Data Movement-Aware GPU Sharing for Data-Intensive Systems

Yi Jiang
EPFL
Lausanne, Switzerland
yi.jiang@epfl.ch

Hamish Nicholson
EPFL
Lausanne, Switzerland
hamish.nicholson@epfl.ch

Viktor Sanca*
Oracle
Redwood City, United States
viktor.sanca@oracle.com

Anastasia Ailamaki
EPFL
Lausanne, Switzerland
anastasia.ailamaki@epfl.ch

ABSTRACT

Modern data platforms require GPU acceleration for both analytical query processing and AI inference. These workloads exhibit contrasting resource bottlenecks: analytics saturates the PCIe interconnect with data transfers, while inference stresses on-device computation. We observe that when colocated, these workloads create opportunities rather than conflicts: one workload's bottleneck corresponds to idle resources for the other. Yet existing GPU sharing mechanisms fail to exploit this insight, degrading performance.

We present GPU Unified Sharing with Transfer-awareness (GUST) scheduling that treats the PCIe interconnect as a first-class, schedulable resource alongside compute and memory. Our approach monitors workload interconnect intensity to classify tasks as transfer-intensive or device-intensive, then intelligently interleaves their execution. By scheduling transfer-intensive analytics kernels to saturate PCIe bandwidth while running device-intensive inference kernels in the gaps, our scheduler achieves high utilization of both the interconnect and GPU compute resources. In experiments that colocate four mixed analytics and inference workloads, our prototype reduces performance degradation compared to dedicated GPU performance from 3.9-7× (geometric mean) for existing sharing mechanisms to 2.8×. By making data movement visible to the scheduler, we transform GPUs from dedicated accelerators into efficiently shared computational resources suitable for the heterogeneous workloads of modern data systems.

1 INTRODUCTION

Graphics Processing Units (GPUs) are increasingly pivotal for a spectrum of demanding computational tasks [22]. Substantial research has demonstrated the benefits of GPUs for analytical query processing [2, 11, 15, 24, 34], and modern data platforms are increasingly integrating AI with traditional analytics for workflows like hybrid vector-relational query processing [13, 23, 26, 38] and retrieval-augmented generation [3, 21, 28]. Analytics and AI inference workloads each underutilize GPU resources, but along different dimensions, exposing an opportunity to evolve GPUs from dedicated accelerators into shared general-purpose accelerators that enable tighter application integration while simultaneously improving hardware utilization.

We make the key observation that analytical and inference workloads are bottlenecked by disjoint resources: on one hand, traditional data-intensive workloads, such as analytical query processing, are frequently bottlenecked by the movement of data from host memory or storage to the GPU over the system interconnect (e.g., PCIe) [15, 25, 36]. On the other hand, many modern AI inference tasks are device-intensive, either compute-bound or memory-bound, demanding sustained use of the GPU's on-device processing cores and high-bandwidth memory, with comparatively small data inputs [8]. This inherent conflict in resource demands, i.e., one class of workload stressing the interconnect, the other stressing on-device resources, creates a significant scheduling problem that existing systems are ill-equipped to handle.

Current GPU resource management strategies fail to address the scheduling challenges when device-intensive and transfer-intensive workloads are colocated. The simplest solution, deploying workloads on dedicated GPUs, provides complete performance isolation but multiplies infrastructure costs. Time-slicing alternates exclusive GPU access, resulting in resource waste: the interconnect sits idle during AI kernel execution, while compute units sit relatively idle while analytics is running. NVIDIA's Multi-Process Service (MPS) [19] enables spatial sharing but lacks workload-aware scheduling, resulting in uncontrolled interference as both workload types compete for resources without coordination. More sophisticated schedulers like Orion [29] optimize for AI workloads by managing on-device compute and memory interference, but are oblivious to the PCIe transfers that dominate analytics workloads.

This paper argues for a future where GPUs evolve from dedicated accelerators into shared, general-purpose compute substrates that natively support colocated device-intensive and transfer-intensive workloads. We argue that the interconnect must be treated as a first-class, schedulable resource. By making the scheduler aware of the data-movement characteristics of each task sharing the GPU, a system can intelligently interleave the data transfer phase of a data-transfer-intensive workload with the compute phase of a device-intensive one. This "whole-system" pipelining aims to keep both the PCIe bus and the GPU's execution units highly utilized, thereby improving overall throughput and latency. In summary, the contributions of this paper are:

- We demonstrate that GPU-accelerated analytics and AI inference have complementary resource patterns that existing

*This work was done while the author was at EPFL.

 16th Annual Conference on Innovative Data Systems Research (CIDR '26). January 18-21, Chaminade, USA

schedulers fail to exploit, resulting in poor resource utilization and performance degradation.
- We propose GPU Unified Sharing with Transfer-awareness (GUST), a GPU scheduler that dynamically co-schedules data transfers with processing-intensive tasks to saturate GPU compute resources and interconnect bandwidth.
- Our experiments show that a transfer-aware GPU scheduler can reduce performance degradation from 3.9-7× (geometric mean) in existing sharing mechanisms to 2.8× for mixed transfer-intensive and device-intensive workloads.

2 GPUS AS GENERAL-PURPOSE ACCELERATORS

The evolution of GPUs from specialized graphics processors to general-purpose accelerators has created opportunities and challenges for modern data systems. Originally designed for rendering, GPUs have evolved into powerful computational workhorses that now underpin various workloads [22]. This evolution began with scientific computing and machine learning, where GPUs demonstrated orders-of-magnitude speedups for parallel computations. Today, GPU acceleration extends far beyond these traditional domains: financial institutions use GPUs for risk modeling and fraud detection, healthcare systems accelerate genomic analysis and medical imaging, and increasingly, data management systems leverage GPUs to accelerate query processing [2, 27, 34].

2.1 From Specialized to Shared Infrastructure

The programmability of GPUs has dramatically improved alongside this hardware evolution. Modern programming frameworks like CUDA and ROCm, with higher-level libraries and the recent emergence of LLM-assisted development, have lowered the barrier to GPU acceleration. Yet despite these advances, individual workloads frequently underutilize GPU resources. Analytics workloads may saturate the PCIe interconnect while leaving compute units idle; inference tasks may fully utilize compute cores while the interconnect sits unused. This pattern of partial resource utilization suggests that GPUs, like CPUs before them, require sophisticated runtime scheduling to maximize their value.

Modern data platforms no longer treat analytics and AI as separate concerns. Systems [21, 28, 38] blend SQL query processing with embedding generation, vector similarity search with relational operations, all within unified platforms. Retrieval-augmented generation (RAG) systems require both vector similarity search and traditional query processing [3]. These hybrid workloads represent a fundamental shift in how we architect data systems. Application needs are driving this convergence. A financial fraud detection system might join transaction histories while simultaneously running inference on behavioral patterns; a recommendation engine combines collaborative filtering with semantic understanding. Running these workloads on separate infrastructure introduces complexity, latency, and data movement overhead. Colocating them on shared GPU infrastructure offers the potential for tighter integration and better resource utilization.

2.2 GPU Cost-Performance Evolution

The rapid pace of GPU innovation has created a shifting market dynamic. As organizations adopt cutting-edge GPUs for training frontier models, a substantial volume of previous-generation hardware becomes available [1]. Cloud providers have responded with corresponding price reductions [4]. This pricing evolution means workloads can now affordably access GPU acceleration at the appropriate capability level. Previous-generation GPUs offer compelling alternatives for data-intensive workloads. The trend is clear: yesterday's flagship becomes today's commodity, but with capabilities that often exceed current mid-range offerings in key metrics like memory bandwidth that matter for analytics workloads. A Volta-generation V100, despite being released in 2017, provides higher bandwidth HBM than newer GDDR6-based cards. For workloads bottlenecked by memory bandwidth rather than compute, these older high-bandwidth GPUs offer a good value proposition [16–18].

2.3 The Challenge of Heterogeneous Colocation

Despite this convergence of workloads and improving economics, a fundamental mismatch persists. Individual workloads chronically underutilize GPU resources, but in complementary ways. Analytics queries processing large datasets spend most cycles waiting on PCIe transfers, leaving GPU compute units idle [15, 25]. Conversely, AI inference tasks with small input batches saturate compute resources while the interconnect sits unused [8]. These disjoint resource demands represent an opportunity: where one workload creates a bottleneck, the other has slack resources.

The challenge is that these heterogeneous workloads break the assumptions of existing GPU sharing approaches. Prior work on GPU multiplexing, whether through spatial sharing [19, 29] or temporal sharing was designed for relatively homogeneous deep learning workloads that compete for on-device resources. These approaches either manage concurrent kernel-level interference or alternate exclusive access without concern for the interconnect, failing to exploit the complementary resource usage of interconnect-bound and device-bound workloads. While recent work like Vortex [36] recognizes that analytics and AI workloads stress different resources, it addresses static placement on multi-GPU servers rather than dynamic runtime scheduling. Recent GPU operating system advancements, such as LithOS [7], introduce fine-grained spatial scheduling at the granularity of texture processing clusters. While such systems significantly improve on-device resource utilization, they remain orthogonal to our approach as they do not consider the PCIe interconnect as a schedulable resource. Consequently, these approaches focus on managing on-device resource interference in isolation from the interconnect and cannot exploit the complementary resource usage patterns of data-transfer-intensive and device-intensive workloads.

This evolution, from specialized to general-purpose hardware, from isolated to converged workloads, from scarcity to accessibility, demands rethinking GPU resource management. The interconnect must become a first-class schedulable resource, not an afterthought. The key insight is that effective GPU sharing for heterogeneous workloads requires not just managing interference, but actively exploiting complementary resource patterns. In the following section,

we characterize these workload patterns in detail to understand how a data-aware scheduler can realize this vision.

3 THE OVERLOOKED INTERCONNECT

To understand GPU sharing challenges, we analyze two representative workloads from modern data systems. Analytics workloads perform GPU-accelerated query processing. Embedding generation workloads process text into vector representations, a foundational operation for semantic search and retrieval-augmented generation. These exhibit contrasting resource patterns: analytics is bottlenecked by PCIe data transfers from host memory, while embedding generation is bottlenecked by on-device computation. Section 3.1 first characterizes these patterns, then Section 3.2 shows how current GPU sharing mechanisms fail to exploit their complementarity.

3.1 A Tale of Two Workloads

Data-Transfer-Intensive Analytics: When datasets exceed the limited capacity of GPU memory, moving data to the GPU either before or during query execution is the dominant bottleneck in GPU-accelerated query processing [15, 25, 36]. For non-GPU-memory-resident data, streaming execution implements pipelining by initiating transfers asynchronously as kernels consume previously transferred data, maintaining a bounded buffer on the GPU [24]. However, even with pipelining, query processing is frequently blocked waiting on data transfers. Figure 2a conceptually plots a timeline of data transfers over the PCIe bus and kernel execution while processing a query on a GPU. As a concrete example, in our A100 experimental setup (Section 5), Star Schema Benchmark (SSB) query 3.1 requires 330 μs to transfer each 8 MiB batch of column chunks but only 92 μs to process it. After the first batch of column chunks arrives, the transfer of the next batch begins while the first batch is consumed. This pattern results in high PCIe interconnect utilization (near 100% in our example) while GPU compute resources remain severely underutilized, with the GPU spending most of its time idle waiting for data transfers to complete.

Device-Intensive Inference: In contrast to data-transfer-intensive analytics, AI inference workloads primarily stress on-device GPU resources, the compute units and GPU memory, rather than the interconnect [8]. The computational characteristics vary across different model architectures and phases of execution. Some phases, such as processing large batches of inputs in parallel, can be compute-bound with high arithmetic intensity that fully saturates GPU compute units. Other phases, particularly those involving sequential processing or small batch sizes, become memory-bandwidth-bound as the GPU spends more time loading model parameters from memory than performing actual computation [37]. This creates a complementary resource pattern: analytics workloads heavily utilize the PCIe bus but underutilize GPU compute, while inference workloads intensively use GPU compute and memory bandwidth while leaving the PCIe interconnect largely idle. Figure 2b illustrates this pattern, showing sustained GPU kernel execution with minimal PCIe activity, presenting an opportunity for intelligent colocation of these complementary workloads.

3.2 GPU Sharing in Practice

Existing GPU sharing mechanisms were not designed for data movement heterogeneity. We examine three approaches: temporal sharing, spatial sharing, and physical partitioning, and show how each fails to exploit these complementary patterns, resulting in either underutilization or interference.

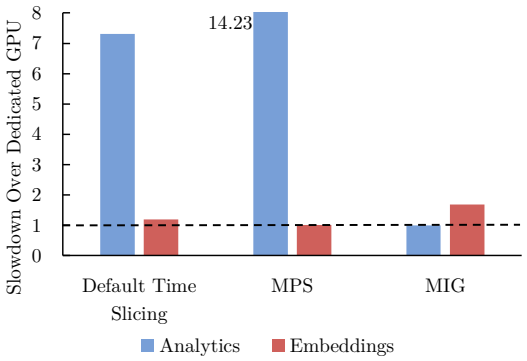

Figure 1: Performance degradation of mixed analytics and inference workloads under existing GPU sharing mechanisms.

Temporal sharing mechanisms alternate exclusive GPU access between different workloads, with CUDA's default time-slicing being the most widely used implementation. When multiple processes or CUDA contexts use the same GPU, the CUDA driver time-slices kernel execution, but we observe that PCIe transfers can proceed concurrently from any context. As illustrated in Figure 2c, when analytics kernels cannot execute frequently enough to consume transferred data, the query engine's bounded transfer buffer fills up. Once full, the streaming execution model halts new PCIe transfers, creating gaps in PCIe utilization despite transfers not being time-sliced. Meanwhile, during inference kernel execution, the GPU compute units are fully utilized, but the PCIe bus sits idle. This pattern of infrequent analytics kernel scheduling leads to transfer pipeline stalls, resulting in poor utilization of both the PCIe interconnect and GPU compute resources across the workload mix.

Spatial Sharing allows multiple workloads to execute concurrently on the GPU. On NVIDIA GPUs, MPS [19] and CUDA streams enable concurrent kernel execution across processes and streams, respectively. However, both lack awareness of workload resource requirements. When analytics and inference workloads run together, they compete for resources without coordination: inference kernels with high register requirements prevent analytics kernels from being scheduled on the same SMs, which stalls data consumption and subsequently halts PCIe transfers (Figure 2d). When kernels do execute concurrently, resource contention degrades both workloads' performance. The unmanaged competition leads to cascading failures: register pressure blocks analytics progress, stalling the transfer pipeline, and leaving the PCIe bus idle.

Physical partitioning provides hardware-level isolation by dividing GPU resources into separate partitions for each workload. Modern NVIDIA GPUs offer this through the Multi-Instance GPU (MIG) feature [20], which splits a GPU into multiple isolated instances, each with dedicated SMs, memory bandwidth, and cache.

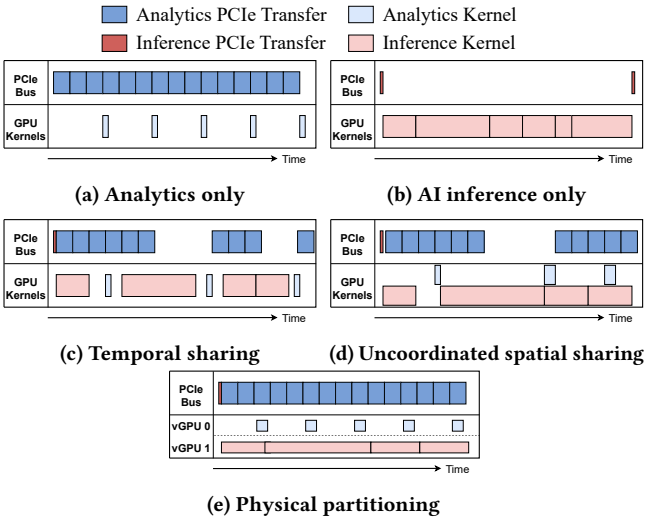

Figure 2: Conceptual illustration of GPU resource utilization patterns under different GPU sharing approaches. Each timeline shows PCIe bus (top) and GPU kernel (bottom) activity. (a-b) Isolated workload baselines showing complementary resource usage patterns. (c-d) Basic GPU sharing mechanisms (without workload-aware scheduling) underutilize the hardware with mixed device- and transfer-intensive workloads.

Unlike CUDA Stream and MPS's logical sharing, MIG guarantees performance isolation through physical resource separation. As shown in Figure 2e, when the two workloads are assigned to separate GPU instances, each receives guaranteed resources. However, this isolation comes at the cost of static partitioning: resources allocated to one instance cannot be used by another, even when idle. In our example, vGPU 0 remains underutilized during periods between analytics kernels, while vGPU 1 cannot access these idle compute resources despite having work ready to execute. Moreover, MIG provides no PCIe isolation or coordination between instances.

Figure 1 quantifies these limitations. We measure the performance when colocating our representative analytics and inference workloads using these three vendor-provided mechanisms. Each favors one workload at the expense of the other: temporal sharing degrades analytics performance by 7.3× while inference sees only 1.2× slowdown; MPS devastates analytics with 14.2× slowdown while leaving inference nearly unaffected (1.01×); MIG isolates analytics from interference, and since analytics requires minimal compute resources but penalizes inference with 1.7× slowdown due to reduced available compute. These results demonstrate that MPS, time-slicing, and MIG cannot efficiently support workloads with contrasting interconnect versus on-device resource demands, suggesting the need for schedulers that understand and exploit the different bottlenecks of data-transfer-intensive and device-intensive workloads.

While our experiments focus on vendor-provided mechanisms due to their widespread deployment, we recognize that these three approaches (temporal sharing, spatial sharing, and physical partitioning) represent the fundamental ways GPUs can be shared between workloads. The vendor-provided mechanisms we examined (CUDA time-slicing, MPS/streams, and MIG) offer basic implementations of these concepts but lack workload awareness. More advanced schedulers proposed in prior work build on top of these same underlying mechanisms with additional scheduling logic. For example, Orion [29] implements interference-aware spatial sharing by using MPS or CUDA streams for concurrent kernel execution while adding a scheduling layer that utilizes ahead-of-time kernel profiling to minimize interference. Similarly, schedulers like Gandiva [35] and Clockwork [9] enhance temporal sharing with improved state management and deadline awareness. These works have made significant progress in managing on-device resource contention for their target workloads. Our goal is complementary, to extend GPU scheduling to explicitly consider PCIe data transfers as a first-class resource, enabling efficient colocation of workloads with contrasting resource demands: those bottlenecked by data movement and those bottlenecked by on-device computation.

4 TRANSFER-AWARE GPU SHARING

This section presents our vision for GPU Unified Sharing with Transfer-awareness (GUST), a GPU scheduler that treats PCIe bandwidth as a first-class resource. By making the interconnect visible and schedulable, GUST can enable the efficient colocation of transfer-intensive and device-intensive workloads.

4.1 Core Design Principles

We identify three core design principles required for transfer-aware GPU scheduling:

Whole-system visibility: The scheduler must observe both PCIe transfers and GPU kernel execution to understand the complete resource utilization pattern. By intercepting CUDA API calls through a proxy library or operating system kernel module, the scheduler can track all data movement operations and kernel launch activities. This visibility extends beyond simple event tracking; using CUDA events to measure kernel execution times and tools like NVML and CUPTI to monitor hardware utilization, the scheduler builds a comprehensive view of each workload's behavior.

Online workload characterization: The scheduler dynamically classifies workloads based on their interconnect intensity: the ratio of time spent in data transfer over the interconnect to that in computational work. Through runtime monitoring of data transfer latencies and kernel execution times, combined with hardware utilization metrics from NVML and CUPTI, the scheduler can estimate each workload's position on the spectrum from data-transfer-intensive to device-intensive. This characterization happens online, allowing the scheduler to adapt to unseen and varying workloads.

Dynamic adaptation: Rather than using static scheduling policies, the scheduler adjusts its decisions based on observed workload characteristics and system state. Data-transfer-intensive workloads are granted execution priority to prevent data movement stalls due to contention with colocated tasks. Device-intensive workloads are scheduled opportunistically, leveraging the idle GPU cycles available during data transfer phases of other workloads.

4.2 System Architecture: An OS-inspired Blueprint

To realize these principles, GUST adopts the architectural blueprint of a classic operating system scheduler. We map the logical requirements of the core design principles to three distinct system layers: Accounting, Scheduling, and Dispatching.

Accounting (Visibility): Similar to the Linux kernel's Accounting Subsystem, which tracks resource usage (e.g., via `taskstats` or Control Group stats) to inform system policy [30], GUST implements an accounting layer using CUPTI and NVML. This layer provides system visibility by measuring the *Interconnect Intensity* ($IC_{\text{intensity}}$) metric for every active context.

The Scheduler (Characterization): Analogous to the kernel's scheduler which uses these statistics to organize tasks [31], GUST's runtime classification engine evaluates the accounting data, mapping workloads to specific priority classes, classifying them as *transfer-intensive* or *device-intensive*.

The Dispatcher (Adaptation): Finally, just as the operating system performs a context switch to physically enforce priorities [32], GUST uses an interception layer via `LD_PRELOAD` as a user-space dispatcher. This mechanism intercepts CUDA API calls and routes them into the prioritized streams or queues dictated by the scheduler, enforcing the logical decisions made by the scheduling layer.

4.3 From GPU Workload Metrics to Decisions

Just as the Linux kernel separates mechanism from policy, GUST employs a dedicated heuristic to transform raw accounting data into actionable dispatching decisions. To exploit the complementary nature of data-transfer-intensive and device-intensive workloads, the scheduler first needs to characterize and identify the resource patterns of colocated tasks. The classification must be lightweight and online to ensure generalization to a wide range of workloads.

For example, the most straightforward way to characterize workloads is to sample their execution on the target GPU in isolation for a short profiling window, and then classify their resource patterns as data-transfer-intensive or device-intensive. During this interval (5s in our experiment), the scheduler monitors and collects the aggregate time spent on data transfers, denoted as $T_{\text{data\_transfer}}(\omega)$, and kernel execution, denoted as $T_{\text{kernel\_execution}}(\omega)$. These metrics are used to calculate workload ω's **Interconnect Intensity**, denoted as $IC_{\text{intensity}}(\omega)$.

$$IC_{\text{intensity}}(\omega) = \frac{T_{\text{data\_transfer}}(\omega)}{T_{\text{kernel\_execution}}(\omega)}$$

After obtaining the Interconnect Intensity of all colocated workloads, the scheduler classifies as data-transfer-intensive or device-intensive, using a tunable threshold. In our experiments, we set this threshold to 1. Workloads with $IC_{\text{intensity}}$ greater than 1 are classified as data-transfer-intensive, as they spend more time moving data than processing it. For instance, the queries in our evaluation exhibit high interconnect intensity (e.g., SSB Q1.1 at 22.8 and Q3.1 at 4.9), suggesting that query processing is bottlenecked by interconnect bandwidth on our hardware configurations. Conversely, workloads with $IC_{\text{intensity}} \leq 1$ are classified as device-intensive, as their performance is dominated by on-device computation and/or memory accesses, like the embedding generation workload.

For data-transfer-intensive workloads, the scheduler prioritizes their execution to ensure that the data transfers effectively utilize the available PCIe interconnect bandwidth. Between these prioritized kernel executions, substantial GPU compute resources remain available, as data-transfer-intensive kernels typically underutilize the GPU's processing cores. During these gaps, the scheduler runs kernels from device-intensive workloads. Since these workloads have minimal data transfer requirements, they can fully utilize the GPU's processing cores without competing for PCIe bandwidth.

This approach aims to improve the utilization of both resources: the PCIe bus remains saturated with data transfers from data-intensive workloads, while GPU compute units process kernels from both workload types.

5 EXPERIMENTAL EVALUATION

Hardware. We evaluate our approach on two GPU platforms representing different hardware generations and capabilities. The first is a bare metal dual socket server with 2× 12-core Intel Xeon Gold 5118 CPUs and a NVIDIA V100S connected via 16× PCIe 3.0 lanes to each socket. The second setup is an Azure `Standard_NC24ads_A100_v4` instance with a single NVIDIA A100 connected via 16× PCIe 4.0 lanes. In all experiments on the V100S machine, we use a single V100S. These configurations represent the transition from PCIe 3.0 (V100S) to PCIe 4.0 (A100) interconnects.

Software. We implement our experiments using the following:

Analytics engine: We use Proteus [6], a code-generating query engine with GPU acceleration support. Proteus generates fused CUDA kernels for query operators and uses asynchronous transfers and pipelined execution to overlap data transfer and processing.

Inference framework: Embedding generation is an increasingly critical component of modern data platforms, powering semantic search, retrieval-augmented generation (RAG), and similarity queries. To represent this important class of workload, we implement a batch embedding generation task using the vLLM v0.8.5 inference framework [12] to generate embeddings from pre-tokenized text. The framework processes requests in batches to maintain continuous GPU utilization throughout our experiments.

System To implement a proof-of-concept scheduler we utilize MPS to allow basic sharing between processes, and intercept calls that launch kernels or transfer data between system memory and GPU memory. The calls are intercepted by overriding the relevant CUDA runtime API functions through pre-loading symbols with `LD_PRELOAD`. For workloads that are classified as device-intensive, CUDA runtime calls are forwarded unaltered to the CUDA runtime. For data-transfer-intensive workloads, intercepted calls are forwarded to high-priority streams. The scheduler provides basic control over when data-transfer-intensive and device-intensive kernels execute concurrently. While this represents only the foundational mechanism for the data movement-aware scheduling described in Section 4, it enables us to demonstrate the benefits of treating the interconnect as a first-class schedulable resource.

Workloads We evaluate using two representative workloads:

Analytics Workload: We use the Star Schema Benchmark (SSB) at scale factor 100, with data resident in system DRAM. We continuously execute the same query type in a closed loop to maintain consistent compute-transfer patterns.

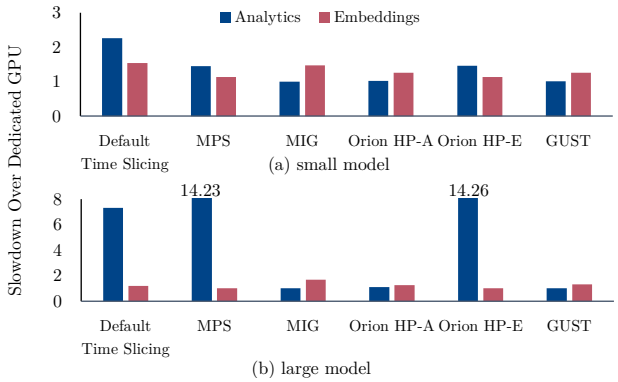

Figure 3: Performance of mixed analytics (SSB Q3.1) and inference workloads under GPU sharing on an A100 using the small (a) and large (b) embedding models.

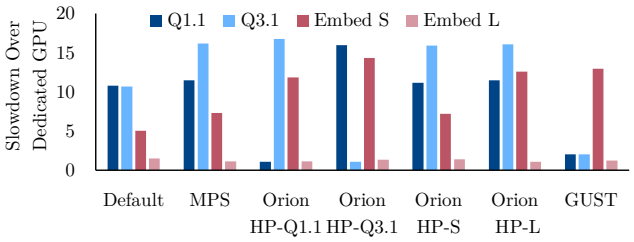

Figure 4: 4-W colocation performance (SSB Q1.1, Q3.1, embedding small and large) on an A100.

Inference Workload: We continuously generate embeddings from the English Wikipedia dataset using two models: multilingual-e5-large [33] (560M parameters, 1 GB GPU memory) which we refer to as *small*, and BGE-Multilingual-Gemma2 [5] (9.2B parameters, 17.2 GB GPU memory) which we refer to as *large*. The workload processes pre-tokenized text in batches for a fixed duration, allowing us to measure sustained throughput and latency. We use a batch size of 1024 and target token lengths of 512.

Baselines. We compare GUST with the four GPU sharing approaches described in Section 3: *Default Time Slicing*, *MPS* spatial sharing, *MIG* hardware partitioning, and *Orion* [29], an interference-aware scheduler that optimizes on-device resources but oblivious to PCIe data movement. We also measure *dedicated GPU performance*, where each workload runs in isolation (upper-bound baseline). *Orion* is designed to preserve the performance of a single high-priority (HP) task while maximizing the throughput of colocated best-effort (BE) jobs. It profiles kernels offline to classify them as compute- or memory-intensive, and only BE kernels with complementary classifications can be executed concurrently to HP kernels. Orion classifies SSB and inference kernels as memory- and compute-intensive, respectively. We separately report the results for Orion for each workload selected as high priority, denoted as HP-*task*. For *MIG*, we partition the A100 into two GPU instances, a *3g.40gb* instance and a *4g.40gb* instance, for analytics and embedding generation, respectively[1]. The V100S does not support MIG.

To evaluate the potential of transfer-aware GPU sharing, we colocate multiple analytics and embeddings workloads on a single GPU using different GPU sharing approaches. We calculate the workload slowdown over dedicated GPU performance using analytics' average latency (s) and embedding throughput (tokens/s).

Results: We first evaluate GUST by colocating a single embedding generation task with analytics query processing. Figure 3 shows the performance of SSB Q3.1 colocated with embedding generation on an A100 GPU, normalized to dedicated GPU execution.

[1]The A100 has 7 SM slices. A GPU instance is made up of 1 or more SM slices, so it is not possible to evenly split the SMs of an A100 using MIG.

With the small embedding model (Figure 3a), both GUST and Orion configured with analytics as the high-priority task (Orion HP-A) achieve minimal performance degradation, with geometric mean slowdowns of only 1.13×. This demonstrates that transfer-aware scheduling can effectively exploit the complementary resource demands of these workloads. In contrast, when Orion prioritizes embeddings (Orion HP-E), the geometric mean slowdown increases to 1.29×. This degradation occurs because prioritizing the device-intensive embeddings task limits analytics kernel scheduling opportunities, causing the data transfer pipeline to stall and leaving the PCIe bus underutilized.

The benefits of transfer-aware scheduling become even more pronounced with the large embedding model (Figure 3b). While GUST maintains a modest 1.16× slowdown, Orion HP-E results in a much larger geometric mean slowdown of 3.8×; analytics runs 14.26× slower than on a dedicated GPU. The large model's higher register pressure exacerbates the scheduling problem: analytics kernels are rarely scheduled, creating a cascading failure where stalled kernels block data transfers, leaving the PCIe bus underutilized.

These results validate our core insight: effective GPU sharing for heterogeneous workloads requires explicit awareness of data movement patterns. Transfer-oblivious schedulers, even sophisticated ones like Orion that optimize for on-device interference, fail to capitalize on the complementary resource demands of transfer-intensive and device-intensive workloads. By contrast, GUST's transfer-aware approach maintains high utilization of both interconnect and compute resources.

To evaluate GUST under higher contention, we colocate four concurrent workloads: two analytics queries (SSB Q1.1 and Q3.1) and two embedding generation tasks (using the small and large models). This configuration challenges schedulers to balance multiple transfer-intensive workloads competing for PCIe bandwidth while simultaneously managing device-intensive tasks. MPS and default time slicing result in over 10× slowdowns for both analytics tasks. Orion can prioritize one, but not both of the transfer-intensive analytics tasks. Orion achieves its best geometric mean slowdown across workloads of 3.9× with HP-Q1.1 and its worst slowdown of 7× with HP-L. In contrast, GUST achieves a 2.8× geometric mean slowdown by maintaining balanced progress across both analytics queries. By prioritizing analytics kernels that collectively saturate the PCIe bandwidth while interleaving device-intensive embedding kernels, GUST ensures that neither analytics query starves. This demonstrates that transfer-aware scheduling scales beyond

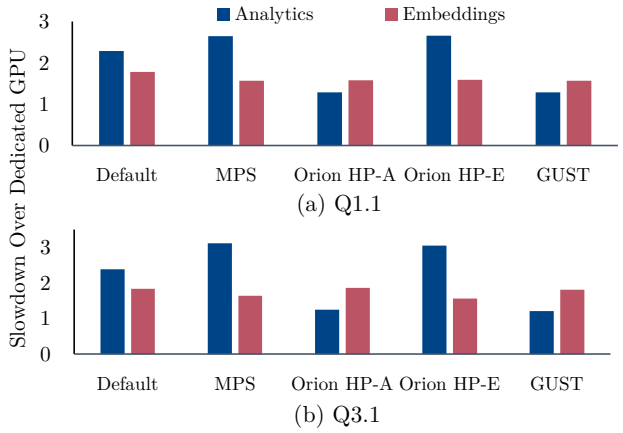

Figure 5: Performance of SSB Q1.1 (a), Q3.1 (b) with embedding generation using the large model on a V100S.

pairwise colocation, enabling efficient multiplexing of multiple workloads with diverse resource requirements.

Figure 5 shows colocation performance on V100S with the large embedding model paired with either Q1.1 or Q3.1. Q3.1 is more device-intensive than Q1.1 due to its multiple hash joins and group-bys. This difference manifests in embedding throughput: when colocated with Q3.1, both GUST and Orion HP-A achieve lower embedding throughput compared to colocation with Q1.1, as Q3.1's higher compute demands reduce available resources for inference.

Comparing identical workloads across GPU generations reveals significant differences in failure modes. For Q3.1 with the large embedding model, MPS and Orion HP-E achieve over 14× slowdowns for analytics slowdowns on the A100 (Figure 3b) versus just over 3× on the V100S (Figure 5b).

6 CONCLUSION AND OUTLOOK

This paper introduces GUST, a scheduler that treats PCIe bandwidth as a first-class resource. By monitoring workload interconnect intensity and intelligently interleaving data-transfer-intensive and device-intensive tasks, our approach achieves high utilization of both the interconnect and GPU compute resources. While our prototype uses time and brief isolated profiling windows as a coarse-grained proxy for resource utilization, finer-grained statistics would enable associating device utilization with each active context while kernels from multiple contexts are executing concurrently, avoiding the necessity of isolated profiling windows.

Thus far, our work on GUST has targeted PCIe, as it is the most prevalent GPU interconnect. However, today's flagship GPUs can also be connected to the host CPU with high-bandwidth interconnects, such as NVLink 5.0, which offers 900 GB/s of unidirectional bandwidth. Over time, such interconnects may become commodity hardware. While these interconnects reduce the data transfer bottleneck from host memory [10, 14], for many workloads, data does not reside in host memory to begin with. The bottleneck for query processing shifts elsewhere in the data path: to CPU cores pre-processing data, or to the path through PCIe-attached NICs or NVMe storage. Efficiently utilizing GPUs with high-bandwidth interconnects will therefore require co-scheduling transfers across the entire end-to-end data path, not just the final hop from host memory.

Our work opens several research directions. Transparent compression and decompression could further optimize interconnect utilization, particularly with hardware acceleration available on newer architectures, such as NVIDIA Blackwell. GPU Direct bypasses CPU memory, creating new scheduling challenges as data flows directly from NICs/NVMe drives to GPUs; the scheduler must account for network/storage I/O characteristics alongside interconnect bandwidth.

Data transfer awareness represents a fundamental shift in GPU resource management. As data platforms increasingly combine analytics, vector databases, and large language models, treating the interconnect as a schedulable resource becomes essential. By making interconnect bandwidth schedulable, we enable a new generation of data platforms that can seamlessly combine analytics and AI on shared GPU infrastructure.

REFERENCES

[1] Anastasia Ailamaki, Samuel Madden, Daniel Abadi, Gustavo Alonso, Sihem Amer-Yahia, Magdalena Balazinska, Philip A. Bernstein, Peter A. Boncz, Michael J. Cafarella, Surajit Chaudhuri, Susan B. Davidson, David J. DeWitt, Yanlei Diao, Xin Luna Dong, Michael J. Franklin, Juliana Freire, Johannes Gehrke, Alon Y. Halevy, Joseph M. Hellerstein, Mark D. Hill, Stratos Idreos, Yannis E. Ioannidis, Christoph Koch, Donald Kossmann, Tim Kraska, Arun Kumar, Guoliang Li, Volker Markl, Renée J. Miller, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Aditya G. Parameswaran, Ippokratis Pandis, Jignesh M. Patel, Andrew Pavlo, Danica Porobic, Viktor Sanca, Michael Stonebraker, Julia Stoyanovich, Dan Suciu, Wang-Chiew Tan, Shivaram Venkataraman, Matei Zaharia, and Stanley B. Zdonik. 2025. The Cambridge Report on Database Research. *CoRR* abs/2504.11259 (2025). doi:10.48550/ARXIV.2504.11259 arXiv:2504.11259

[2] Peter Bakkum and Kevin Skadron. 2010. Accelerating SQL database operations on a GPU with CUDA. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, Pittsburgh Pennsylvania USA, 94–103. doi:10.1145/1735688.1735706

[3] Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E. Gonzalez, Carlos Guestrin, and Matei Zaharia. 2024. Text2SQL is Not Enough: Unifying AI and Databases with TAG. doi:10.48550/arXiv.2408.14717

[4] Georgia Butler. 2025. AWS cuts costs for H100, H200, and A100 instances by up to 45%. https://www.datacenterdynamics.com/en/news/aws-cuts-costs-for-h100-h200-and-a100-instances-by-up-to-45/

[5] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. BGE M3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *CoRR* abs/2402.03216 (2024). doi:10.48550/ARXIV.2402.03216 arXiv: 2402.03216.

[6] Periklis Chrysogelos, Manos Karpathiotakis, Raja Appuswamy, and Anastasia Ailamaki. 2019. HetExchange: encapsulating heterogeneous CPU-GPU parallelism in JIT compiled engines. *Proceedings of the VLDB Endowment* 12, 5 (Jan. 2019), 544–556. doi:10.14778/3303753.3303760

[7] Patrick H. Coppock, Brian Zhang, Eliot H. Solomon, Vasilis Kypriotis, Leon Yang, Bikash Sharma, Dan Schatzberg, Todd C. Mowry, and Dimitrios Skarlatos. 2025. LithOS: An Operating System for Efficient Machine Learning on GPUs. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles, SOSP 2025, Lotte Hotel World, Seoul, Republic of Korea, October 13-16, 2025*, Youjip Won, Youngjin Kwon, Ding Yuan, and Rebecca Isaacs (Eds.). ACM, 1–17. doi:10.1145/3731569.3764818

[8] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*.

[9] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving dnns like clockwork: Performance predictability from the bottom up. In *14th USENIX symposium on operating systems design and implementation, OSDI 2020, virtual event, november 4-6, 2020*. USENIX Association, 443–462. https://www.usenix.org/conference/osdi20/presentation/gujarati

[10] Marko Kabic, Bowen Wu, Jonas Dann, and Gustavo Alonso. 2025. Powerful GPUs or Fast Interconnects: Analyzing Relational Workloads on Modern GPUs. *Proc. VLDB Endow.* 18, 11 (2025), 4350–4363. https://www.vldb.org/pvldb/vol18/p4350-

kabic.pdf
[11] 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 (2021), 1797–1804. doi:10.14778/3467861.3467869
[12] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *SOSP '23*. ACM, Koblenz Germany, 611–626. doi:10.1145/3600006.3613165
[13] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, and Gerardo Vitagliano. 2025. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing.
[14] Clemens Lutz. [n. d.]. Scalable Data Management using GPUs with Fast Interconnects. ([n. d.]).
[15] Sina Meraji, Berni Schiefer, Lan Pham, Lee Chu, Peter Kokosielis, Adam Storm, Wayne Young, Chang Ge, Geoffrey Ng, and Kajan Kanagaratnam. 2016. Towards a Hybrid Design for Fast Query Processing in DB2 with BLU Acceleration Using Graphical Processing Units: A Technology Demonstration. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, San Francisco California USA, 1951–1960. doi:10.1145/2882903.2903735
[16] NVIDIA. 2017. *NVIDIA Volta Architecture.* Architecture Whitepaper. NVIDIA. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
[17] NVIDIA. 2020. *NVIDIA Ampere Architecture.* Architecture Whitepaper. NVIDIA. https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.1.pdf
[18] NVIDIA. 2022. *NVIDIA H100 Architecture.* Architecture Whitepaper. NVIDIA. https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c
[19] NVIDIA. 2025. Multi-Process Service. https://docs.nvidia.com/deploy/mps/index.html
[20] NVIDIA. 2025. NVIDIA Multi-Instance GPU User Guide. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html
[21] Oracle. 2025. AI Vector Search. https://docs.oracle.com/en/database/oracle/oracle-database/23/nfcoa/ai_vector_search.html
[22] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. 2008. GPU Computing. *Proc. IEEE* 96, 5 (2008), 879–899. doi:10.1109/JPROC.2008.917757
[23] Liana Patel, Siddharth Jha, Melissa Pan, Harshit Gupta, Parth Asawa, Carlos Guestrin, and Matei Zaharia. 2025. Semantic Operators: A Declarative Model for Rich, AI-based Data Processing. doi:10.48550/arXiv.2407.11418
[24] Johns Paul, Shengliang Lu, and Bingsheng He. 2021. Database systems on GPUs. *Foundations and Trends® in Databases* 11, 1 (2021), 1–108. doi:10.1561/1900000076
[25] Aunn Raza, Periklis Chrysogelos, Panagiotis Sioulas, Vladimir Indjic, Angelos-Christos G. Anadiotis, and Anastasia Ailamaki. 2020. GPU-accelerated data management under the test of time. In *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, 2020*. www.cidrdb.org.
[26] Viktor Sanca, Manos Chatzakis, and Anastasia Ailamaki. 2024. Optimizing Context-Enhanced Relational Joins. In *ICDE*. IEEE, Utrecht, Netherlands, 501–515. doi:10.1109/ICDE60146.2024.00045
[27] Anil Shanbhag, Samuel Madden, and Xiangyao Yu. 2020. A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM, Portland OR USA, 1617–1632. doi:10.1145/3318464.3380595
[28] Snowflake. 2025. Snowflake Cortex. https://www.snowflake.com/content/snowflake-site/global/en/product/features/cortex
[29] Foteini Strati, Xianzhe Ma, and Ana Klimovic. 2024. Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 1075–1092. doi:10.1145/3627703.3629578
[30] The Linux Kernel Archives. 2025. *Accounting Subsystem Documentation.* https://docs.kernel.org/accounting/index.html Linux Kernel Documentation, Version 6.x.
[31] The Linux Kernel Archives. 2025. *CFS Scheduler Design.* https://www.kernel.org/doc/html/latest/scheduler/sched-design-CFS.html Linux Kernel Documentation, Version 6.x.
[32] The Linux Kernel Archives. 2025. *Scheduler API and Implementation.* https://www.kernel.org/doc/html/latest/scheduler/index.html Linux Kernel Documentation, Version 6.x.
[33] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Multilingual E5 text embeddings: A technical report. *CoRR* abs/2402.05672 (2024). doi:10.48550/ARXIV.2402.05672 arXiv: 2402.05672.
[34] Bowen Wu, Wei Cui, Carlo Curino, Matteo Interlandi, and Rathijit Sen. 2025. Terabyte-Scale Analytics in the Blink of an Eye. doi:10.48550/arXiv.2506.09226
[35] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *OSDI*, Andrea C. Arpaci-Dusseau and Geoff Voelker (Eds.). USENIX Association, 595–610. https://www.usenix.org/conference/osdi18/presentation/xiao
[36] 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 (2025), 1250–1263. https://www.vldb.org/pvldb/vol18/p1250-yuan.pdf
[37] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. 2024. LLM Inference Unveiled: Survey and Roofline Model Insights. doi:10.48550/arXiv.2402.16363
[38] Lixi Zhou, Qi Lin, Kanchan Chowdhury, Saif Masood, Alexandre Eichenberger, Hong Min, Alexander Sim, Jie Wang, Yida Wang, Kesheng Wu, Binhang Yuan, and Jia Zou. 2024. Serving Deep Learning Models from Relational Databases. OpenProceedings.org. doi:10.48786/EDBT.2024.61