fu, Aditya Gemawat, Lihao Zhang, Rathijit Sen, Carlo Curino, Jesús Camacho-Rodríguez, and Matteo Interlandi. 2022. The Tensor Data Platform: Towards an AI-centric Database System. *arXiv:2211.02753* [cs, DB]. <https://arxiv.org/abs/2211.02753>

[30] Hao Gao and Nikolay Sakhnaykh. 2021. Scaling Joins to a Thousand GPUs. In *Proceedings of the 12th International Workshop on Accelerating Analytics and Data Management Systems* (ADMS). Copenhagen, Denmark. https://adms-conf.org/2021-camera-ready/gao_adms21.pdf Camera-ready version.

[31] Chengxin Guo, Hong Chen, Feng Zhang, and Cuiping Li. 2019. Distributed Join Algorithms on Multi-CPU Clusters with GPUDirect RDMA. In *Proceedings of the 48th International Conference on Parallel Processing* (Kyoto, Japan) (ICPP '19). Association for Computing Machinery, New York, NY, USA, Article 65, 10 pages. <https://doi.org/10.1145/3337821.3337862>

[32] Dong He, Supun C Nakandala, Dalitso Banda, Rathijit Sen, Karla Saur, Kwanghyun Park, Carlo Curino, Jesús Camacho-Rodríguez, Konstantinos Karanasos, and Matteo Interlandi. 2022. Query Processing on Tensor Computation Runtimes. *PVLDB* (2022), 2811–2825.

[33] Heavy.ai. (last accessed) 2025. HeavyDB. [Online] Available from: <https://www.heavy.ai/product/heavydb>.

[34] Heavy.ai. (last accessed) 2025. Speed at Scale: Benchmarking GPU-Accelerated HeavyDB. [Online] Available from: <https://support.heavy.ai/hc/en-us/articles/3444468459543-Speed-at-Scale-Benchmarking-GPU-Accelerated-HeavyDB>.

[35] Sven Hepkema, Azim Afrozeh, Charlotte Felius, Peter Boncz, and Stefan Manegold. 2025. G-ALP: Rethinking Light-weight Encodings for GPUs. In *Proceedings of the 21st International Workshop on Data Management on New Hardware* (DaMoN '25). Association for Computing Machinery, New York, NY, USA, Article

11, 10 pages. <https://doi.org/10.1145/3736227.3736242>

[36] Hewlett-Packard Enterprise. 2021. *TPC-H Full Disclosure Report for the HPE ProLiant DL385 Gen11*. Technical Report. Transaction Processing Performance Council (TPC). https://www.tpc.org/results/fdr/tpch/hpe-tpch-3000-hpe_proliant_dl385_gen11-fdr-2024-09-30-v01.pdf Report Date: October 1, 2024.

[37] Hewlett-Packard Enterprise. 2022. *TPC-H Full Disclosure Report for the HPE ProLiant DL385 Gen11*. Technical Report. Transaction Processing Performance Council (TPC). https://www.tpc.org/results/fdr/tpch/hpe-tpch-1000-hpe_proliant_dl385_gen11-fdr-2022-11-10-v01.pdf Report Date: November 10, 2022.

[38] Mert Hidayetoglu, Simon Garcia De Gonzalo, Elliott Slaughter, Yu Li, Christopher Zimmer, Tekin Bicer, Bin Ren, William Gropp, Wen-Mei Hwu, and Alex Aiken. 2024. CommBench: Micro-Benchmarking Hierarchical Networks with Multi-GPU, Multi-NIC Nodes. In *Proceedings of the 38th ACM International Conference on Supercomputing* (Kyoto, Japan) (ICS '24). Association for Computing Machinery, New York, NY, USA, 426–436. <https://doi.org/10.1145/3650200.3656591>

[39] Mert Hidayetoglu, Simon Garcia de Gonzalo, Elliott Slaughter, Pinku Surana, Wen mei Hwu, William Gropp, and Alex Aiken. 2024. HiCCL: A Hierarchical Collective Communication Library. [arXiv:2408.05962 \[cs.DC\]](https://arxiv.org/abs/2408.05962) <https://arxiv.org/abs/2408.05962>

[40] Kijae Hong, Kyoungmin Kim, Young-Koo Lee, Yang-Sae Moon, Sourav S Bhowmick, and Wook-Shin Han. 2025. Themis: A GPU-Accelerated Relational Query Execution Engine. *Proc. VLDB Endow.* 18, 2 (Feb. 2025), 426–438. <https://doi.org/10.14778/3705829.3705856>

[41] Yu-Ching Hu, Yuliang Li, and Hung-Wei Tseng. 2022. TCUDB: Accelerating Database with Tensor Processors. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 1360–1374. <https://doi.org/10.1145/3514221.3517869>

[42] Zhiyi Hu, Siyuan Shen, Tommaso Bonato, Sylvain Jeaugey, Cedell Alexander, Eric Spada, Jeff Hammond, and Torsten Hoefer. 2025. Demystifying NCCL: An In-depth Analysis of GPU Communication Protocols and Algorithms. [arXiv:2507.04786 \[cs.DC\]](https://arxiv.org/abs/2507.04786) <https://arxiv.org/abs/2507.04786>

[43] Zezhou Huang, Krystian Sakowski, Hans Lehrert, Wei Cui, Carlo Curino, Matteo Interlandi, Marius Dumitru, and Rathijit Sen. 2025. GPU Acceleration of SQL Analytics on Compressed Data. [arXiv:2506.10092 \[cs.DB\]](https://arxiv.org/abs/2506.10092) <https://arxiv.org/abs/2506.10092>

[44] Facebook Incubator. (last accessed) 2025. Gloo. [Online] Available from: <https://github.com/facebookincubator/gloo>.

[45] Michael Jungmair, André Kohn, and Jana Giceva. 2022. Designing an open framework for query optimization and compilation. *Proc. VLDB Endow.* 15, 11 (July 2022), 2389–2401. <https://doi.org/10.14778/3551793.3551801>

[46] Marko Kabić, Shriram Chandran, and Gustavo Alonso. 2025. Maximus: A Modular Accelerated Query Engine for Data Analytics on Heterogeneous Systems. *Proc. ACM Manag. Data* 3, 3, Article 187 (June 2025), 25 pages. <https://doi.org/10.1145/3725324>

[47] Marko Kabić, Bowen Wu, Jonas Dann, and Gustavo Alonso. 2025. Powerful GPUs or Fast Interconnects: Analyzing Relational Workloads on Modern GPUs. *Proc. VLDB Endow.* 18, 11 (Sept. 2025), 4350–4363. <https://doi.org/10.14778/374946.3749698>

[48] Tim Kaldewey, Guy Lohman, Rene Mueller, and Peter Volk. 2012. GPU join processing revisited. In *Proceedings of the Eighth International Workshop on Data Management on New Hardware* (Scottsdale, Arizona) (DaMoN '12). Association for Computing Machinery, New York, NY, USA, 55–62. <https://doi.org/10.1145/2236584.2236592>

[49] Tomas Karnagel, René Müller, and Guy M. Lohman. 2015. Optimizing GPU-accelerated Group-By and Aggregation. In *ADMS@VLDB*. <https://api.semanticscholar.org/CorpusID:5017248>

[50] Heehoon Kim, Junyeol Ryu, and Jaejin Lee. 2024. TCCL: Discovering Better Communication Paths for PCIe GPU Clusters. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) (ASPLOS '24). Association for Computing Machinery, New York, NY, USA, 999–1015. <https://doi.org/10.1145/3620666.3651362>

[51] Dimitrios Koutsoukos, Ingo Müller, Renato Marroquín, Ana Klimovic, and Gustavo Alonso. 2021. Modularis: modular relational analytics over heterogeneous distributed platforms. *Proc. VLDB Endow.* 14, 13 (Sept. 2021), 3308–3321. <https://doi.org/10.14778/3484224.3484229>

[52] Dimitrios Koutsoukos, Supun Nakandala, Konstantinos Karanasos, Karla Saur, Gustavo Alonso, and Matteo Interlandi. 2021. Tensors: an abstraction for general data processing. *Proc. VLDB Endow.* 14, 10 (June 2021), 1797–1804. <https://doi.org/10.14778/3467861.3467869>

[53] Artem Krovikov, Peter Kurapov, Christoph Anneser, and Jana Giceva. 2024. Heterogeneous Intra-Pipeline Device-Parallel Aggregations. In *Proceedings of the 20th International Workshop on Data Management on New Hardware* (Santiago, AA, Chile) (DaMoN '24). Association for Computing Machinery, New York, NY, USA, Article 3, 10 pages. <https://doi.org/10.1145/3662010.3663441>

[54] Lambda. (last accessed) 2025. On-Demand Cloud. [Online] Available from: <https://lambda.ai/service/gpu-cloud>.

[55] Yiran Lei, Dongjoo Lee, Liangyu Zhao, Dianiar Kurniawan, Chanmyeong Kim, Heetaek Jeong, Changsu Kim, Hyeonseong Choi, Liangcheng Yu, Arvind Krishnamurthy, Justine Sherry, and Erik Nurvitadhi. 2025. FLASH: Fast All-to-All Communication in GPU Clusters. [arXiv:2505.09764 \[cs.DC\]](https://arxiv.org/abs/2505.09764) <https://arxiv.org/abs/2505.09764>

[56] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. [arXiv:2006.16668 \[cs.CL\]](https://arxiv.org/abs/2006.16668) <https://arxiv.org/abs/2006.16668>

[57] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R. Tallent, and Kevin J. Barker. 2020. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Transactions on Parallel and Distributed Systems* 31, 1 (Jan 2020), 94–110. <https://doi.org/10.1109/TPDS.2019.2928289>

[58] Yinan Li, Bailu Ding, Ziyun Wei, Lukas Maas, Momin Al-Ghosn, Spyros Blanas, Nicolas Bruno, Carlo Curino, Matteo Interlandi, Craig Peepert, Kaushik Rajan, Surajit Chaudhuri, and Johannes Gehrke. 2025. Scaling GPU-Accelerated Databases Beyond GPU Memory Size. *Proc. VLDB Endow.* 18, 11 (Sept. 2025), 4518–4531. <https://doi.org/10.14778/374946.3749710>

[59] Yinan Li, Ippokratis Pandis, Rene Mueller, Vijayshankar Raman, and Guy M Lohman. 2013. NUMA-aware algorithms: the case of data shuffling. In *CIDR*.

[60] Zhongyi Lin, Ning Sun, Pallab Bhattacharya, Xizhou Feng, Louis Feng, and John D. Owens. 2024. Towards Universal Performance Modeling for Machine Learning Training on Multi-GPU Platforms. [arXiv:2404.12674 \[cs.DC\]](https://arxiv.org/abs/2404.12674) <https://arxiv.org/abs/2404.12674>

[61] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2020. Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 1633–1649. <https://doi.org/10.1145/3318464.3389705>

[62] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2022. Triton Join: Efficiently Scaling to a Large Join State on GPUs with Fast Interconnects. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 1017–1032. <https://doi.org/10.1145/3514221.3517911>

[63] Tobias Maltenberger, Ivan Ilic, Ilin Tolovski, and Tilmann Rabl. 2022. Evaluating Multi-GPU Sorting with Modern Interconnects. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 1795–1809. <https://doi.org/10.1145/3514221.3517842>

[64] Tobias Maltenberger, Ilin Tolovski, and Tilmann Rabl. 2025. Efficiently Joining Large Relations on Multi-GPU Systems. *Proc. VLDB Endow.* 18, 11 (Sept. 2025), 4653–4667. <https://doi.org/10.14778/374946.3749700>

[65] Wes McKinney. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, Vol. 445, 51–56.

[66] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, Hossein Ahmadi, Dan Delorey, Slava Min, Moshav Pasumansky, and Jeff Shute. 2020. Dremel: a decade of interactive SQL analysis at web scale. *Proc. VLDB Endow.* 13, 12 (Aug. 2020), 3461–3472. <https://doi.org/10.14778/3415478.3415568>

[67] Message Passing Interface Forum. 2023. *MPI: A Message-Passing Interface Standard Version 4.1*. <https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf>

[68] Microsoft. (last accessed) 2025. Linux Virtual Machines Pricing. [Online] Available from: <https://azure.microsoft.com/en-in/pricing/details/virtual-machines-linux/#pricing>.

[69] Hubert Mohr-Daurat, Xuan Sun, and Holger Pirk. 2023. BOSS - An Architecture for Database Kernel Composition. *Proc. VLDB Endow.* 17, 4 (Dec. 2023), 877–890. <https://doi.org/10.14778/3636218.3636239>

[70] NVIDIA. (last accessed) 2025. Doubling all2all Performance with NVIDIA Collective Communication Library 2.12. [Online] Available from: <https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/>.

[71] NVIDIA. (last accessed) 2025. NVIDIA GB200 NVL72. [Online] Available from: <https://https://www.nvidia.com/en-us/data-center/gb200-nvl72/>.

[72] NVIDIA. (last accessed) 2025. NVIDIA GH200 Grace Hopper Superchip. [Online] Available from: <https://resources.nvidia.com/en-us/grace-cpu/grace-hopper-superchip>.

[73] Johns Paul, Bingsheng He, Shengliang Lu, and Chiew Tong Lau. 2020. Revisiting hash join on graphics processors: A decade later. *Distributed and Parallel Databases* 38 (2020), 771–793.

[74] Johns Paul, Shengliang Lu, Bingsheng He, and Chiew Tong Lau. 2021. MG-Join: A Scalable Join for Massively Parallel Multi-GPU Architectures. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) (SIGMOD '21). Association for Computing Machinery, New York, NY,

USA, 1413–1425. <https://doi.org/10.1145/3448016.3457254>

[75] Carl Pearson, Abdul Dakkak, Sarah Hashash, Cheng Li, I-Hsin Chung, Jinjun Xiong, and Wen-Mei Hwu. 2019. Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering* (Mumbai, India) (ICPE '19). Association for Computing Machinery, New York, NY, USA, 209–218. <https://doi.org/10.1145/3297663.3310299>

[76] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G.E. Fagg, E. Gabriel, and J.J. Dongarra. 2005. Performance analysis of MPI collective operations. In *19th IEEE International Parallel and Distributed Processing Symposium*. 8 pp.–. <https://doi.org/10.1109/IPDPS.2005.335>

[77] RAPIDS. (last accessed) 2025. cuDF: A GPU DataFrame Library. [Online] Available from: <https://github.com/rapidsai/cudf>.

[78] RAPIDS. (last accessed) 2025. Dask cuDF's documentation. [Online] Available from: <https://docs.rapids.ai/api/dask-cudf/stable/>.

[79] Wolf Rödiger, Tobias Mühlbauer, Alfons Kemper, and Thomas Neumann. 2015. High-speed query processing over high-speed networks. *Proc. VLDB Endow.* 9, 4 (Dec. 2015), 228–239. <https://doi.org/10.14778/2856318.2856319>

[80] Viktor Rosenfeld, Sebastian Breß, and Volker Markl. 2022. Query Processing on Heterogeneous CPU/GPU Systems. *ACM Comput. Surv.* 55, 1, Article 11 (jan 2022), 38 pages. <https://doi.org/10.1145/3485126>

[81] Viktor Rosenfeld, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2019. Performance Analysis and Automatic Tuning of Hash Aggregation on GPUs. In *Proceedings of the 15th International Workshop on Data Management on New Hardware* (Amsterdam, Netherlands) (DaMoN'19). Association for Computing Machinery, New York, NY, USA, Article 8, 11 pages. <https://doi.org/10.1145/3329785.3329922>

[82] Ran Rui and Yi-Cheng Tu. 2017. Fast Equi-Join Algorithms on GPUs: Design and Implementation. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management* (Chicago, IL, USA) (SSDBM '17). Association for Computing Machinery, New York, NY, USA, Article 17, 12 pages. <https://doi.org/10.1145/3085504.3085521>

[83] Peter Sanders, Jochen Speck, and Jesper Larsson Träff. 2009. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Comput.* 35, 12 (2009), 581–594. <https://doi.org/10.1016/j.parco.2009.09.001> Selected papers from the 14th European PVM/MPI Users Group Meeting.

[84] Amazon Web Service. (last accessed) 2025. Amazon EC2 P5 Instances. [Online] Available from: <https://aws.amazon.com/ec2/instance-types/p5/>.

[85] Andres Sewell, Ke Fan, Ahmedur Rahman Shovon, Landon Dyken, Sidharth Kumar, and Steve Petruzzia. 2024. Bruck Algorithm Performance Analysis for Multi-GPU All-to-All Communication. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region* (Nagoya, Japan) (HPCAsia '24). Association for Computing Machinery, New York, NY, USA, 127–133. <https://doi.org/10.1145/3635035.3635047>

[86] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation* (NSDI '23). USENIX Association, Boston, MA, 593–612. <https://www.usenix.org/conference/nsdi23/presentation/shah>

[87] Pavel Shamis, Manjunath Gorenthal Venkata, M. Graham Lopez, Matthew B. Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L. Graham, Liran Liss, Yiftah Shahar, Sreeram Potluri, Davide Rossetti, Donald Becker, Duncan Poole, Christopher Lamb, Sameer Kumar, Craig Stunkel, George Bosilca, and Aurelien Bouteiller. 2015. UCX: An Open Source Framework for HPC Network APIs and Beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. 40–43. <https://doi.org/10.1109/HOTI.2015.13>

[88] Anil Shanbhag, Samuel Madden, and Xiangyao Yu. 2020. A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics. In *SIGMOD*. 1617–1632.

[89] Anil Shanbhag, Bobbi W. Yogatama, Xiangyao Yu, and Samuel Madden. 2022. Tile-based Lightweight Integer Compression in GPU. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 1390–1403. <https://doi.org/10.1145/3514221.3526132>

[90] Panagiotis Sioulas, Periklis Chrysogelos, Manos Karpathiotakis, Raja Appuswamy, and Anastasia Ailamaki. 2019. Hardware-Conscious Hash-Joins on GPUs. In *2019 IEEE 35th International Conference on Data Engineering* (ICDE). 698–709. <https://doi.org/10.1109/ICDE.2019.00068>

[91] Wenbo Sun, Asterios Katsifodimos, and Rihan Hai. 2023. An Empirical Performance Comparison between Matrix Multiplication Join and Hash Join on GPUs. In *2023 IEEE 39th International Conference on Data Engineering Workshops (ICDEW)*. 184–190. <https://doi.org/10.1109/ICDEW58674.2023.00034>

[92] Sacheendra Talluri, Guido Walter Di Donato, Luca Danelutti, Koen R. Vlaswinkel, Marco Arnaboldi, Arnaud Delamare, Marco D. Santambrogio, and Daniele Bonetta. 2025. GpJSON: High-Performance JSON Data Processing on GPUs. *Proc. VLDB Endow.* 18, 9 (Sept. 2025), 3216–3229. <https://doi.org/10.14778/3746405.3746439>

[93] Lasse Thoststrup, Gloria Docu, Nils Boeschen, Manisha Luthra, and Carsten Binnig. 2023. Distributed GPU Joins on Fast RDMA-capable Networks. *Proc. ACM Manag. Data* 1, 1, Article 29 (may 2023), 26 pages. <https://doi.org/10.1145/3588709>

[94] Diego G. Tomé, Tim Gubner, Mark Raasveldt, Eyal Rozenberg, and Peter A. Boncz. 2018. Optimizing Group-By and Aggregation using GPU-CPU Co-Processing. In *ADMS@VLDB*. <https://api.semanticscholar.org/CorpusID:52895287>

[95] Karl B. Torp, Simon A. F. Lund, and Pinar Tözün. 2025. Path to GPU-Initiated I/O for Data-Intensive Systems. In *Proceedings of the 21st International Workshop on Data Management on New Hardware* (DaMoN '25). Association for Computing Machinery, New York, NY, USA, Article 3, 9 pages. <https://doi.org/10.1145/3736227.3736232>

[96] Transaction Processing Performance Council. 2022. *TPC Benchmark H (Decision Support) Standard Specification*. Technical Report. Transaction Processing Performance Council (TPC). https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf Version 3.0.1.

[97] Oliphant Travis E. (last accessed) 2025. NumPy. [Online] Available from: <http://www.numpy.org/>.

[98] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Nikhil Devanur, Jorgen Thelin, and Ion Stoica. 2020. Blink: Fast and Generic Collectives for Distributed ML. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 172–186. https://proceedings.mlsys.org/paper_files/paper/2020/file/cd3a9a55f7f3723133fa4a13628cdf03-Paper.pdf

[99] Weiyang Wang, Moein Khazraee, Zhiheng Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation* (NSDI '23). USENIX Association, Boston, MA, 739–767. <https://www.usenix.org/conference/nsdi23/presentation/wang-weiyang>

[100] Adam Weingram, Yuke Li, Hao Qi, Darren Ng, Liuyao Dai, and Xiaoyi Lu. 2023. xCCL: A Survey of Industry-Led Collective Communication Libraries for Deep Learning. *J. Comput. Sci. Technol.* 38, 1 (March 2023), 166–195. <https://doi.org/10.1007/s11390-023-2894-6>

[101] Udayanga Wickramasinghe and Andrew Lumsdaine. 2016. A Survey of Methods for Collective Communication Optimization and Tuning. [arXiv:1611.06334 \[cs.DC\]](https://arxiv.org/abs/1611.06334) <https://arxiv.org/abs/1611.06334>

[102] Bowen Wu, Wei Cui, Carlo Curino, Matteo Interlandi, and Rathijit Sen. 2025. Terabyte-Scale Analytics in the Blink of an Eye. [arXiv:2506.09226 \[cs.DB\]](https://arxiv.org/abs/2506.09226) <https://arxiv.org/abs/2506.09226>

[103] Bowen Wu, Dimitrios Koutsoukos, and Gustavo Alonso. 2025. Efficiently Processing Joins and Grouped Aggregations on GPUs. *Proc. ACM Manag. Data* 3, 1, Article 39 (Feb. 2025), 27 pages. <https://doi.org/10.1145/3709689>

[104] Bobbi Yogatama, Weiwei Gong, and Xiangyao Yu. 2025. Scaling your Hybrid CPU-GPU DBMS to Multiple GPUs. *Proc. VLDB Endow.* 17, 13 (Feb. 2025), 4709–4722. <https://doi.org/10.14778/3704965.3704977>

[105] Bobbi Yogatama, Brandon Miller, Yunsong Wang, Graham Markall, Jacob Hemstad, Gregory Kimball, and Xiangyao Yu. 2023. Accelerating User-Defined Aggregate Functions (UDAf) with Block-wide Execution and JIT Compilation on GPUs. In *Proceedings of the 19th International Workshop on Data Management on New Hardware* (Seattle, WA, USA) (DaMoN '23). Association for Computing Machinery, New York, NY, USA, 19–26. <https://doi.org/10.1145/3592980.3595307>

[106] Bobbi W. Yogatama, Weiwei Gong, and Xiangyao Yu. 2022. Orchestrating data placement and query execution in heterogeneous CPU-GPU DBMS. *Proc. VLDB Endow.* 15, 11 (July 2022), 2491–2503. <https://doi.org/10.14778/3551793.3551809>

[107] Yichao Yuan, Advait Iyer, Lin Ma, and Nishil Talati. 2025. Vortex: Overcoming Memory Capacity Limitations in GPU-Accelerated Large-Scale Data Analytics. *Proc. VLDB Endow.* 18, 4 (May 2025), 1250–1263. <https://doi.org/10.14778/3717755.3717780>

[108] Yuan Yuan, Rubao Lee, and Xiaodong Zhang. 2013. The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. *Proc. VLDB Endow.* 6, 10 (aug 2013), 817–828. <https://doi.org/10.14778/2536206.2536210>

[109] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (Oct. 2016), 56–65. <https://doi.org/10.1145/2934664>