

ACM SIGOPS ASIA-PACIFIC WORKSHOP ON SYSTEMS

Unlocking Software-defined GPU Fabric Scheduling in the LLM Era

Danyang Chen^{1*} Yufeng Gu^{2*} Yibo Huang^{2†} Chengxuan Pei¹
Peichun Hua¹ Yang Zhou³ Yunming Xiao¹

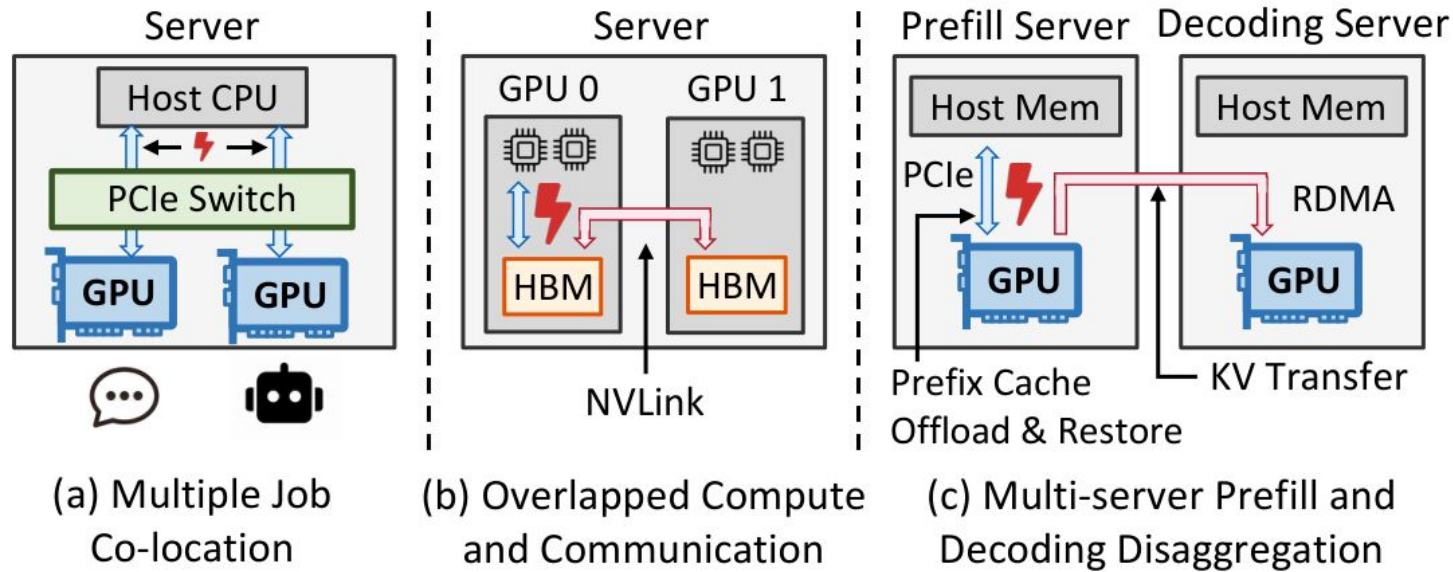
¹ CUHK-Shenzhen ² University of Michigan ³ UC Davis

APSys 2026

* Equal contribution † Corresponding author

MOTIVATION

LLM optimizations share the GPU fabric



Job co-location

Different jobs issue transfers through shared GPU and NIC paths.

Compute + NVLink

SM memory accesses and NVLink copies both consume HBM bandwidth.

PD + KV offload

The prefill worker sends KV to decode while moving cached KV to CPU.

Overlapping operations still compete for the same physical resources.

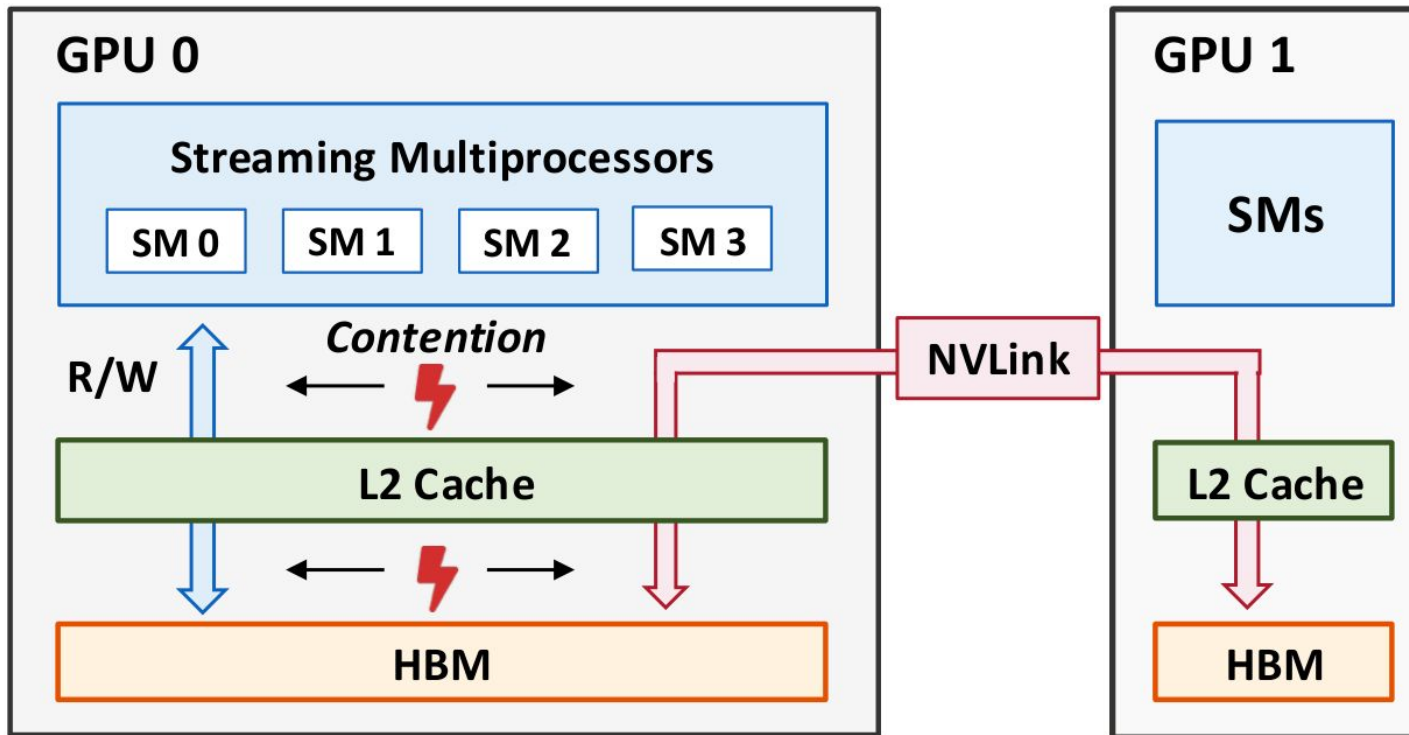
THE GAP

Fabric service can conflict with application urgency

Hardware arbitration lacks application context

Critical KV transfers share resources with background offloading

GPU memory contention



Two sources of traffic

SM memory R/W and NVLink copies share L2 and HBM access.

Controlled experiment

DGX-H100, separate CUDA streams, bidirectional peer copies.

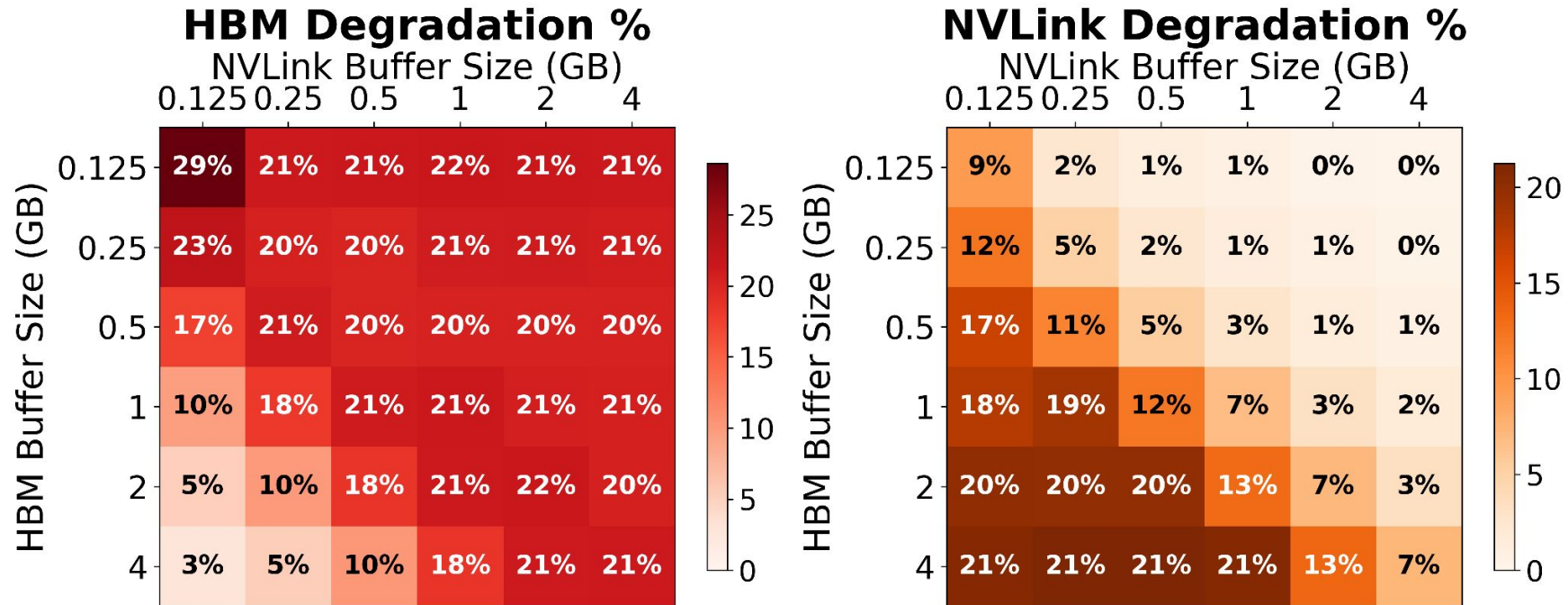
Three runs per size pair

HBM alone, NVLink alone, then both together. Each buffer: 128 MB–4 GB.

$$\text{Bandwidth loss (\%)} = 100 \times (1 - \text{contended bandwidth} / \text{isolated bandwidth})$$

Overlap exposes asymmetric bandwidth loss

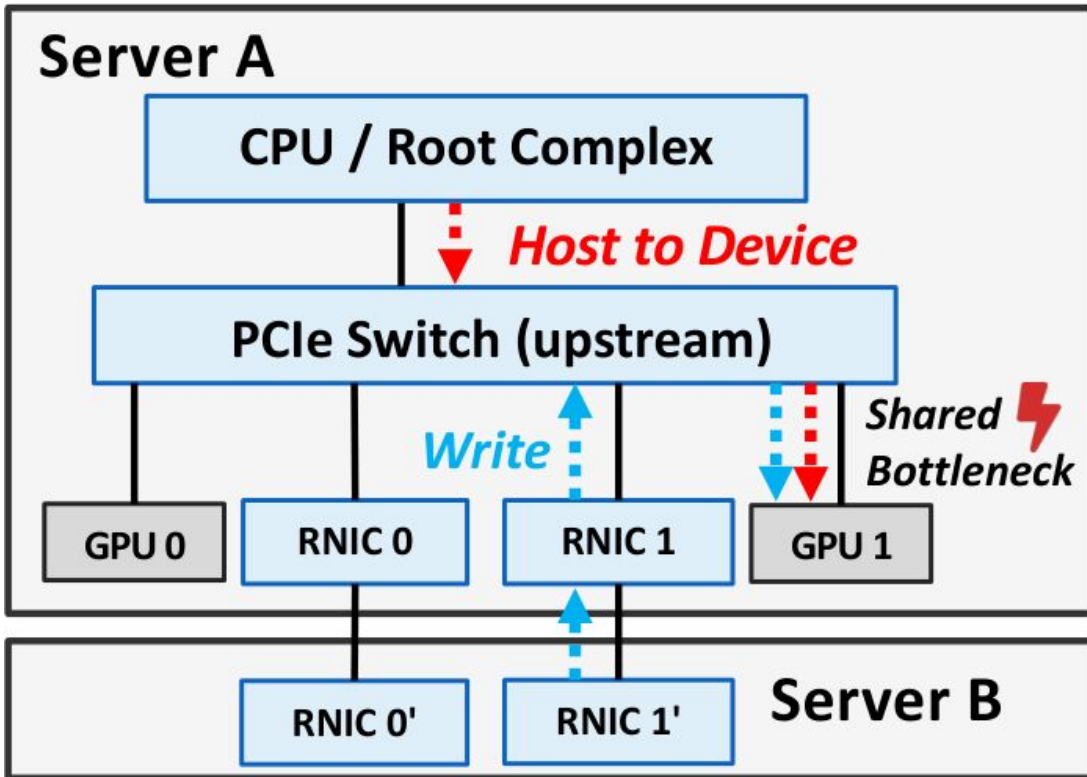
Rows and columns vary the two buffer sizes independently. Each cell shows bandwidth loss (%).



≈20% HBM loss under sustained overlap

Matched sizes ≥ 256 MB: NVLink typically loses 5–7%. Shorter overlap lowers the penalty.

PCIe contention



Traffic entering the GPU

Remote GDR write + local H2D copy share the receiving GPU's PCIe segment.

Traffic leaving the GPU

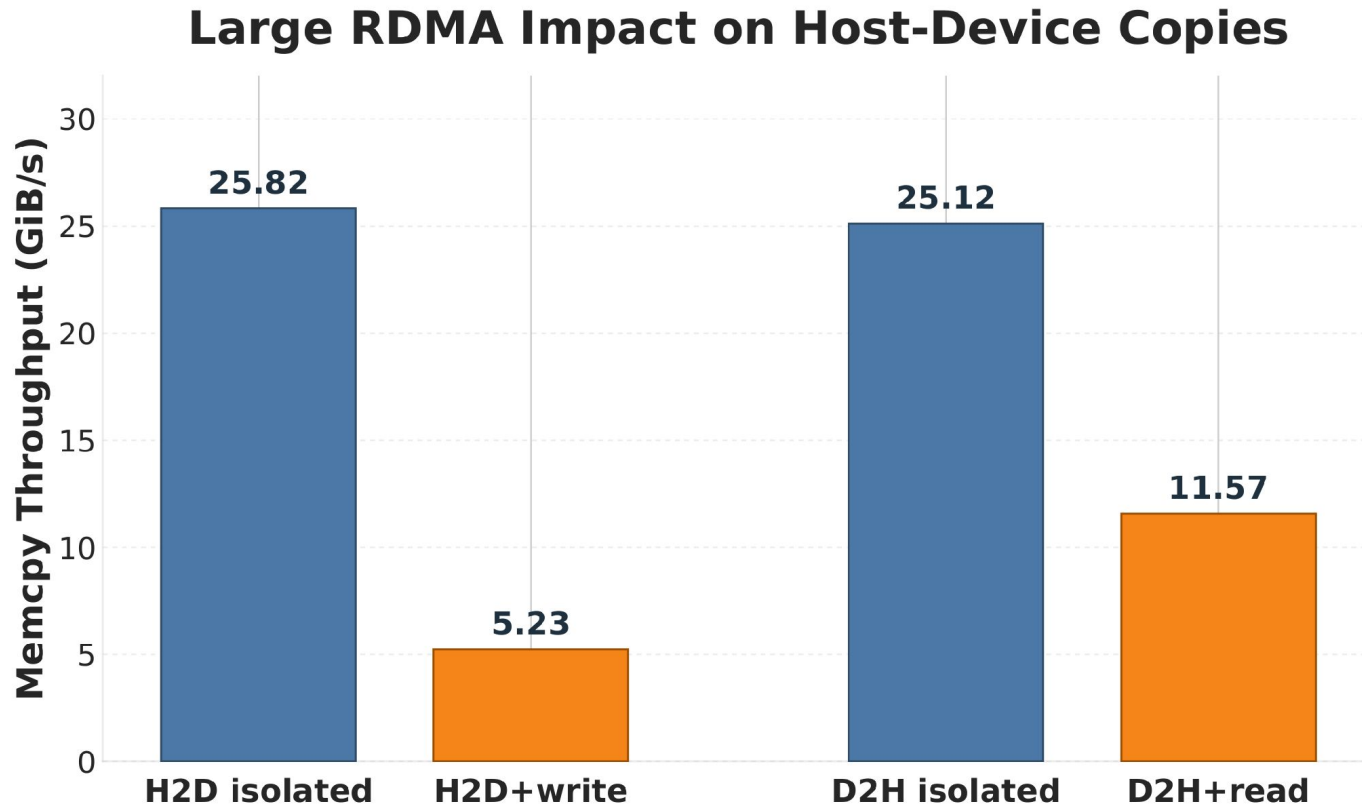
Remote GDR read + local D2H copy use that segment in the reverse direction.

Controlled experiment

Server A runs sustained copies from pinned host memory. Server B issues GDR requests.

Measure each stream alone and with its competitor, then compare bandwidth in GiB/s.

Large RDMA transfers throttle host-device copies



79.8%

H2D loss with GDR write
25.82 to 5.23 GiB/s

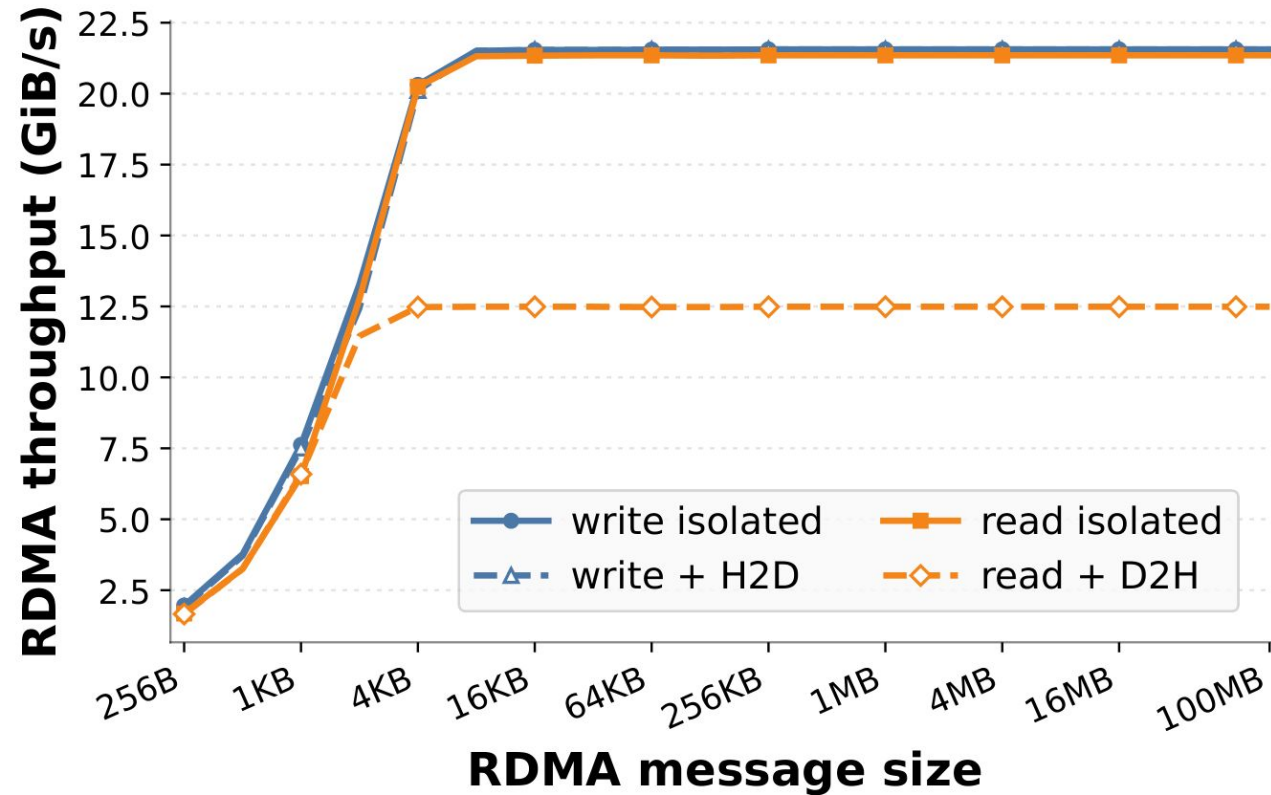
53.9%

D2H loss with GDR read
25.12 to 11.57 GiB/s

Loss uses each copy's own baseline. Example: H2D loss = $100 \times (1 - 5.23 / 25.82) \approx 79.8\%$.

PCIE RESULTS

Host copies affect RDMA reads and writes differently



≈41.5%

RDMA read loss with D2H
Messages ≥8 KB

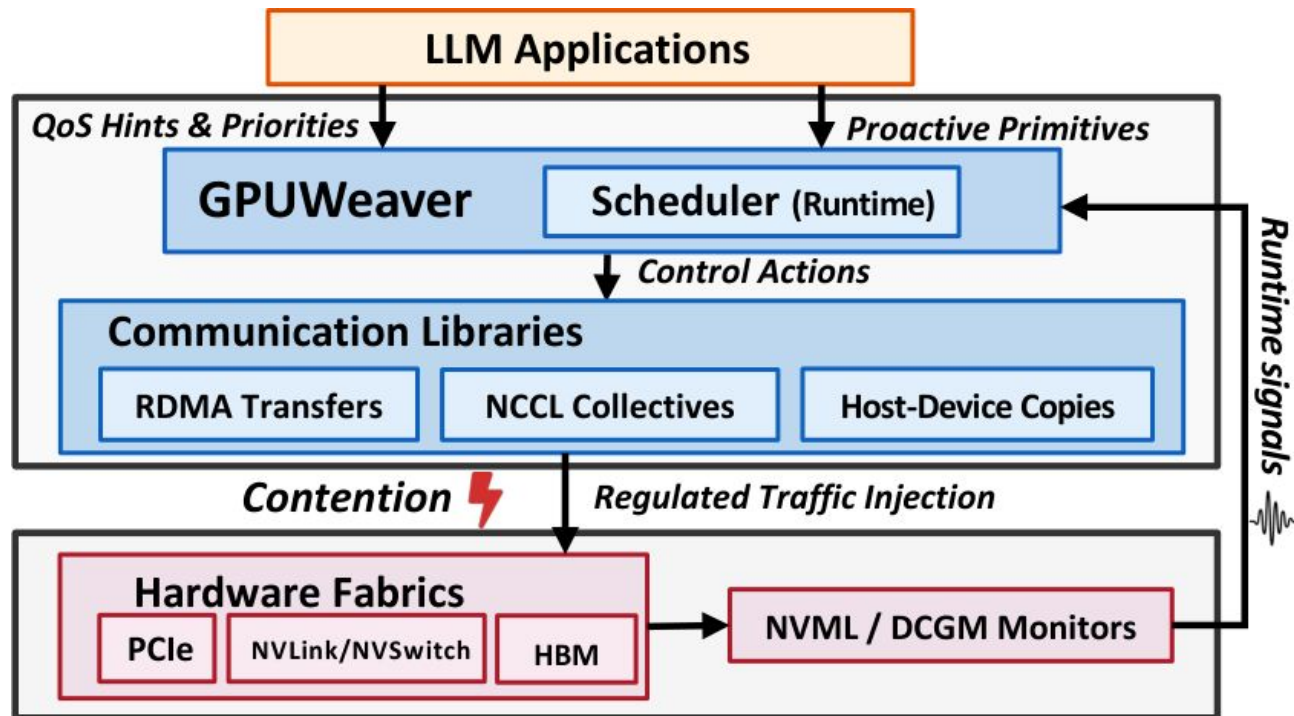
Little change

RDMA write with H2D

Compare write with write + H2D, and read with read + D2H. Each pair uses its own baseline.

FRAMEWORK

GPUWeaver connects flow goals to software controls



Application input

Register an RDMA transfer, collective, or host copy. Attach a priority or latency/bandwidth target.

Scheduler decision

Combine fabric pressure with per-flow progress. Output a bandwidth budget or pacing decision.

Enforcement

Communication libraries change how soon and how fast the selected flow issues traffic.

Framework design: PCIe, NVLink, and HBM. Evaluated serving prototype: PCIe offload control.

Runtime signals and control mechanisms

Signal source	Concrete measurements	What the scheduler learns
Shared fabric	PCIe / NVLink counters, NIC throughput and queue depth, HBM utilization	Which shared resource is under pressure
Application flow	Request / stage / transfer latency and per-flow backlog	Which flow misses its goal, and which work is waiting

RDMA traffic

Service levels, NIC QoS, and rate limits control how fast traffic enters the fabric.

Host-device copies

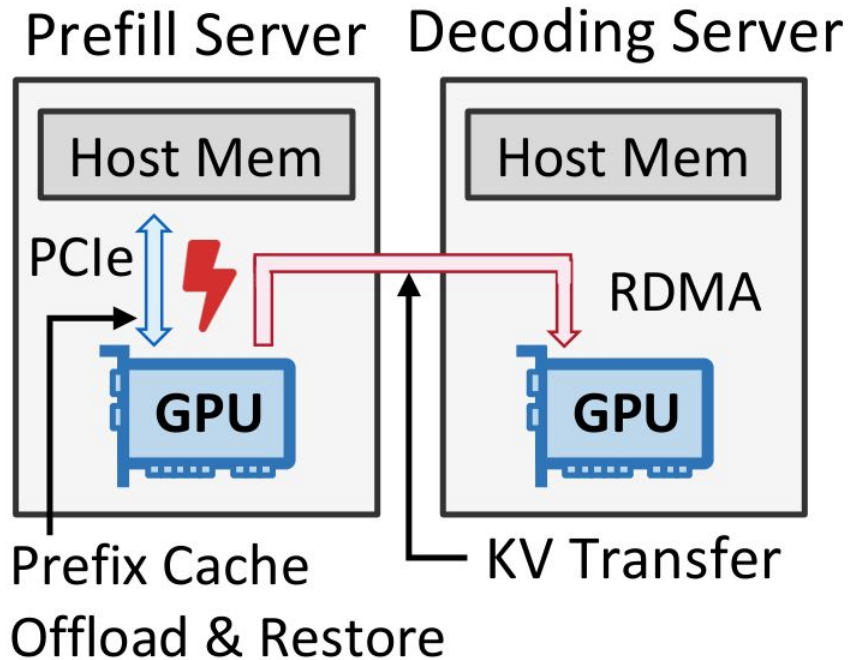
CUDA has no native bandwidth limiter. Delay launches, split copies into chunks, or add yield points.

Case-study mechanism: inject delay into background KV offload to change its PCIe demand.

PROTOTYPE

Two KV flows compete at the prefill GPU

vLLM + Mooncake, Qwen3-8B. Two logical servers, one GPU per server.



kv transfer / latency-critical

New KV moves from prefill GPU to decode GPU through GPUDirect RDMA Write. Decode needs this KV to serve the request.

kv offload / background

Prefix KV moves from prefill GPU to local CPU memory through D2H copies, competing with the KV handoff.

RDMA Write reads the sending GPU's memory. Both flows therefore consume the prefill GPU's PCIe path.

The prototype adjusts offload delay between rounds

01 Start a protected window

The runtime registers both KV flows. A PD signal starts protection for the KV transfer.

02 Read the previous round

Collect PCIe bandwidth, RDMA bandwidth, TTFT, and the amount of offload work waiting.

03 Update the next round's delay

RDMA below target: add more offload delay.

PD transfer healthy: reduce delay and return bandwidth to offload.

04 Apply the delay to background copies

Later offload launches reduce PCIe overlap and free bandwidth for the KV transfer.

The primary prototype knob is offload delay. The evaluation selects among candidate allocations.

Evaluation workloads and policy selection

Prompt tokens	Concurrent requests	Total KV
8K	1	1.2 GB
16K	1	2.4 GB
16K	2	4.7 GB
16K	3	7.1 GB
8K	8	9.4 GB

Default baseline

Run each request shape with natural overlap between KV transfer and offload.

GPUWeaver policy

Sweep candidate PCIe allocations. Select using RDMA bandwidth and TTFT from prior runs.

Comparison

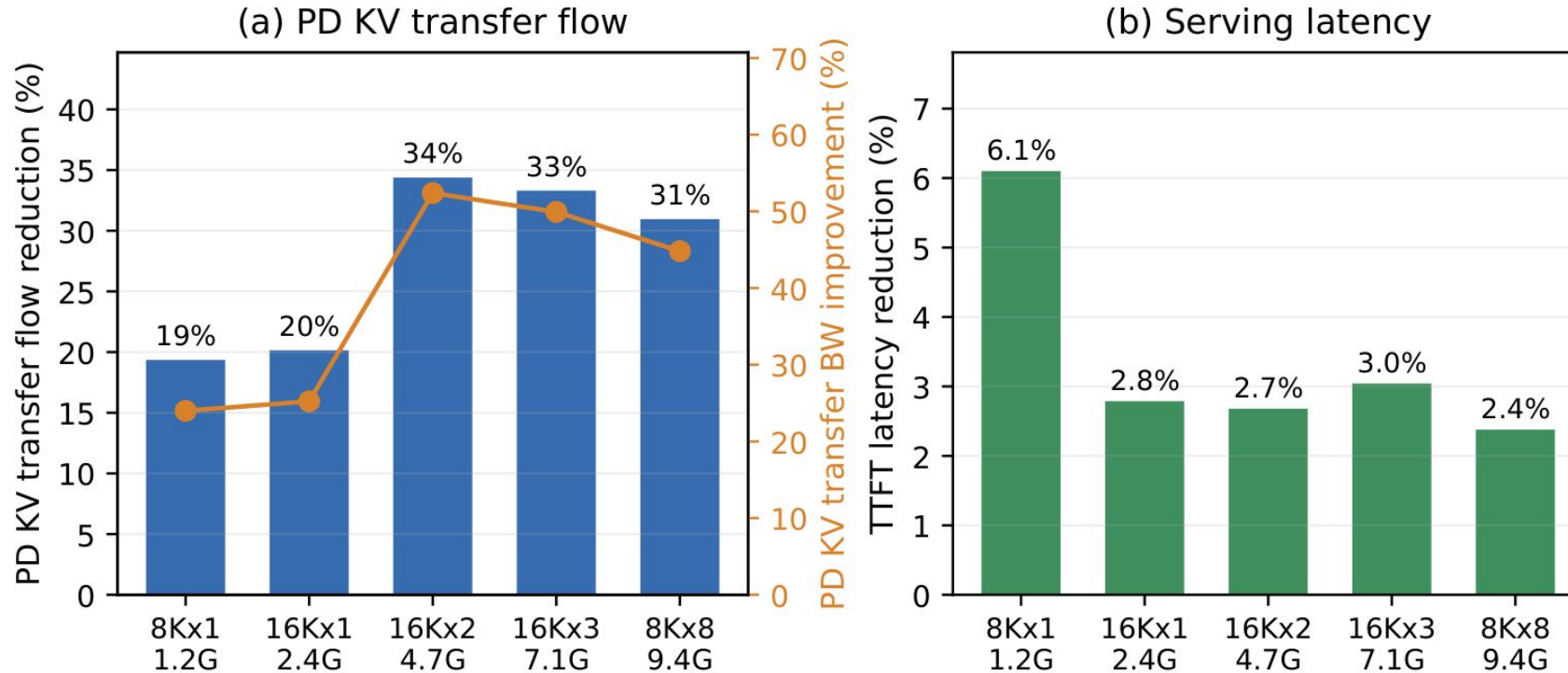
Compare KV transfer flow time and mean TTFT with the default policy for the same shape.

Request shapes come from a real Codex / SWE-bench Pro trace. Faster online adaptation remains future work.

SERVING RESULTS

KV transfers finish sooner and reduce mean TTFT

Default baseline: KV transfer and offload overlap naturally. Lower times are better.



19.4–34.4% shorter KV transfer

2.4–6.1% lower mean TTFT

16K × 2 requests, 4.7 GB KV: about 34% shorter transfer and 2.7% lower mean TTFT.

TTFT includes work beyond KV transfer, so the transfer improvement contributes a smaller total latency gain.

Scope and next steps

- | | | |
|-----------|--------------------------|---|
| 01 | Measured behavior | HBM / NVLink overlap reduces memory bandwidth. PCIe interference differs across reads, writes, and copies. |
| 02 | Implemented case | Two registered KV flows, PD handoff protection, and offload delay. Candidate selection uses prior-run measurements. |
| 03 | Next steps | Adapt allocations faster online, balance background progress, and extend control to NVLink / HBM contention. |

TAKEAWAYS

GPU fabrics deserve application-aware scheduling

GPUWeaver

Regulate traffic injection using application goals and runtime fabric signals

PCIe case study

19.4–34.4% shorter KV transfer time and 2.4–6.1% lower mean TTFT

Thank you

Measurement details

- | | | |
|-----------|----------------------|---|
| 01 | HBM / NVLink | 6 × 6 buffer-size sweep, 128 MB–4 GB
Bidirectional cudaMemcpyPeerAsync, separate CUDA streams |
| 02 | PCIe | Pinned host memory and sustained cudaMemcpyAsync
Remote GDR read / write on a two-server setup |
| 03 | Normalization | $\text{Loss (\%)} = 100 \times (1 - \text{concurrent BW} / \text{isolated BW})$ |

What the results establish

- | | | |
|-----------|------------------------------------|---|
| 01 | Why a smaller TTFT gain? | PD KV transfer occupies only part of the TTFT path |
| 02 | A general online scheduler? | The current evaluation selects allocations using prior runs |
| 03 | Fairness and portability? | Long-term background progress and other platforms need evaluation |