Enabling Efficient GPU Communication over Multiple NICs with FuseLink

Group: Haiquan Wang, Tonghuan Xiao and Jiahui Tan

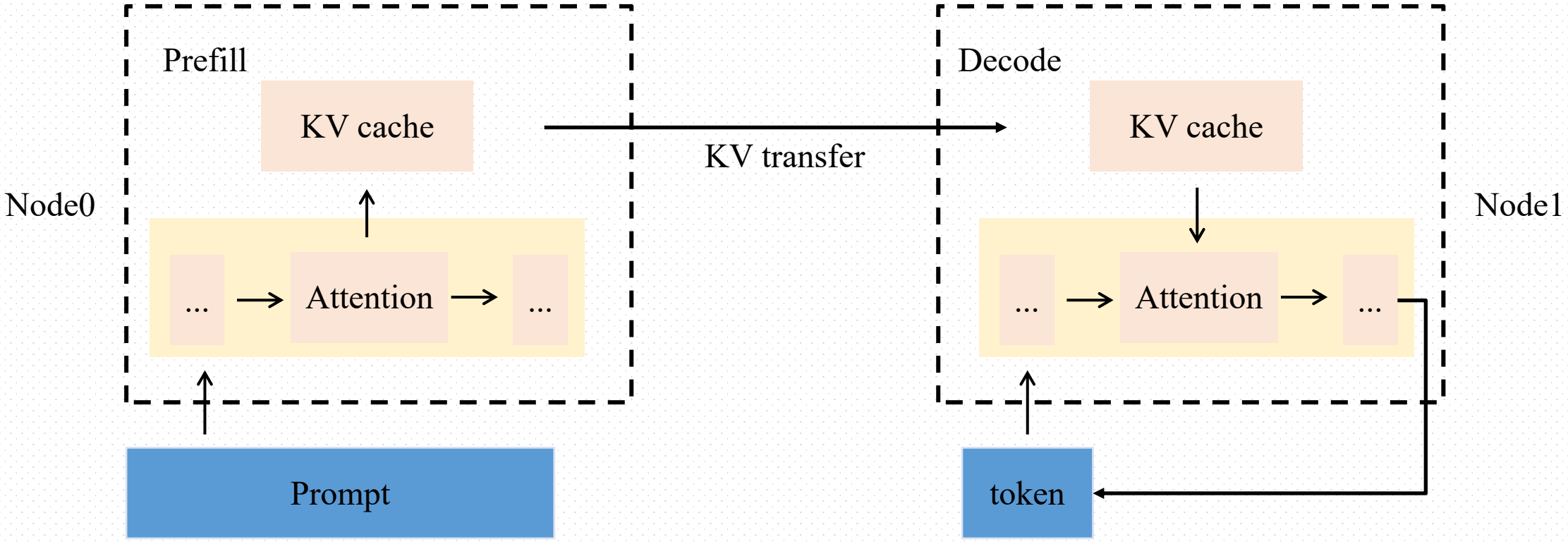
2025-11-04

Agenda

- ❑Background
- ❑Motivation
- ❑Opportunities and Challenges
- ❑Design
- ❑Evaluations
- ❑Conclusion

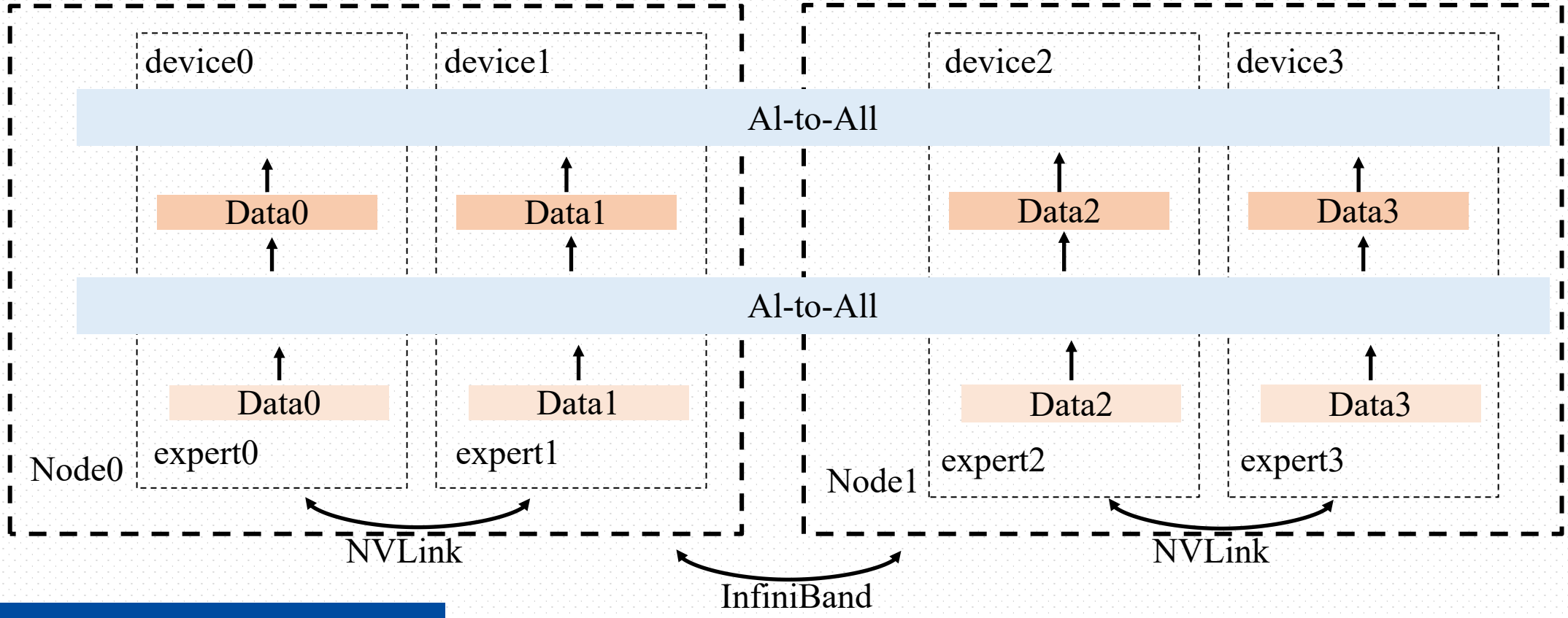
Background

- Inter-server communication is everywhere in large scale machine learning tasks
  - PD disaggregation: KV cache transmission



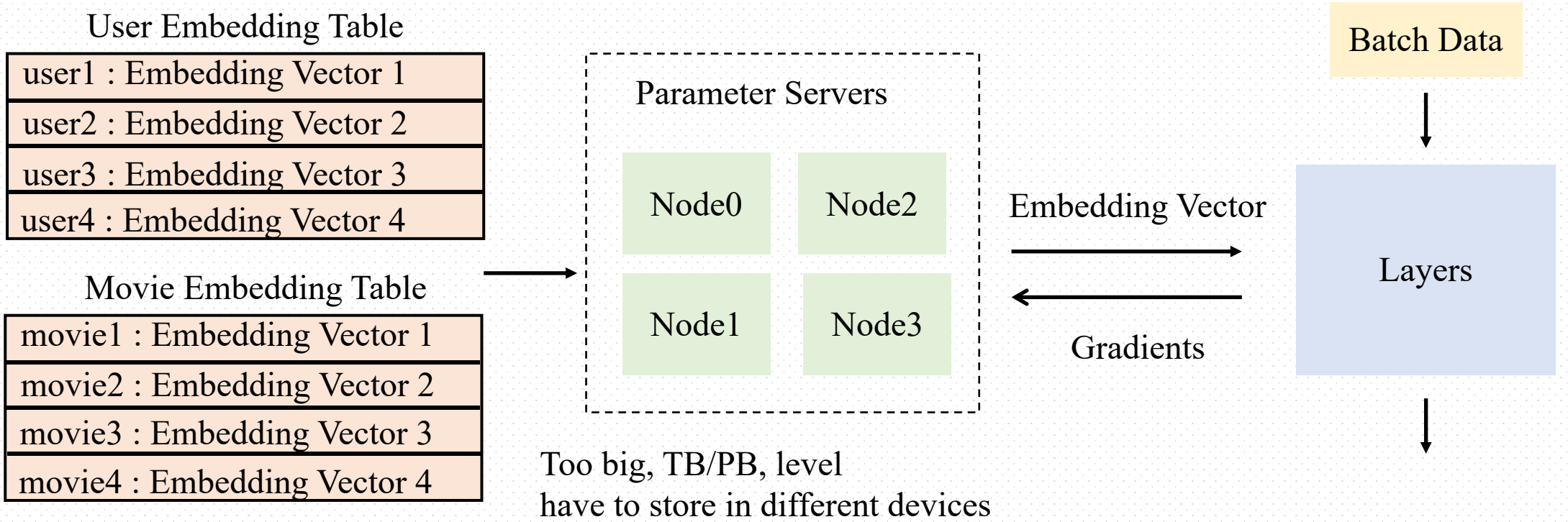
Background

- Inter-server communication is everywhere in large scale machine learning tasks
  - MoE model training: token dispatch and combine in expert parallel



Background

- **Inter-server communication is everywhere in large scale machine learning tasks**
  - **Recommend System Embedding Table Traffic**

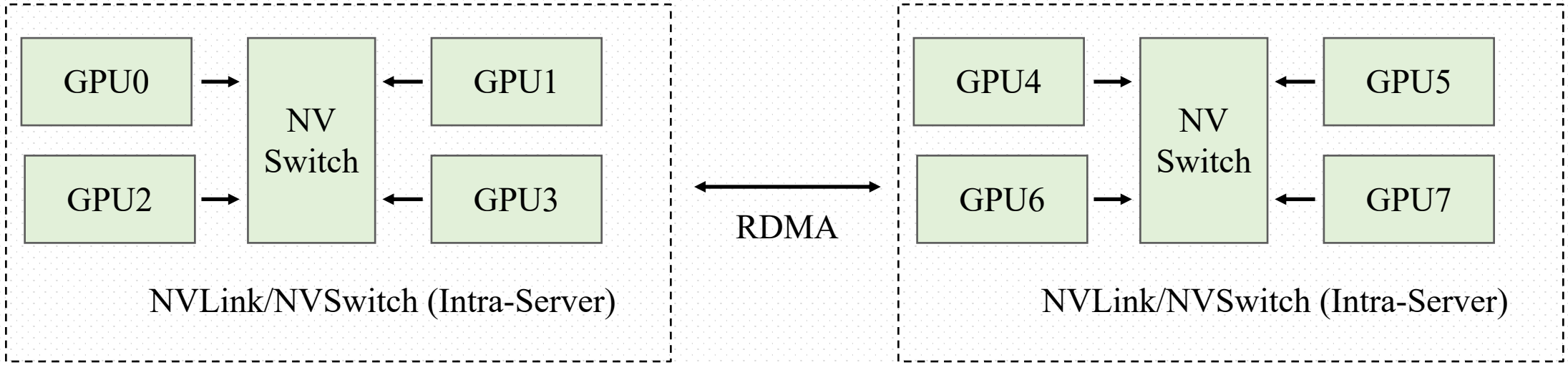


Background

- Distributed ML tasks
  - PD disaggregation: KV cache transmission
  - MoE model training: token dispatch and combine in expert parallel
  - Recommendation model training: embedding table traffic
- Efficient communication among GPUs is crucial

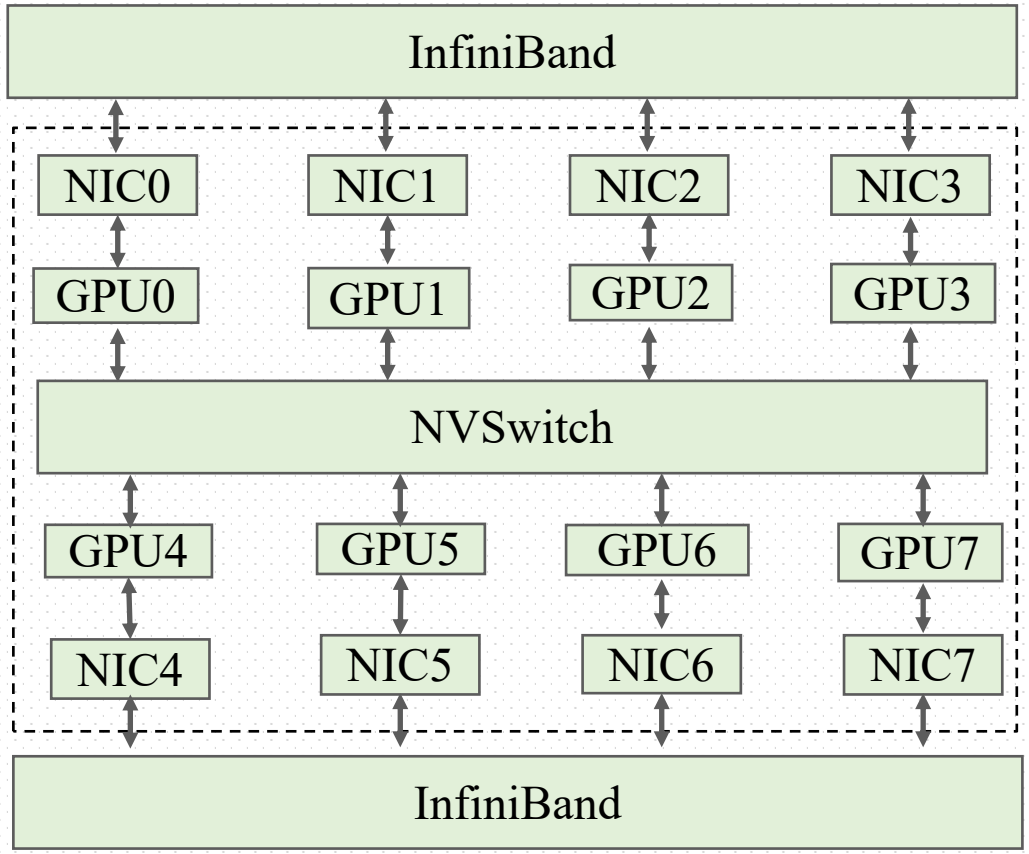
Background

- **Key Hardware Technology: Accelerating Data Exchange**
  - **NVLink/NVSwitch: intra-server**
  - **RDMA: inter-server**



Background

- The bandwidth of single NIC is much lower than NVLink
- How about stack multiple network interface cards ?



Dedicated Fabrics
~TB/s

PCIe ~ 50 - 100 Gbps

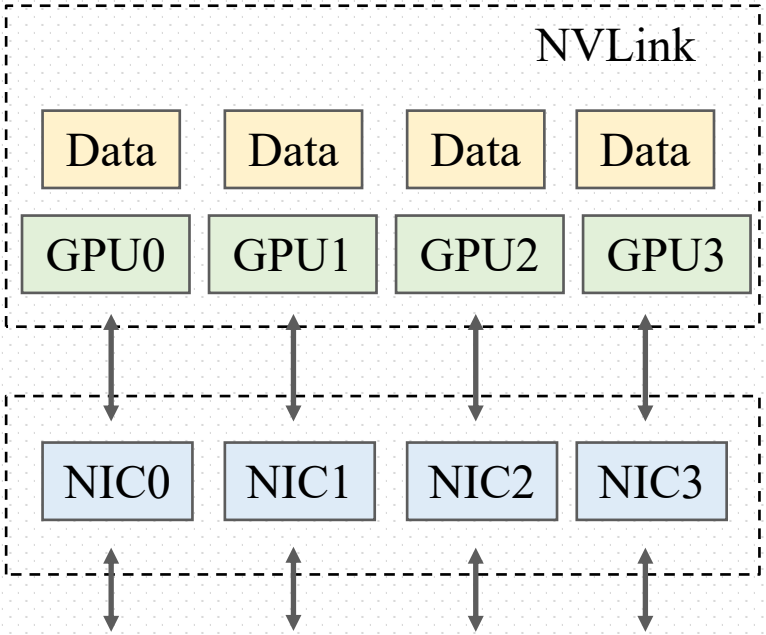
sigle NIC 200Gbps

Agenda

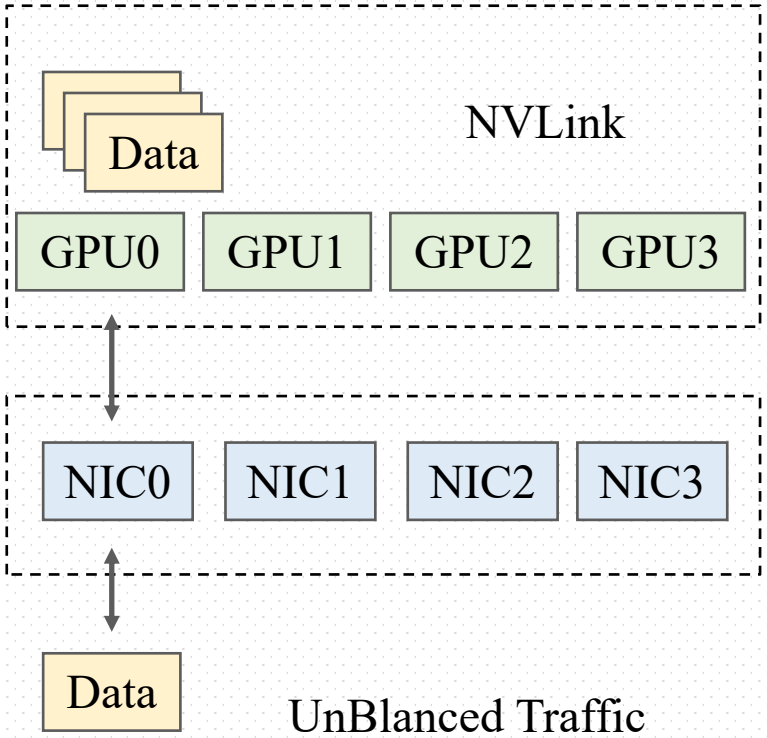
- ❑Background
- ❑Motivation
- ❑Opportunities and Challenges
- ❑Design
- ❑Evaluations
- ❑Conclusion

Motivation

- Data Distribution lead to resource waste
  - Blanced Traffic & Full Utillzation
  - UnBlanced Traffic & Underutillzed



Blanced Traffic

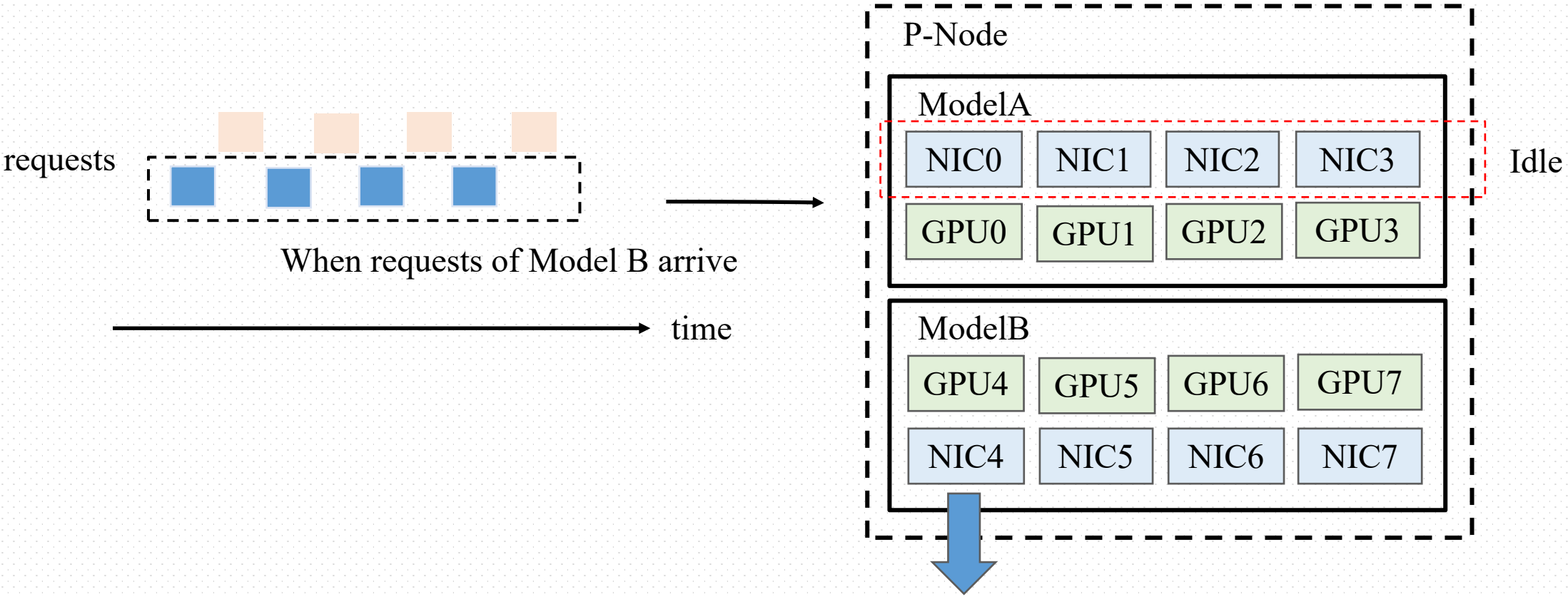


UnBlanced Traffic

Motivation

❑ PD disaggregation: KV cache transmission

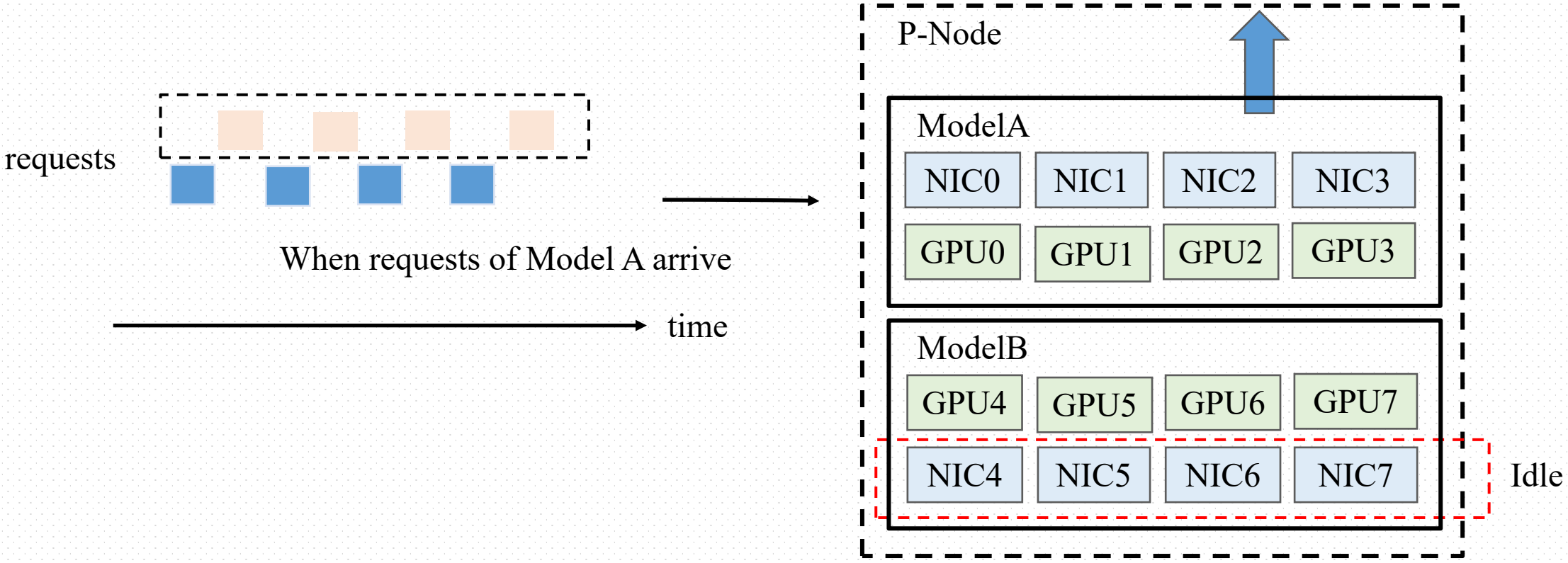
❑ Stochastic Task Arrival & multiple model instance



Motivation

❑ PD disaggregation: KV cache transmission

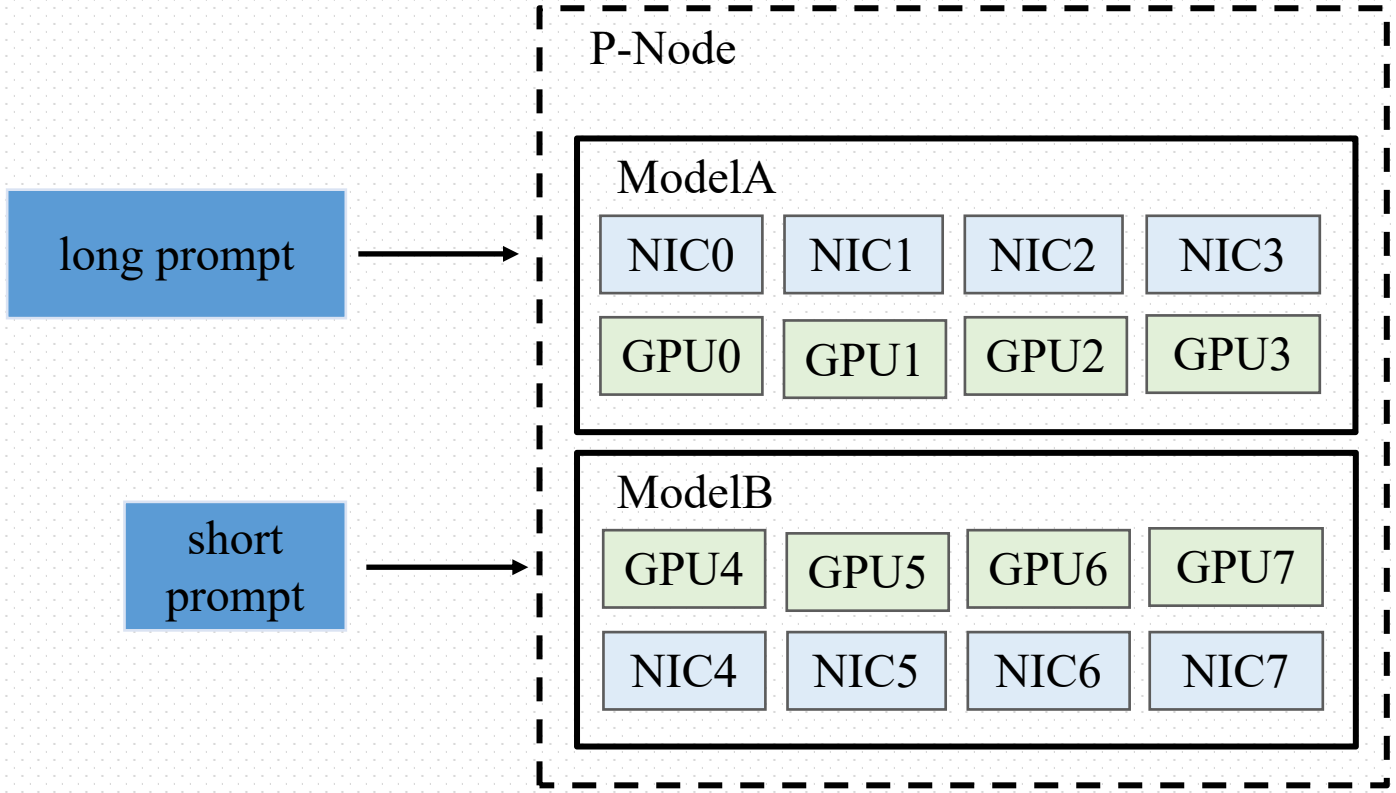
❑ Stochastic Task Arrival & multiple model instance



Motivation

❑ PD disaggregation: KV cache transmission

❑ Varying Requests Size

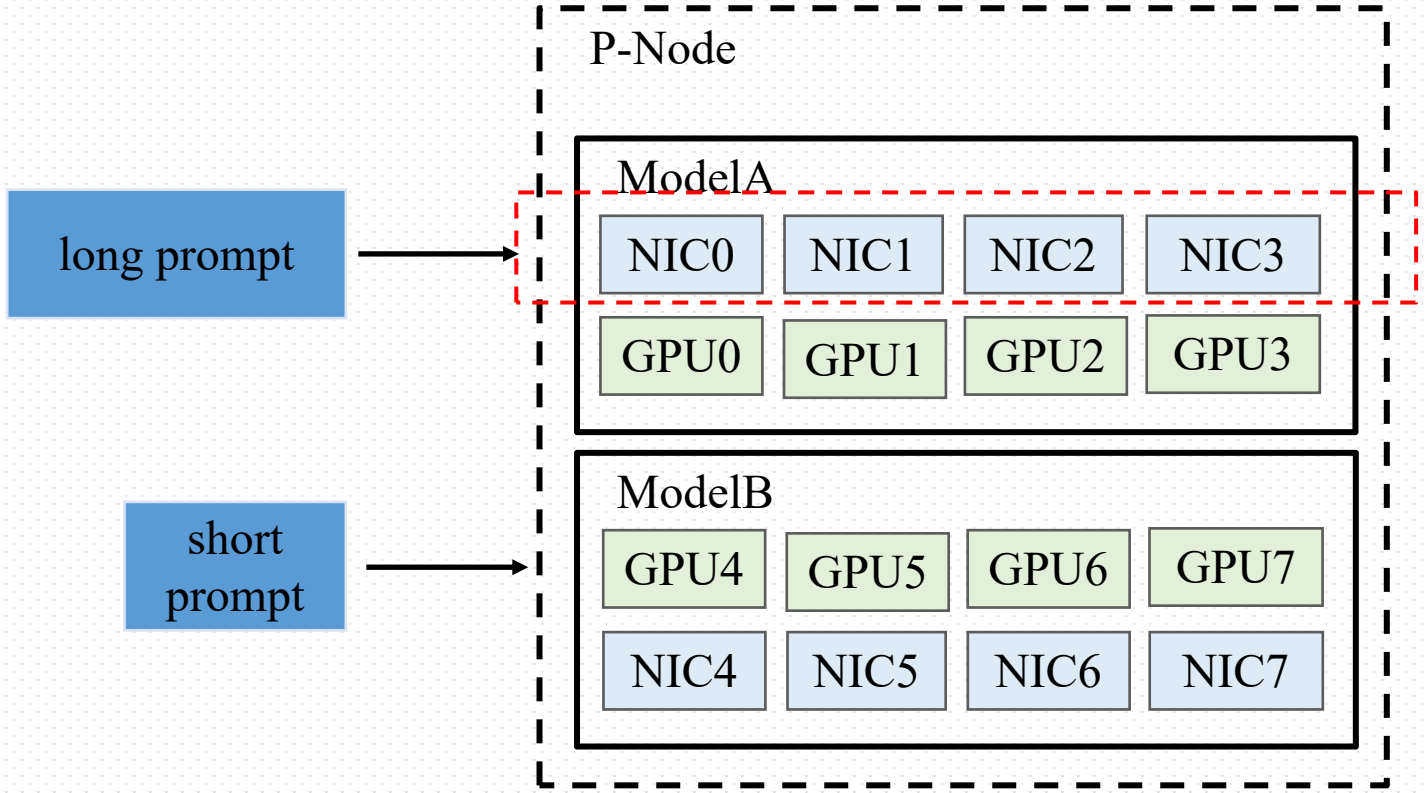


Assume A and B arrive at the same time

Motivation

❑ PD disaggregation: KV cache transmission

❑ Varying Requests Size

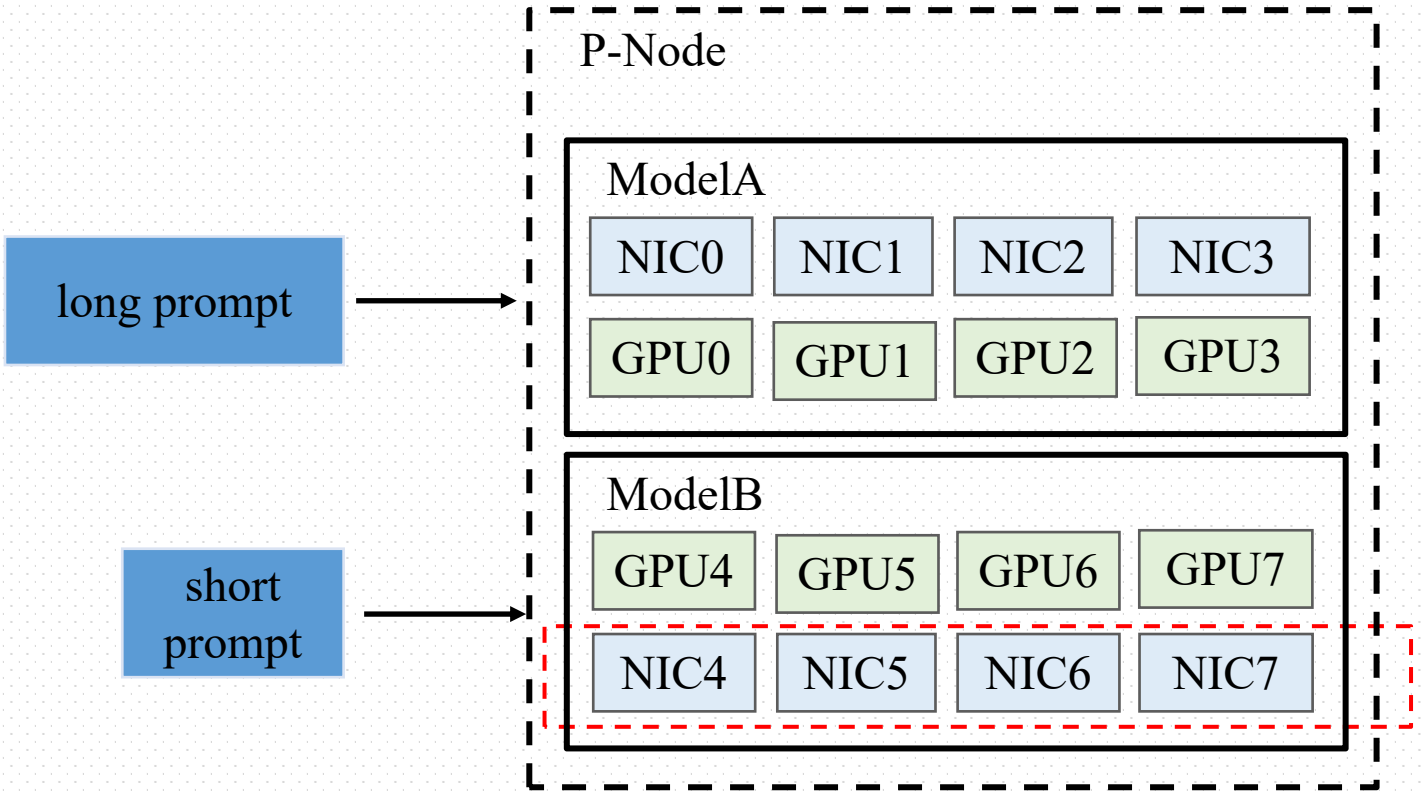


computation of Model B is completed first.
NICs 4-7 busy, NICs 0-3 idle

Motivation

❑ PD disaggregation: KV cache transmission

❑ Varying Requests Size

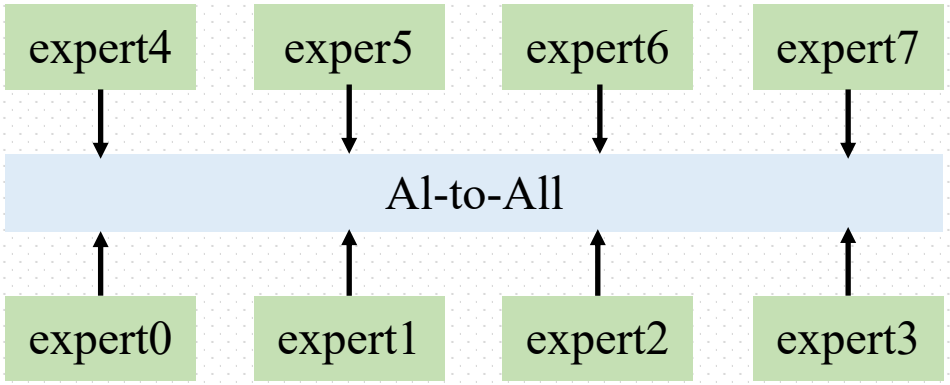


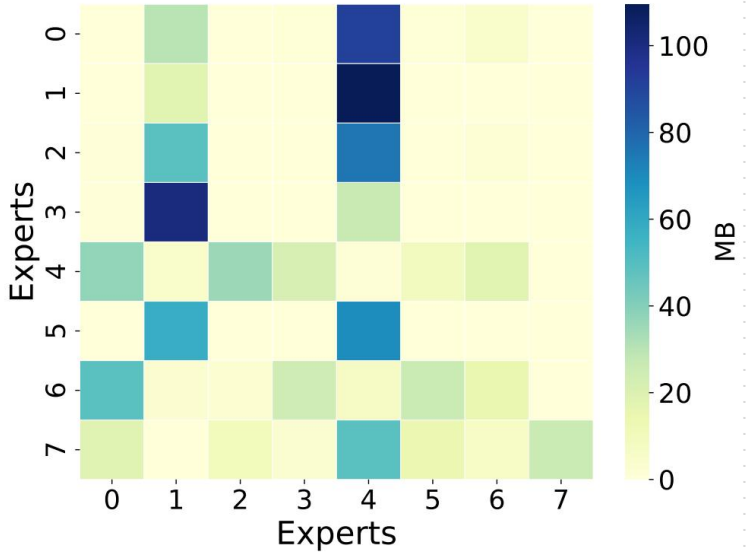
Model B is finished transferring.
Model A is not yet finished.

NICs 0-3 busy, NIC4-7 Idle

Motivation

- MoE model training: token dispatch and combine in expert parallel
  - Different Token Batches



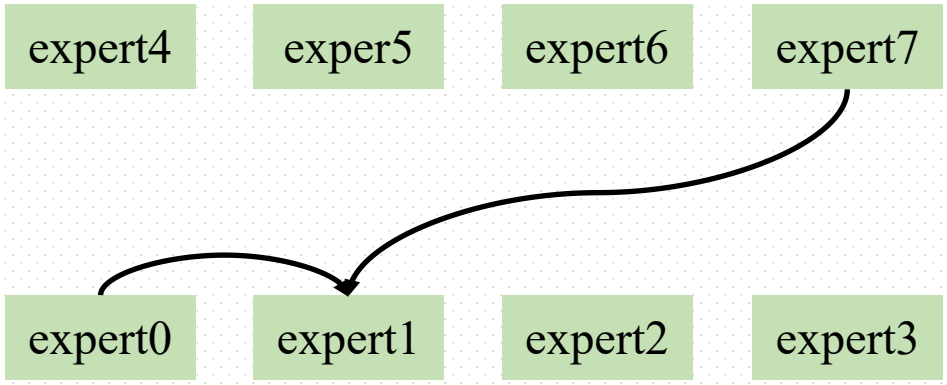


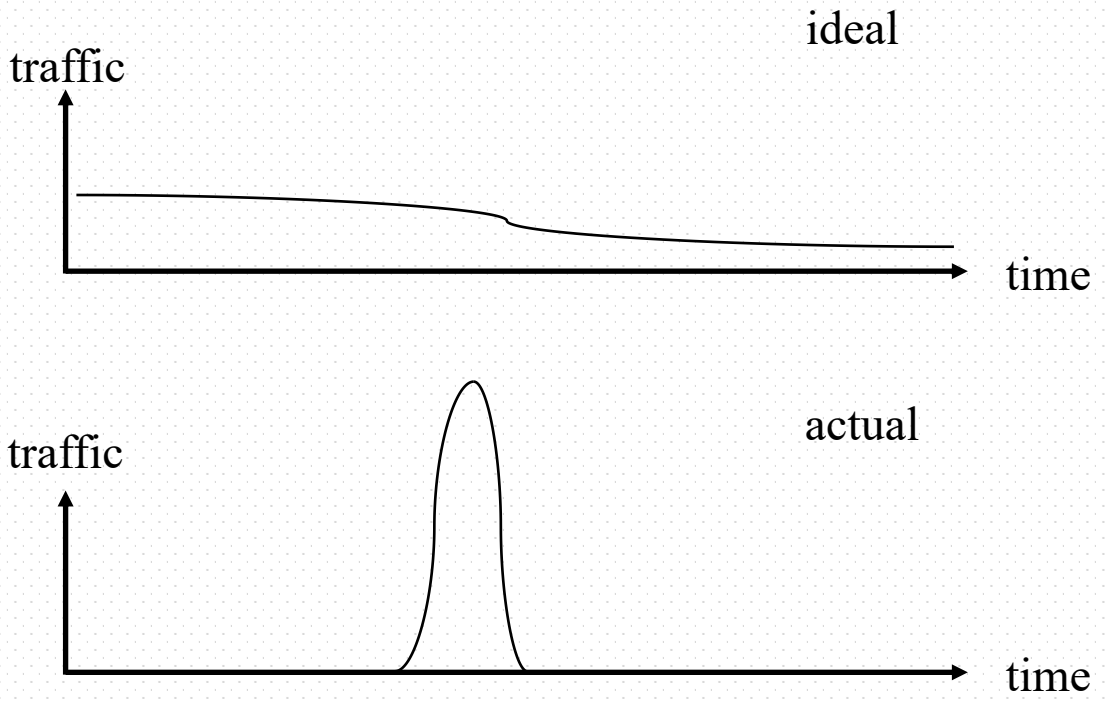
Motivation

❑ MoE model training: token dispatch and combine in expert parallel

❑ Asynchronous Transmission

Multiple, concurrent data overwhelm and congest a single NIC.

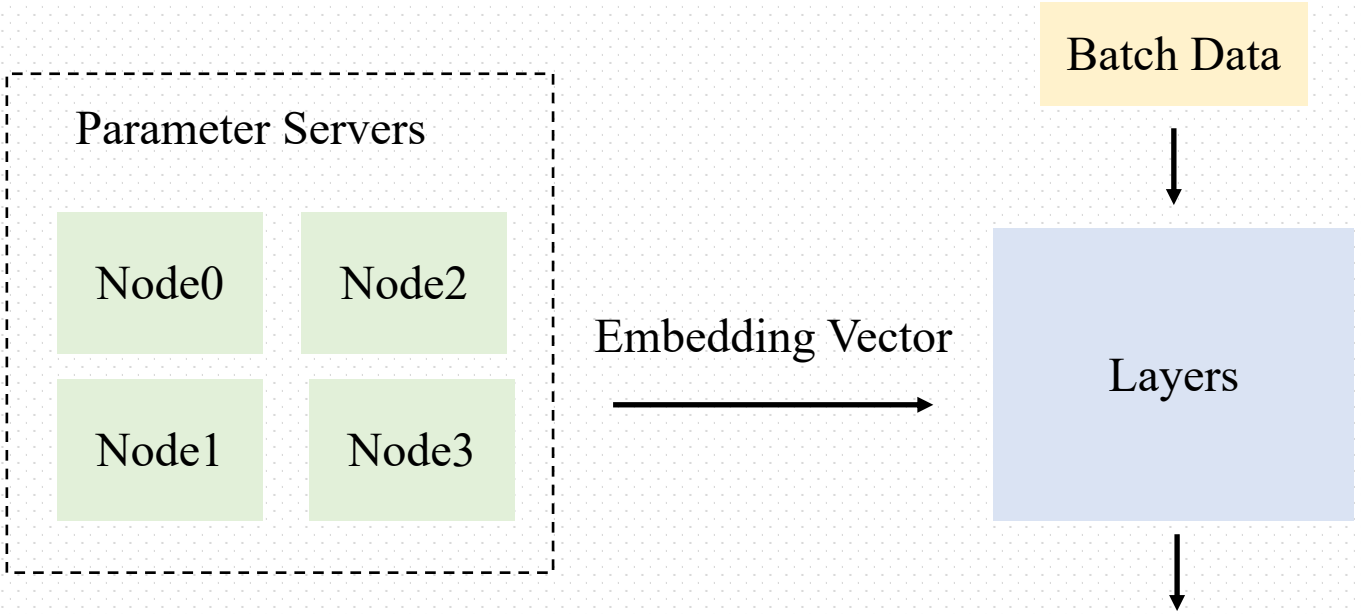


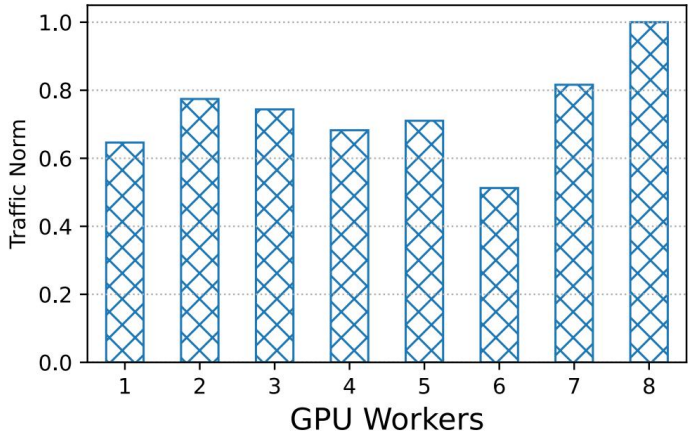


Motivation

❑ Recommend System Embedding Table Traffic

❑ Different Embeddings





Motivation

- Inter-server communication Dynamic traffic in large scale machine learning tasks
  - PD disaggregation: KV cache transmission
  - MoE model training: token dispatch and combine in expert parallel
  - Recommend System Embedding Table Traffic

ML Tasks	Communication Pattern | Cause			NIC util.	Comm. ratio
	via direct NICs	Concurrent transmission	Same traffic volume		
Distributed LLM Serving	✓	✗ | stochastic task arrival	✗ | varying requests sizes	13%-53%	11%-82%
Expert-parallel MoE	✓	✗ | async. transmission	✗ | different token batches	29%-65%	15%-42%
DLRM	✓	✓	✗ | different embeddings	59%-82%	28%-55%

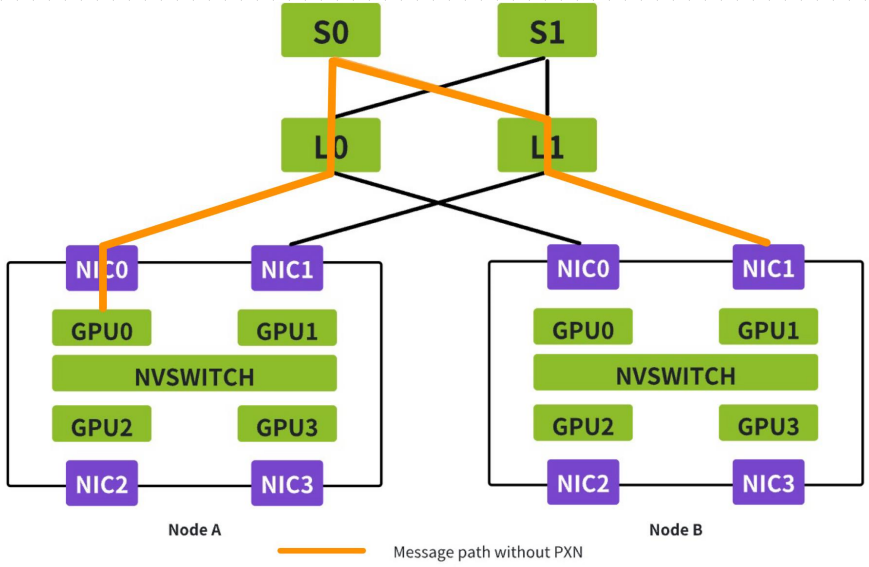
Table 1: Dynamic-traffic ML tasks with communication patterns, ✓(✗) marks satisfied (unsatisfied) patterns

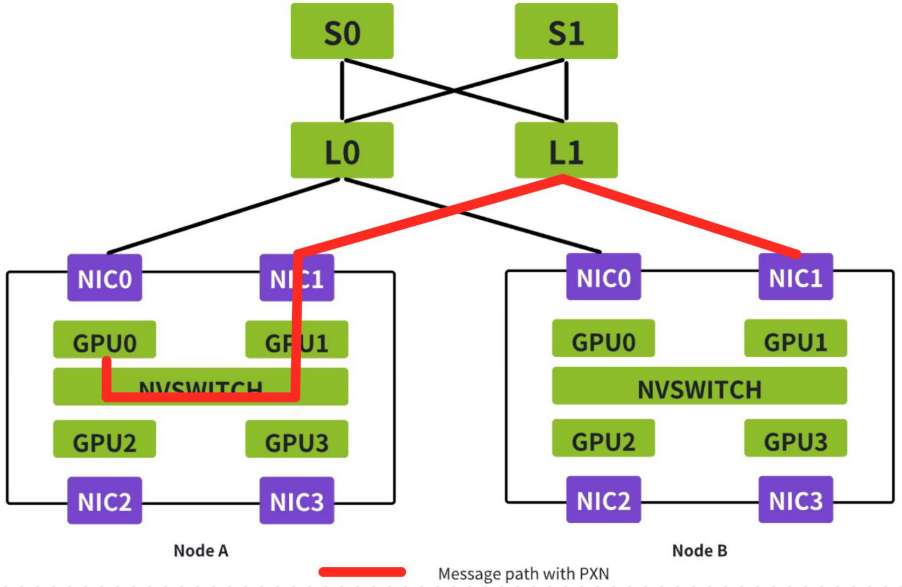
Agenda

- ❑Background
- ❑Motivation
- ❑Opportunities and Challenges
- ❑Design
- ❑Evaluations
- ❑Conclusion

Existing Method: PXN(PCI x Nvlink)

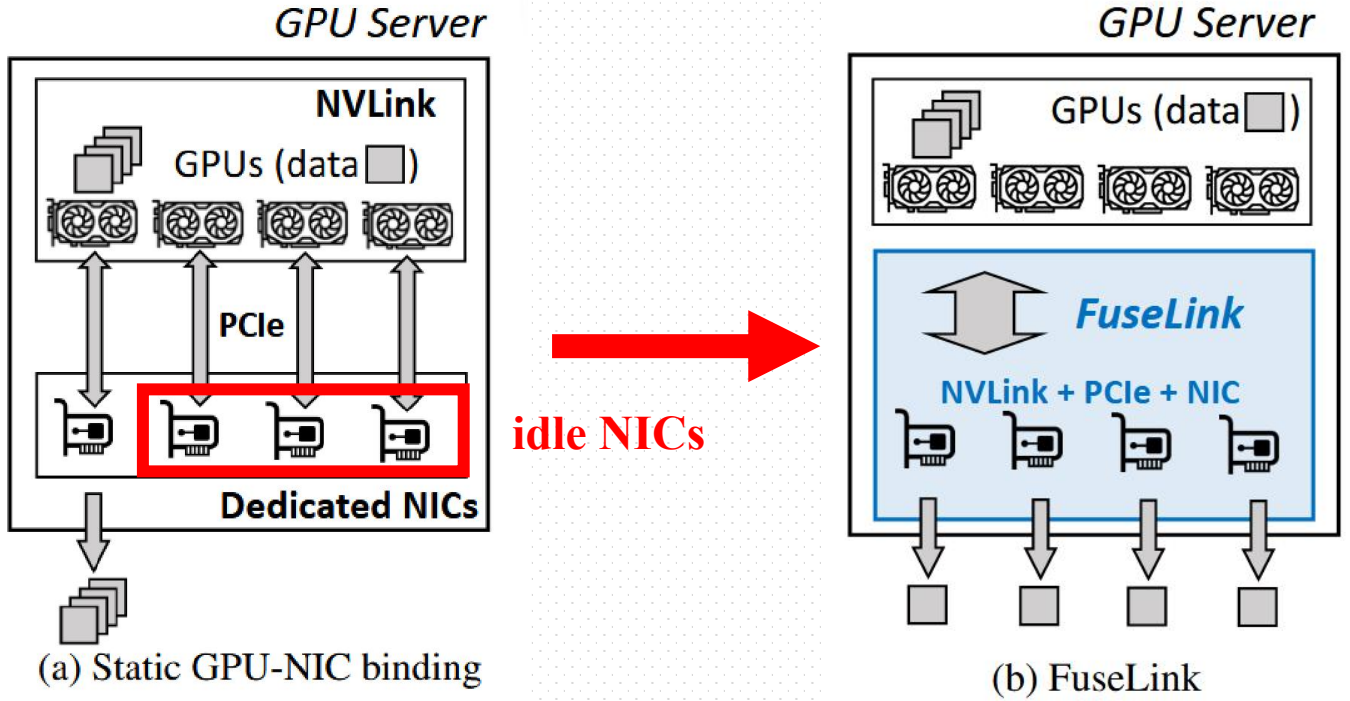
- When the load is unbalanced, PXN still fails to make full use of the bandwidth of each NIC.
  - PXN enables a GPU to communicate with an NIC on the node through NVLink and then PCI.
  - NCCL with PXN is limited by statically bind traffic to paths and cannot support transmitting data through NVLink on receivers.





Opportunities

- Multi-NICs acceleration under imbalanced traffic
  - Leveraging idle NICs
  - NIC load status aware
  - Dynamic NVLink routing to bypass PCIe



(a) Static GPU-NIC binding

(b) FuseLink

Challenges

The incompatibility of intra-server and inter-server networks presents following technical challenges

- **Relaying overhead**
  - **GPU-based relaying incurs significant synchronization overhead，resulting in minimal NIC throughput improvement.**
- **Interruption and contention risks**
  - **Routing traffic to idle NICs risks bandwidth contention and performance disruption when new communications arise from peer workers on relay GPUs.**
- **Scheduling overhead**
  - **Dynamic NIC scheduling incurs high control overhead.**

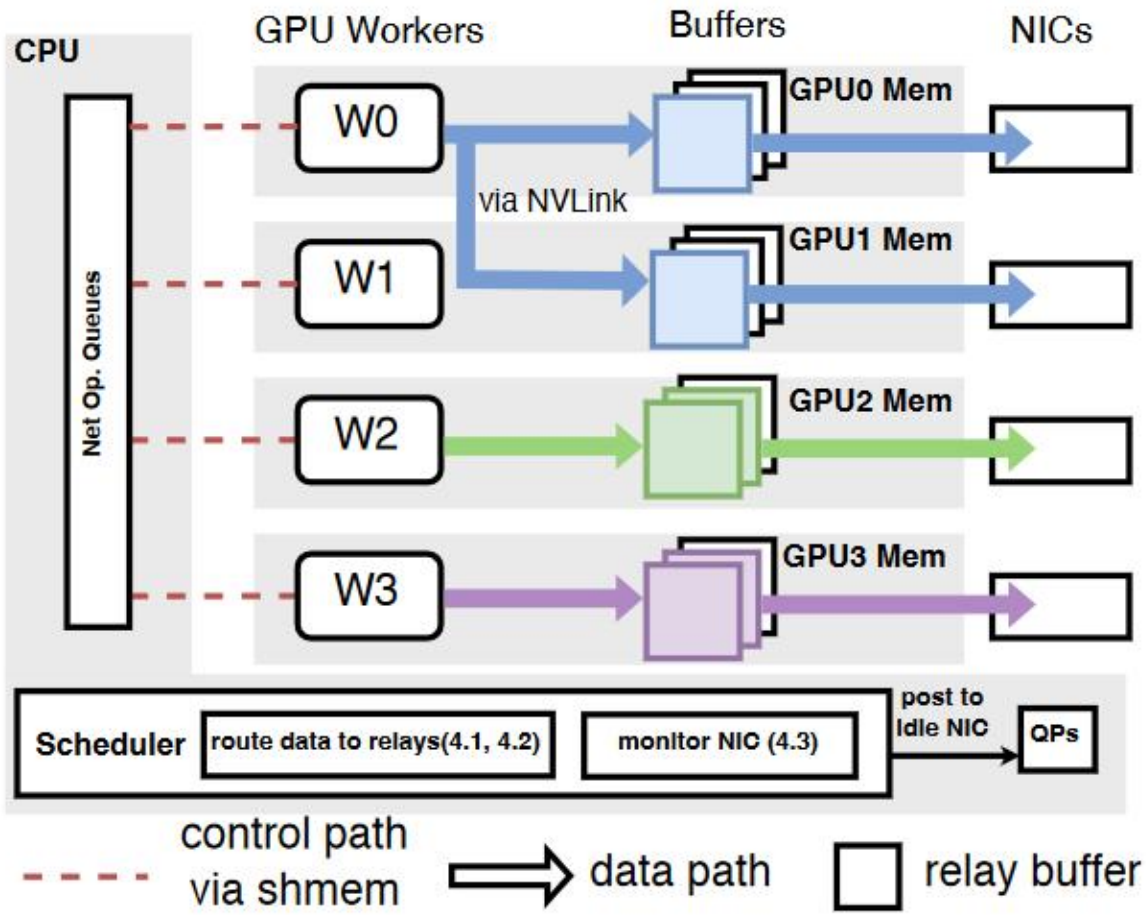
Agenda

- ❑Background
- ❑Motivation
- ❑Opportunities and Challenges
- ❑Design
- ❑Evaluations
- ❑Conclusion

Design - Overview

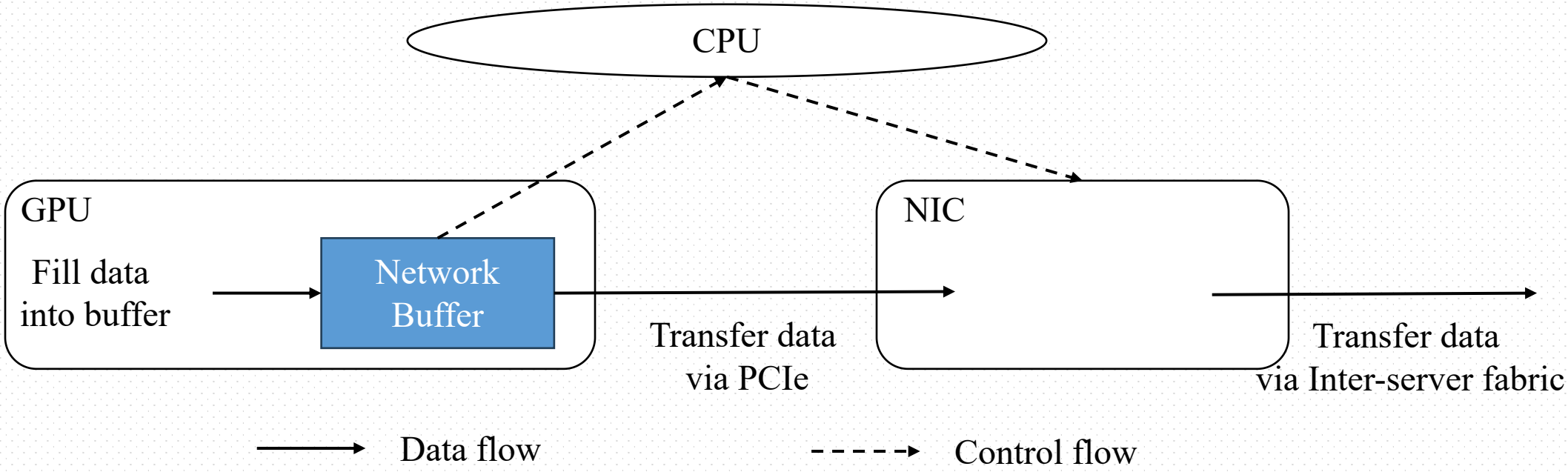
Workflow of FuseLink

- ❑Explores the intra-server topology and establishes RDMA connections
- ❑Identifies available NICs for inter-server communication
- ❑Selects the optimal data path for each NIC and GPU



Design - Relaying

- Inter-server communication (w/o relaying)
  - The GPU fills network buffer (pinned for RDMA communication)
  - The GPU notifies CPU to initiate RDMA transmission

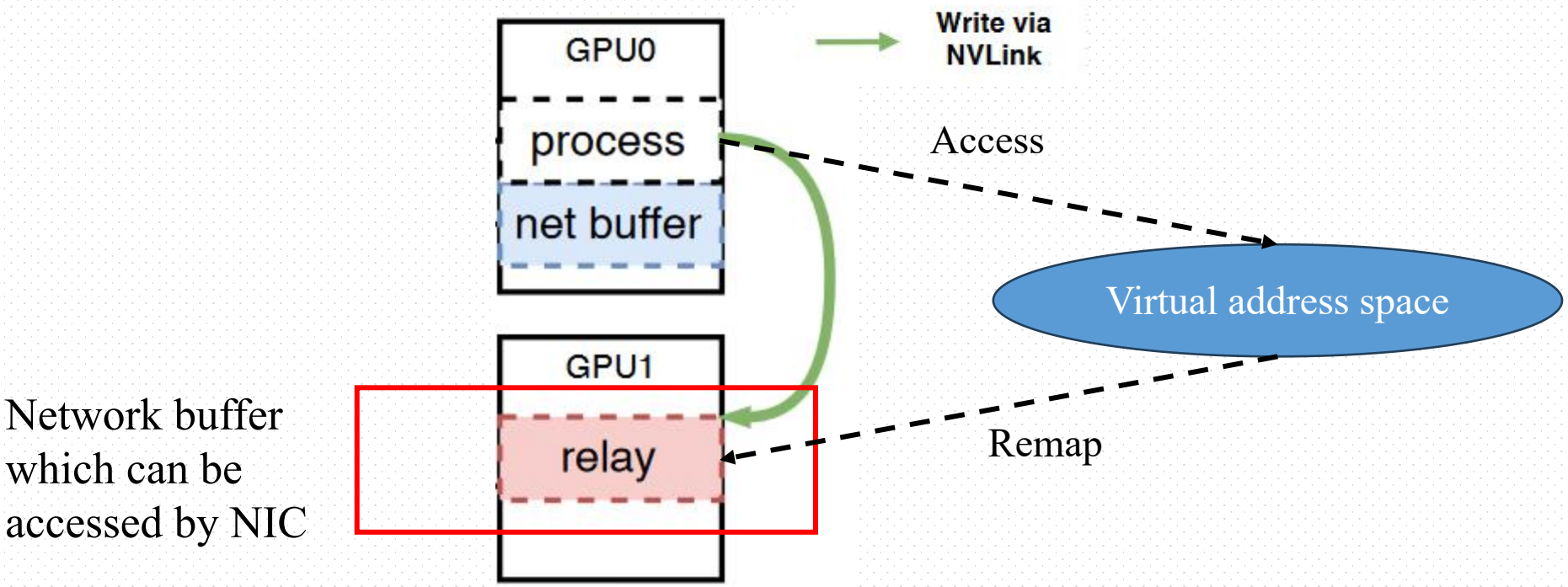


Design - Relaying

- Inter-server communication (w/o relaying)
  - The GPU fills network buffer (pinned for RDMA communication)
  - The GPU notifies CPU to initiate RDMA transmission
- Inter-server communication (w/ relaying)
  - The GPU0 fills network buffer of GPU1
  - The GPU0 notifies CPU to initiate RDMA transmission

Design - Relaying

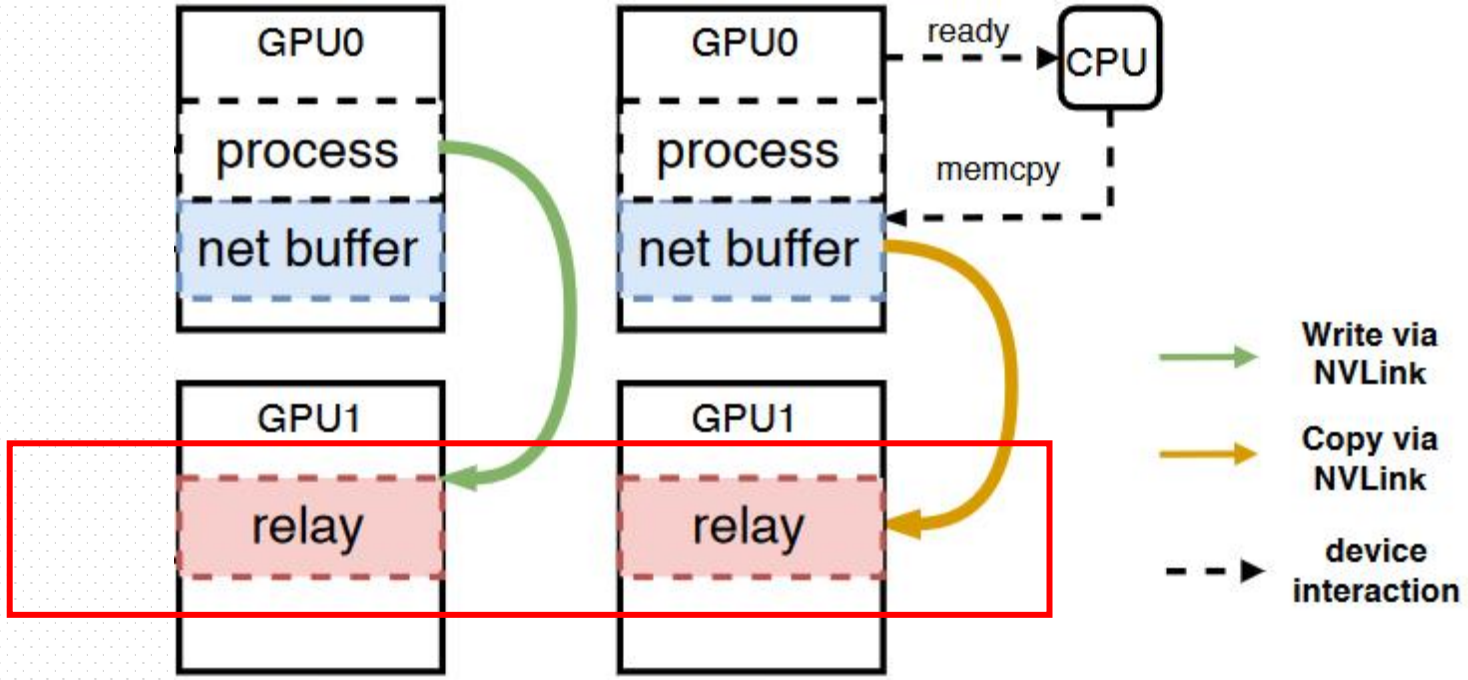
- 4 relaying path as candidates
  - D1: GPU writes data to remapped buffers on relay GPUs
  - D2: CPU copies data to relay GPUs then RDMA



D1

Design - Relaying

- 4 relaying path as candidates
  - D1: GPU writes data to remapped buffers on relay GPUs
  - D2: CPU copies data to relay GPUs then RDMA



Network buffer which can be accessed by NIC

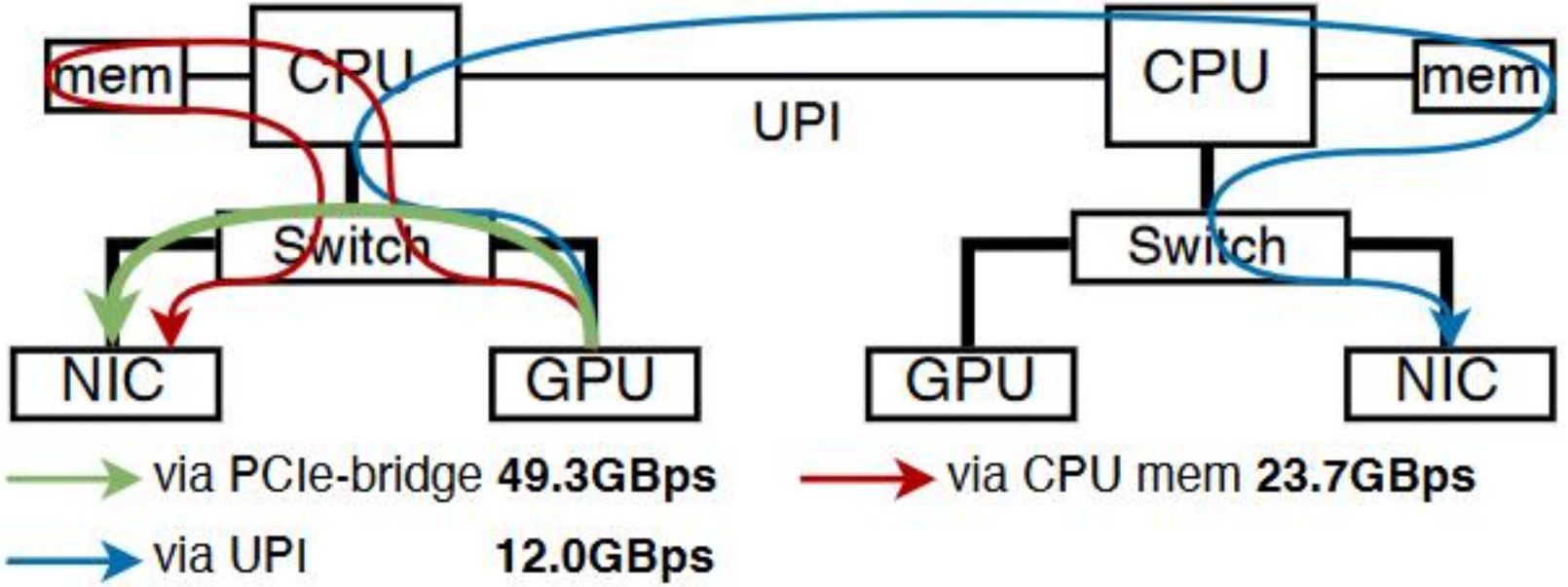
D1 D2

Design - Relaying

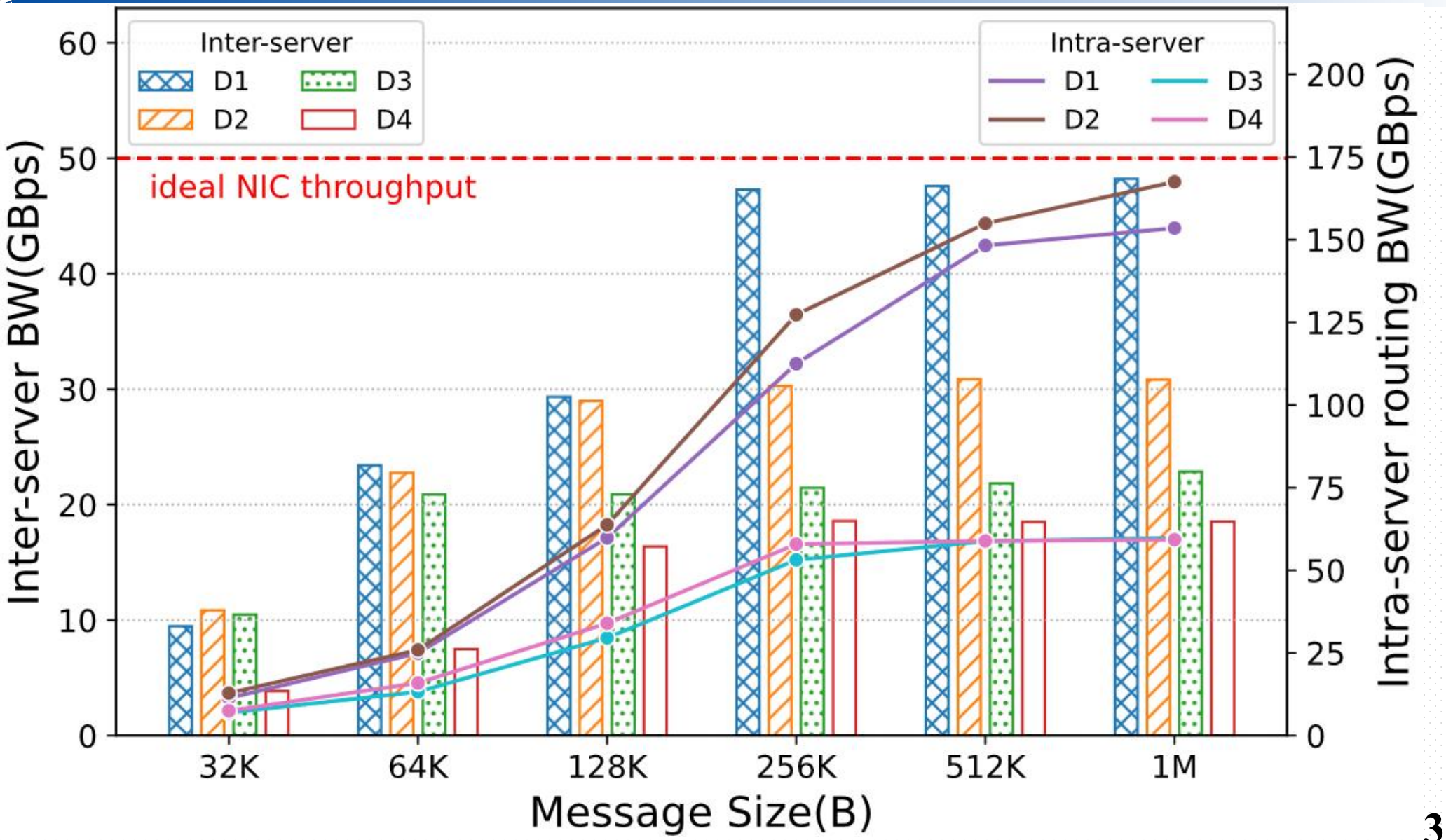
- 4 relaying path as candidates
  - D3: GPU threads write data to buffers mapped to host memory
  - D4: CPU initiates memory copy to host memory

Design - Relaying

- 4 relaying path as candidates
  - D3: GPU threads write data to buffers mapped to host memory
  - D4: CPU initiates memory copy to host memory



Design - Relaying



Design - Relaying

- D3 and D4 achieve the same bandwidth as PCIe, thus limited by PCIe speed and cannot improve relaying efficiency
- D1 achieves higher data path efficiency
  - Only one copy step to relay GPUs, leading to low message latency over indirect NIC
  - Make effect without additional CPU involvement
  - Hide the extra intra-server relaying latency by overlapping it with pipelined intra- and inter-server communication

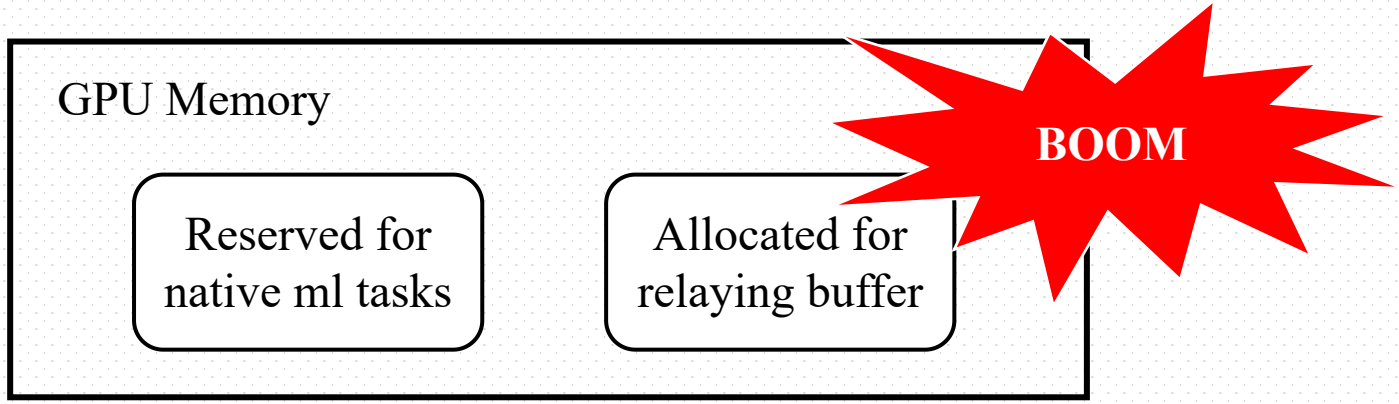
Design - Relaying

- D3 and D4 achieve the same bandwidth as PCIe, thus limited by PCIe speed and cannot improve relaying efficiency
- D1 achieves higher data path efficiency
  - Only one copy step to relay GPUs, leading to low message latency over indirect NIC
  - Hide the extra intra-server relaying latency by overlapping it with pipelined intra- and inter-server communication

Use D1 as relaying traffic

Design - Mitigate Contention

- Data relaying incurs two types of resources contention
  - GPU Memory Contention
    - Allocate extra buffer in relay GPU
    - Exhaust memory of relay GPU causing OOM
    - Cannot have precise estimation of memory usage for dynamic workload

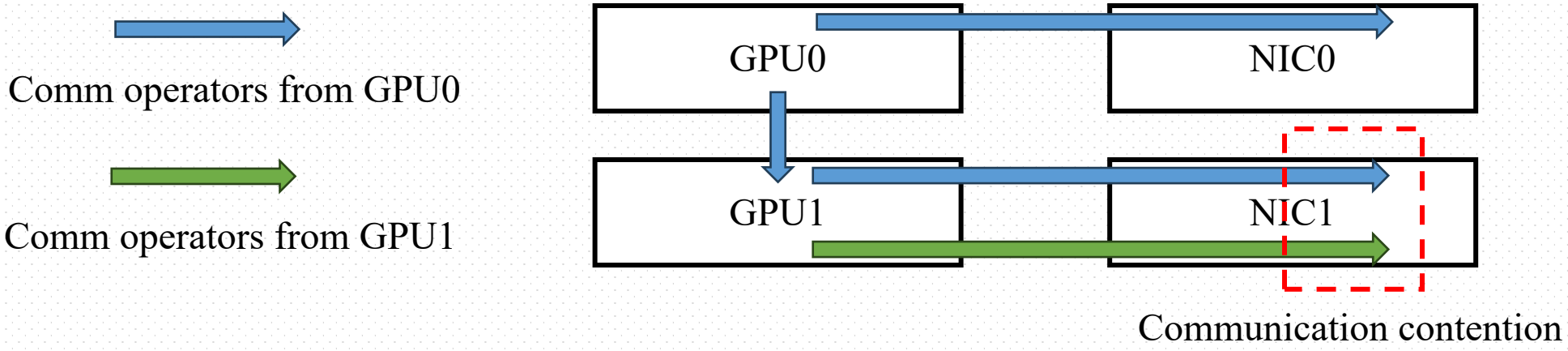


Design - Mitigate Contention

- Relaying without interrupting memory allocation
  - Set a configurable upper limit for relaying memory usage
  - Prioritize memory demands of running tasks on relay GPUs

Design - Mitigate Contention

- Data relaying incurs two types of resources contention
  - Communication Contention
    - Disturb or even interrupt peer GPU communication over direct NICs.



Design - Mitigate Contention

FuseLink provides an interruption-free relaying, including 4 steps

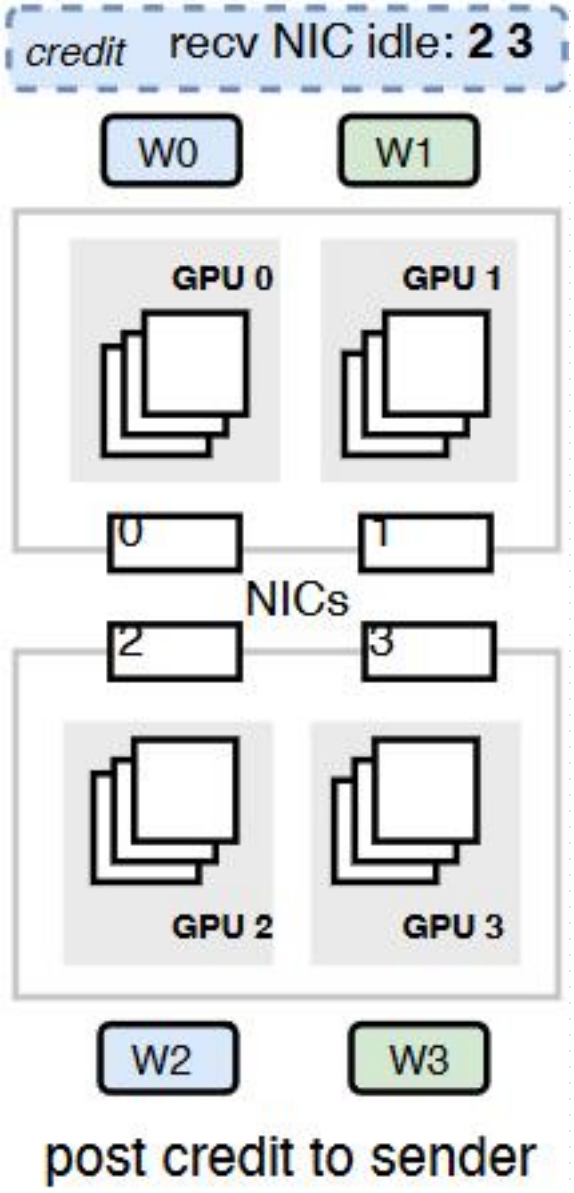
- ❑Monitor NIC status within the server
- ❑Aggregate NIC load status on the sender and receiver side
- ❑Select NICs and initiating data transmission
- ❑Perform intra-server traffic routing inside receivers via memory remapping

Design - Mitigate Contention

- Worker-aware NIC monitoring
  - Mark the status of NICs as idle or busy
    - ML applications typically have large traffic that can fully occupy NICs when they have communication tasks
    - Decided by workers that have the highest priorities on the NIC
  - Monitor NICs workload from the work requests on RDMA connections
    - FuseLink identifies idle NICs by periodically checking new completions of network operations posted by high-priority workers

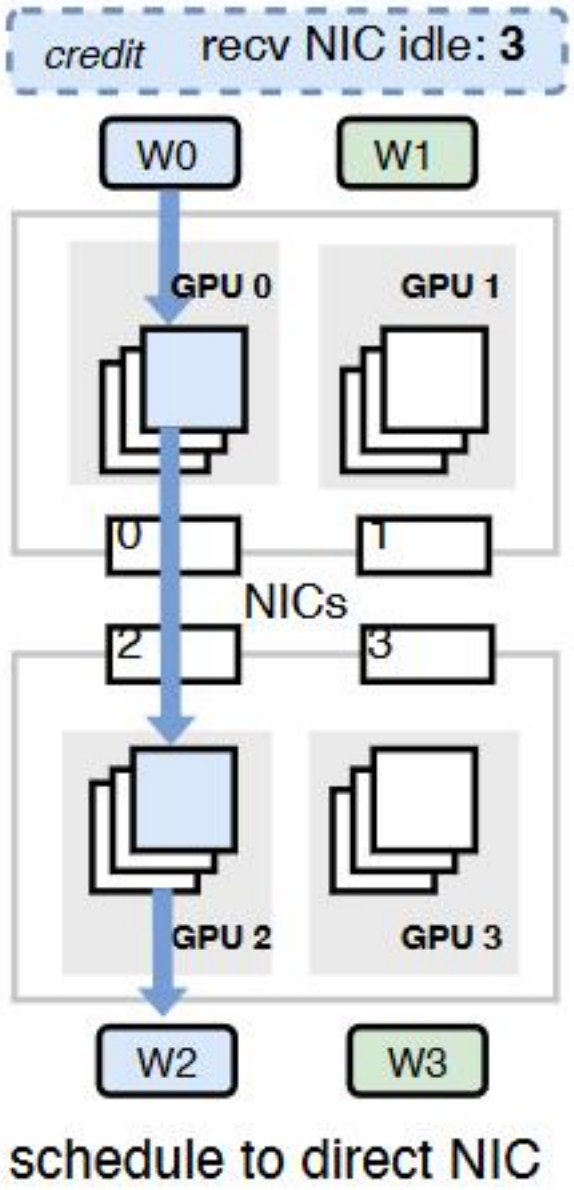
Design - Mitigate Contention

- Load-aware Scheduling
  - Aggregate NIC load status from recievers
    - Write NIC information into credit
    - Collect credits from recievers



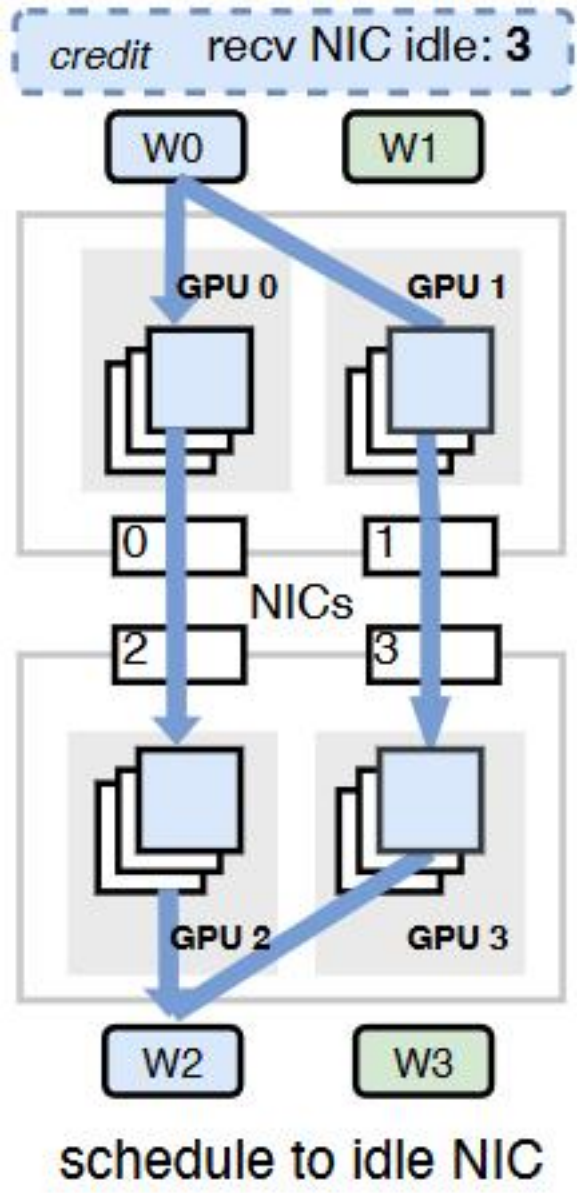
Design - Mitigate Contention

- Load-aware Scheduling
  - Aggregate NIC load status from recievers
    - Write NIC information into credit
    - Collect credits from recievers
  - Select the NIC for sending
    - Selects the direct NIC of worker if it is idle

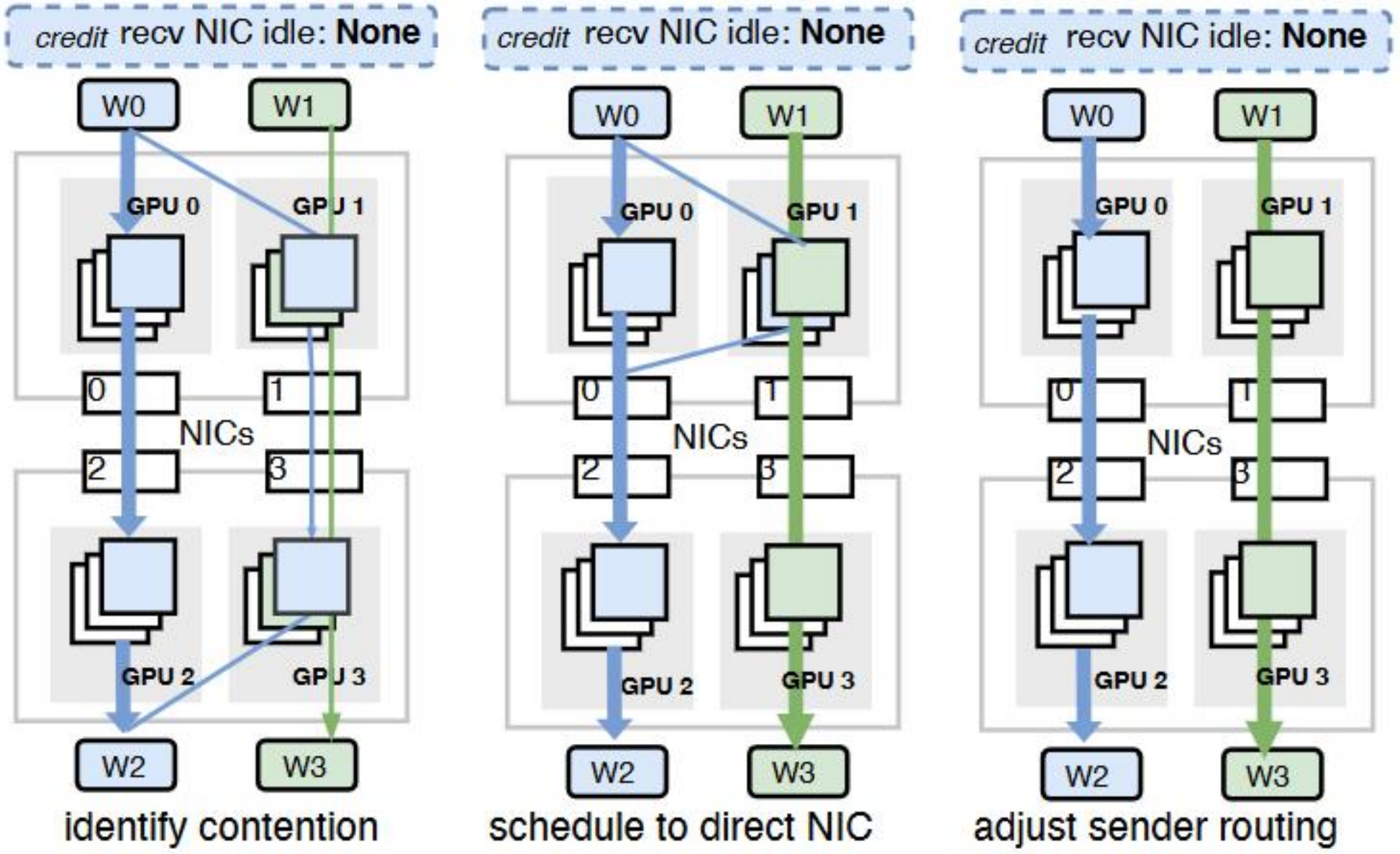


Design - Mitigate Contention

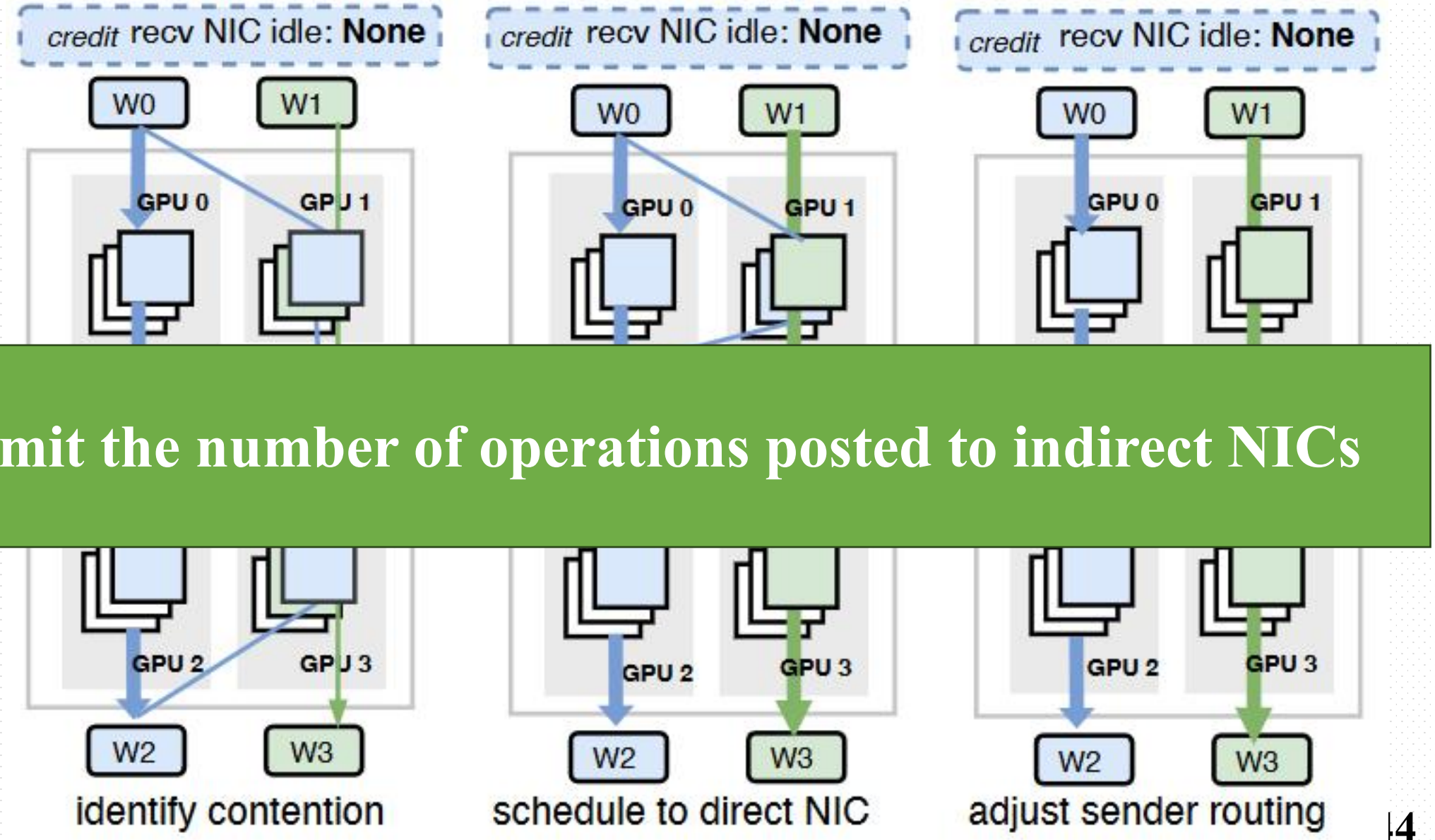
- Load-aware Scheduling
  - Aggregate NIC load status from recievers
    - Write NIC information into credit
    - Collect credits from recievers
  - Select the NIC for sending
    - Selects the direct NIC of worker if it is idle
    - Otherwise, select an idle indirect NIC if exists
    - If no idle NICs found, select the direct NIC



Design - Mitigate Contention



Design - Mitigate Contention



Limit the number of operations posted to indirect NICs

Design - Scheduling with Efficiency

FuseLink has following tradeoffs to ensure scheduling with efficiency:

- Mark NIC load status based on new completions of operations posted by GPU workers, which is later than transmission start time.
- Allows bounded contention over limited number of network operations
- FuseLink allows bounded suboptimal transmission through indirect NICs

Design - Summary

FuseLink can:

- ❑ **Leverage idle NICs through dedicated intra-server fabrics**
- ❑ **Mitigate contention and interruption among GPUs**
- ❑ **Be seamlessly integrated into existing communication library (NCCL, GLOO...)**

Agenda

- ❑Background
- ❑Motivation
- ❑Opportunities and Challenges
- ❑Design
- ❑**Evaluations**
- ❑Conclusion

Evaluation

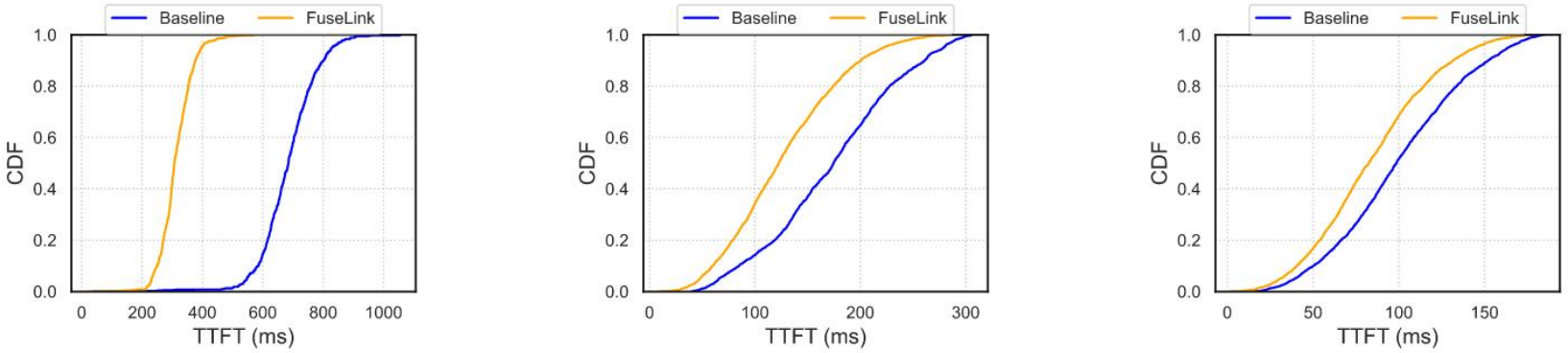
- Setup
  - Intel 8480C CPU, 8*Nvidia Hopper GPUs
  - Inter-server: 8*Connect-X7 50GB/s NICs
  - Intra-server: eight-lane NVLinks & NVSwitches, delivering up to 200GB/s
  - FuseLink scheduler as a daemon process ,FuseLink as a NCCL plugin
- Baseline
  - NCCL with PXN

Evaluation

- In End-to-End Evaluation
  - Q1:What are the performance benefits of FuseLink on machine learning tasks with imbalanced traffic?
- In Microbenchmark
  - Q1:What is the maximum inter-server bandwidth for a single GPU with FuseLink compared to the default?
  - Q2:What is the overhead of FuseLink in traffic scheduling?

Evaluation

- End-to-End performance (Model serving)
- FuseLink shows 1.04-2.73 x→ speedup over the baseline with NIC binding
  - OPT models serving(30B) under 1/2/4 tensor parallel degrees



(a) Eight instances with TP=1 (b) Four instances with TP=2 (c) Two Instances with TP=4

Figure 11: FuseLink model serving performance under different TP degrees and number of serving instances

#Instance	Setting	P50	P99
8	NIC Binding	684.54 ms	903.20 ms
	FuseLink	308.48 ms	464.55 ms
4	NIC Binding	174.46 ms	297.49 ms
	FuseLink	122.61 ms	259.73 ms
2	NIC Binding	98.09 ms	175.36 ms
	FuseLink	81.97 ms	160.36 ms

Table 4: Model serving TTFT comparison under different number of serving instances within a server

Evaluation

❑EP training

- ❑FuseLink improves the training throughput by 1.3×
- ❑Mixtral 8*22B model (TP=4 , EP=8)

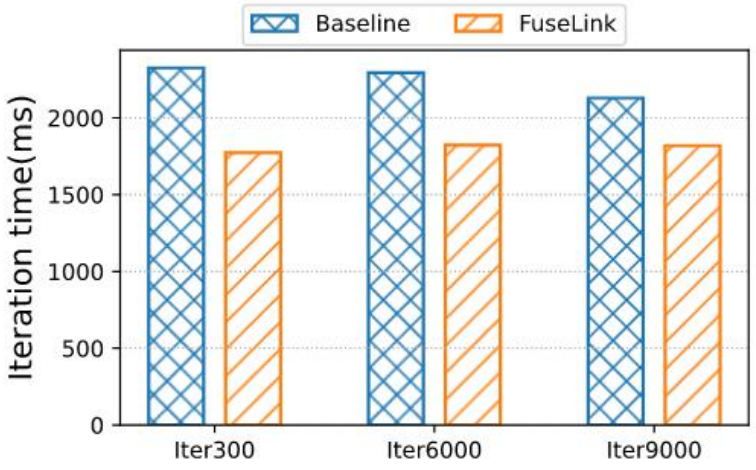


Figure 12: EP training times under NIC binding and FuseLink

❑DLRM training

- ❑FuseLink can effectively reduce the training iteration times
- ❑Criteo advertisement dataset (DP=32， batch size=1024)

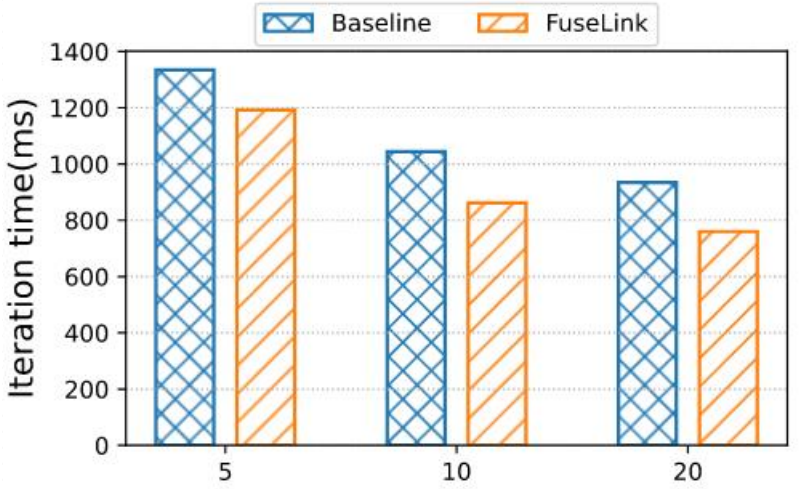
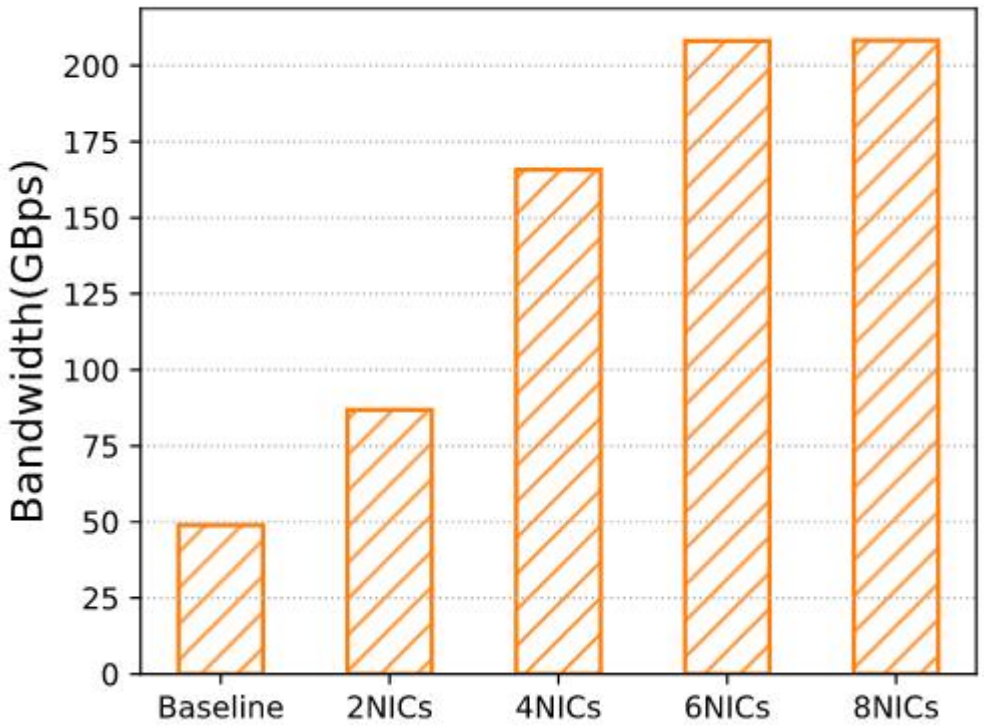


Figure 13: DLRM training iteration times under different cache sizes (GB)

Evaluation

- Microbenchmark-bandwidth improvement
  - FuseLink achieves up to 212GBps bandwidth with six NICs, compared to the default setting with about 50GBps throughput



- NVLink > Indirect NICs bandwidth
  - Maximum bandwidth:all NICs
- NVLink < Indirect NICs bandwidth
  - Maximum bandwidth:

    direct NIC(50GB/s)+Nvilink(200GB/s)

Evaluation

- Microbenchmark-scheduling overhead
  - The scheduling overhead is acceptable and the tradeoff will not harm the overall performance.
  - Relay remapping is called when optimizing the data paths between NICs and GPUs, which brings around 95-193us latency in each remapping.

Controls	Avg. cost(us)
post & pull ops	0.8-1.4
query NIC load	0.9-1.6
process send	2.8-3.5
process recv	4.9-5.6

Table 3: Network operation processing overheads in FuseLink

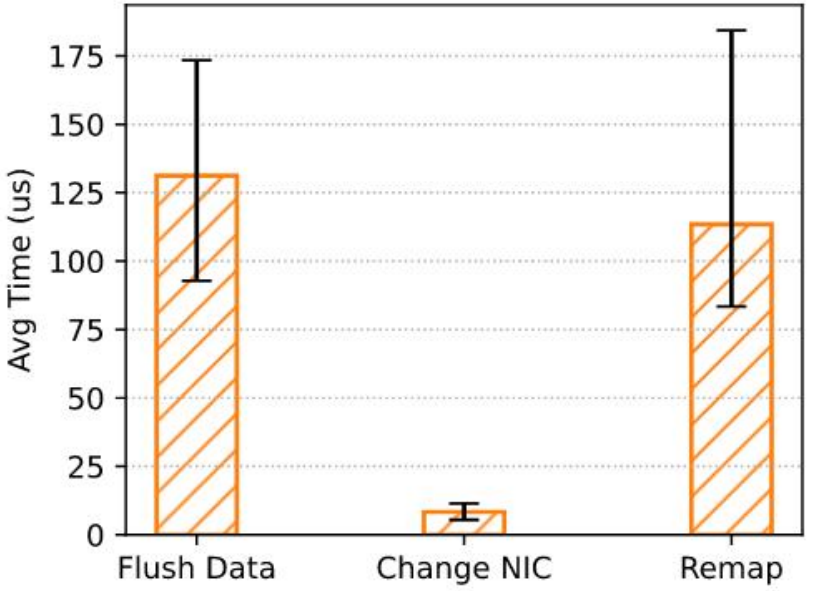


Figure 10: Average scheduling overhead decomposed into flushing data, changing NIC, and remapping

Agenda

- ❑Background
- ❑Motivation
- ❑Opportunities and Challenges
- ❑Design
- ❑Evaluations
- ❑Conclusion

Conclusion

- Why FuseLink?
  - FuseLink enables inter-server GPUs to efficiently communicate over multiple NICs.
  - FuseLink exploits dedicated intra-server network for intra-server traffic routing to fully utilize GPU interconnects.
  - FuseLink is integrated into NCCL, so that ML applications can benefit from FuseLink seamlessly without code modifications.
  - Fuselink implementation has about 3000 lines of code in C++.