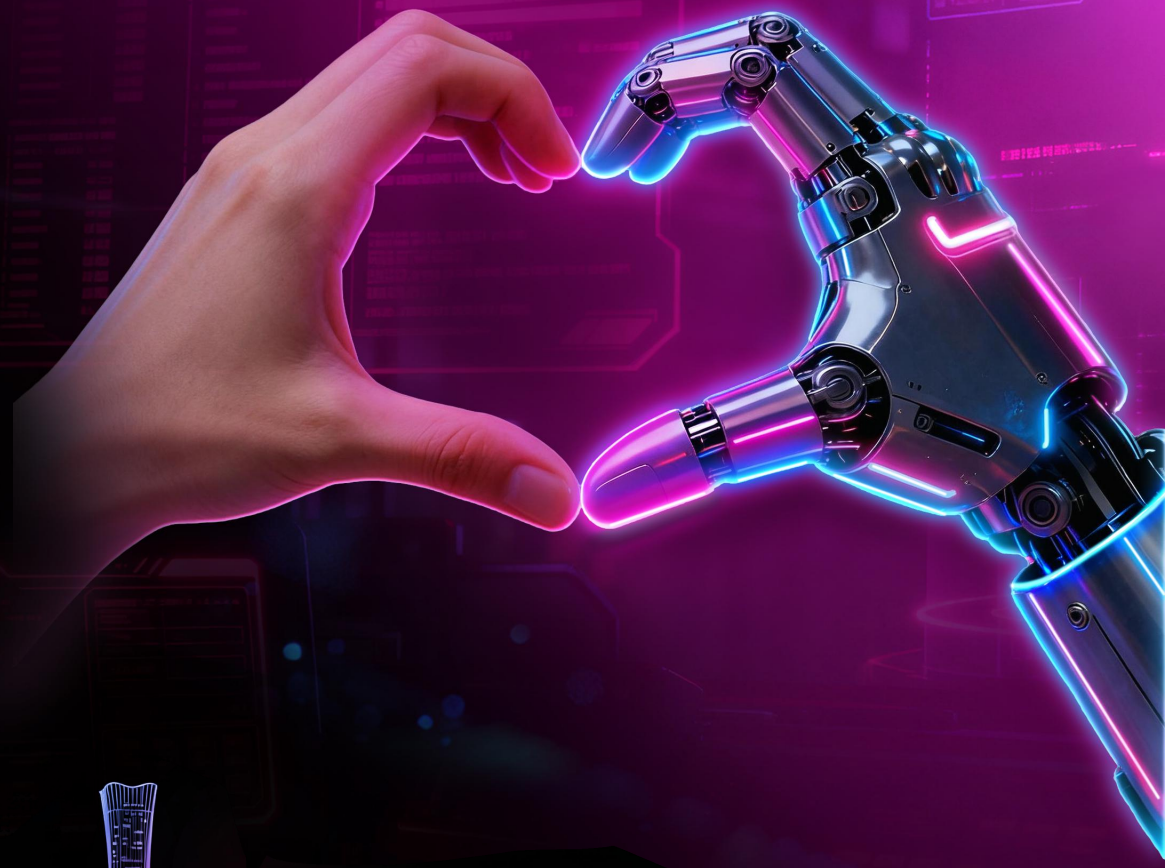
开源社 kaiyuanshe

云原生社区 Cloud Native Community

COSCon'25
第十届中国开源年会

众智开源 | Open Source, Open Intelligence



RDMA赋能云上高性能Serverless LLM弹性推理

刘明轩 西北工业大学计算机学院 应届博士生

2025年12月7日 中国，北京

CONTENTS

目录

PART 01 背景

1.1 为什么需要Serverless LLM弹性推理

Booming demand for serving custom LLMs

- Open-source models ↑
- Fine-tuned models ↑
- Custom LLM services ↑

Llama-Family MISTRAL AI_ Qwen DeepSeek Microsoft Phi-3 snowflake Yi

Serverless as a cost-effective solution

Traditional Choices for Model Serving	
Buy a GPU server	Too expensive
Rent a GPU server	Underutilized
Use LLM-Service API	Usage limit & Cannot custom

Pay-as-you-go Model Serving Platform.

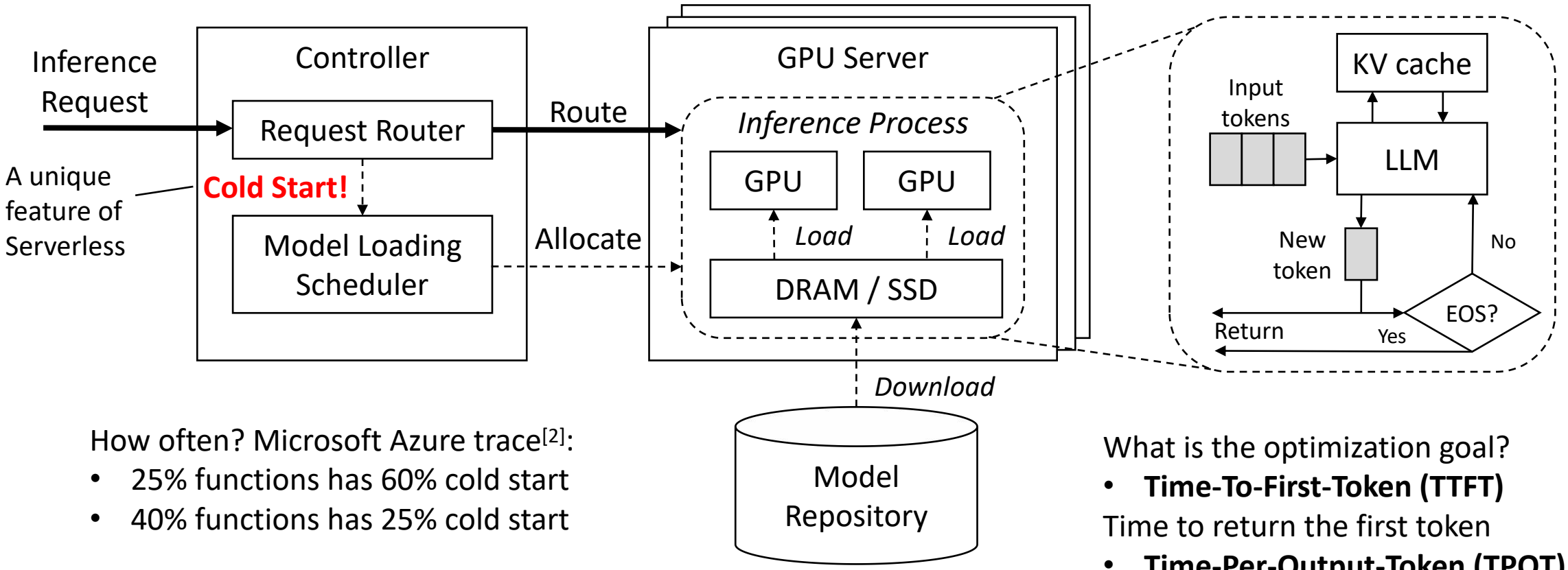
Huge interests from industry and academia

Hundreds competing to develop next-gen AI Serving Platform

Amazon Bedrock · Azure Machine Learning Studio · KServe · Ray Serve

1. 参考https://www.usenix.org/conference/osdi24/presentation/fu.

1.2 逻辑架构图：ServerlessLLM@OSDI24[1]为例



How often? Microsoft Azure trace[2]:

- 25% functions has 60% cold start
- 40% functions has 25% cold start

What is the optimization goal?

- **Time-To-First-Token (TTFT)**

Time to return the first token

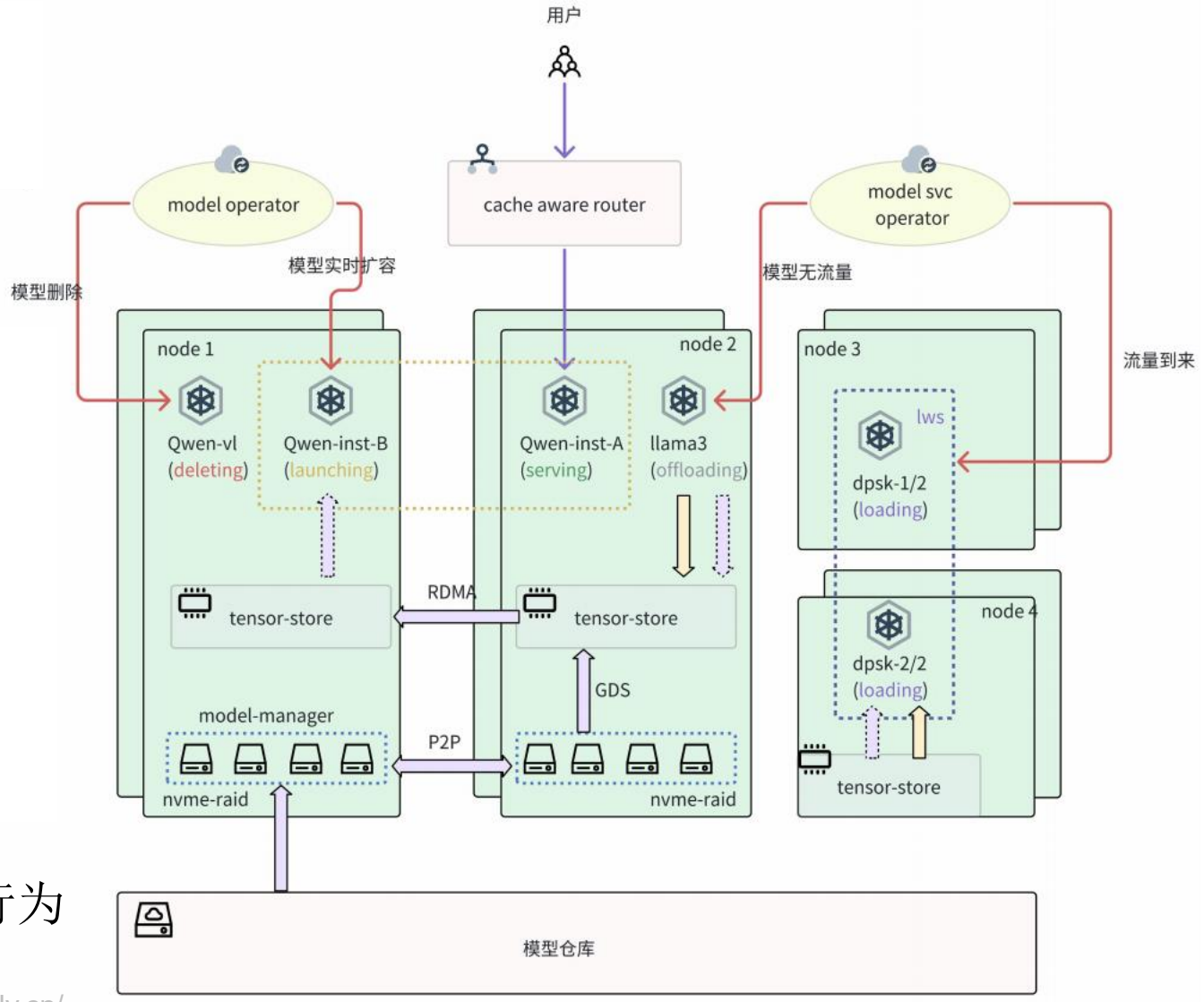
- **Time-Per-Output-Token (TPOT)**

Time between each token response

1. 图参考Fu, Yao, et al. "{ServerlessLLM}:{Low-Latency} serverless inference for large language models." 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). 2024.

1.2 生产环境架构图：无限光年InfSys平台[1]为例

- **Serverless inference system定义[2]**
  - 极致的冷启动速度
  - 系统以单个GPU作为最小调度单位
  - 空闲时释放GPU，流量来时快速抢占GPU
  - 推理实例在设备间/节点间自由调度
- **核心组件**
  - 推理实例（vLLM Pod/LWS）
  - Tensor-store（daemonset）
  - Model-manager（daemonset）
- **三层调度**
  - Router：用户请求流量调度
  - Model operator：推理实例生命周期
  - Model svc operator：推理实例应对流量的行为



1. INF无限光年：打造新一代可信大模型技术及其原生应用. https://www.infly.cn/
2. 观点与图来源：王超. vLLM 推理优化 Meetup 上海站. https://mp.weixin.qq.com/s/8NC5j2q2IS2U-FpoyYCkqA.

1.3 Serverless GPU池化 -> Serverless LLM弹性推理

- Serverless GPU：面向多租户的GPU池化[1]
  - 方法1：**GPU复用（Multiplexing）**，一张GPU同时服务于多个租户
    - 工业界：NVIDIA Time-slicing/Streams/MPS/MIG/GRID vGPU、Microsoft Singularity
    - 学术界：INFless@ASPLOS22、REEF@OSDI22、TGS@NSDI23、LithOS@SOSP25
  - 方法2：**弹性调度（Auto-scaling）**，随时调整模型布局克服GPU内存限制
    - 工业界：开源社区KServe、阿里云Torpor@ATC25、华为云Flex:ai
    - 学术界：Shepherd@NSDI23、Tetris@ATC22、INFaaS@ATC21、Gillis@ICDCS21

v.s. 传统LLM推理平台[3, 4]

- Serverless GPU池化 -> **Serverless LLM弹性推理**的**典型特征（诟病）**：
  - LLM模型权重传输开销大，加重**冷启动延时**
  - LLM推理过程中的PD分离充分利用GPU资源，却引入**模型参数/KVCache传输**
  - GPU未必是Batching使用，大多是多租户Streaming使用[4]，亟需**资源弹性**

1. GPU池化的分类方法参考Aegaeon@SOSP25
2. Mooncake. Moonshot. https://github.com/kvcache-ai/Mooncake
3. Dynamo. NVIDIA. https://github.com/ai-dynamo
4. Azure LLM/LMM Inference Traces：AzureLLMInferenceDataset2023/24/25. https://github.com/Azure/AzurePublicDataset

1.4 克服诟病：主流Serverless LLM架构

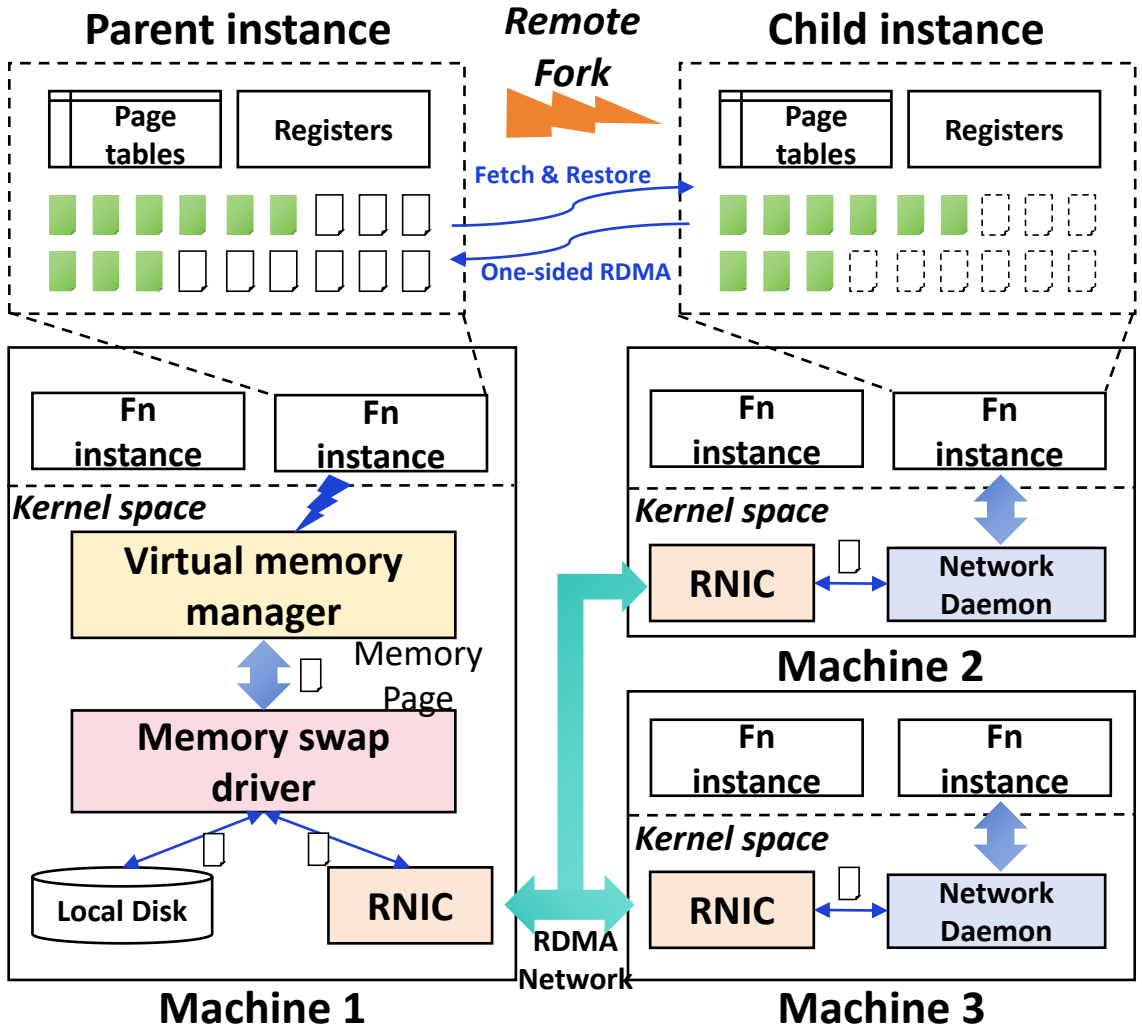
- 工业界：
  - KubeRay[1] + KServe w/ llm-d + vLLM/SGLang
  - NVIDIA Dynamo + Grove
  - 阿里云HydraServe@NSDI26/Aegaeon@SOSP25：优化冷启动/Token弹性调度
  - 华为云DeepServe@ATC25：动态预暖容器 + 分层存储
- 学术界：
  - ServerlessLLM@OSDI24：优化模型权重快速加载
  - Medusa@ASPLOS25：优化冷启动
  - GRouter@EuroSys26：加速GPU-GPU和GPU-Host通信
  - BlitzScale@OSDI25/λ-Scale@arXiv25：多播加速模型分发

- 发现优化空间：**RDMA网卡内资源利用率低，存在潜在价值**

1. 不加 KubeRay 是更常见的选择，因为 KServe 和 llm-d 都使用 LWS 进行多节点部署，无需 KubeRay；加 KubeRay 适合需要与 Ray 生态集成的场景

PART 02 RDMA如何赋能

2.1 RDMA赋能推理实例Remote Fork快速启动



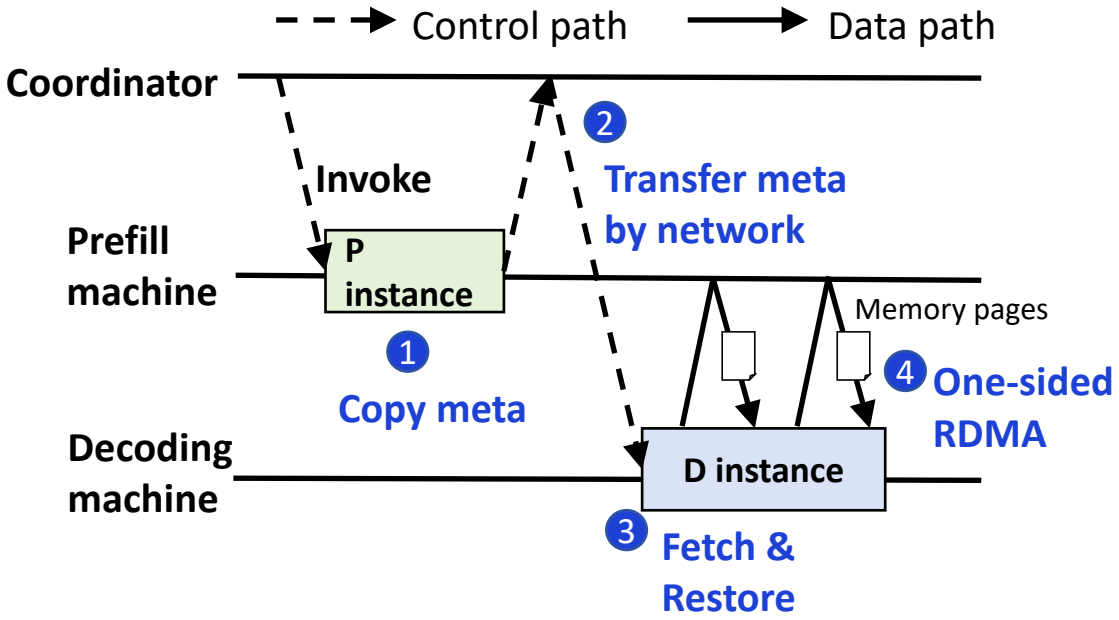
- **基于Fork的函数实例快速启动**
  - 机内：类似Linux Fork，Fork是最受欢迎的Serverless实例快速启动方式之一[1]
  - 跨机：Linux Fork -> **Remote Fork[2]**
- **传统Remote Fork方法[2]：**
  - ckpt + NFS传输 + restore：延时大
- **高效Remote Fork依赖RDMA：**
  - 父实例生成describer交给子实例
  - 子实例Page Fault (PF) 触发RDMA Read
  - Remote Fork for **Serverless LLM**
    - e.g. Cross-machine fork clones prefill instances→decode instances

1. Chai, Xiaohu, et al. "Fork in the Road: Reflections and Optimizations for Cold Start Latency in Production Serverless Systems." 19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25). 2025.
2. Wei, Xingda, et al. "No provisioned concurrency: Fast {RDMA-codesigned} remote fork for serverless computing." 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). 2023.

Remote Fork for Serverless LLM

```
1  def runTransformer(...):
               ...
2  runTransformer(...)
3  p_describer = fork_prepare()      1
4  coordinator.send(p_describer)     2
5  runTransformer(...)
```

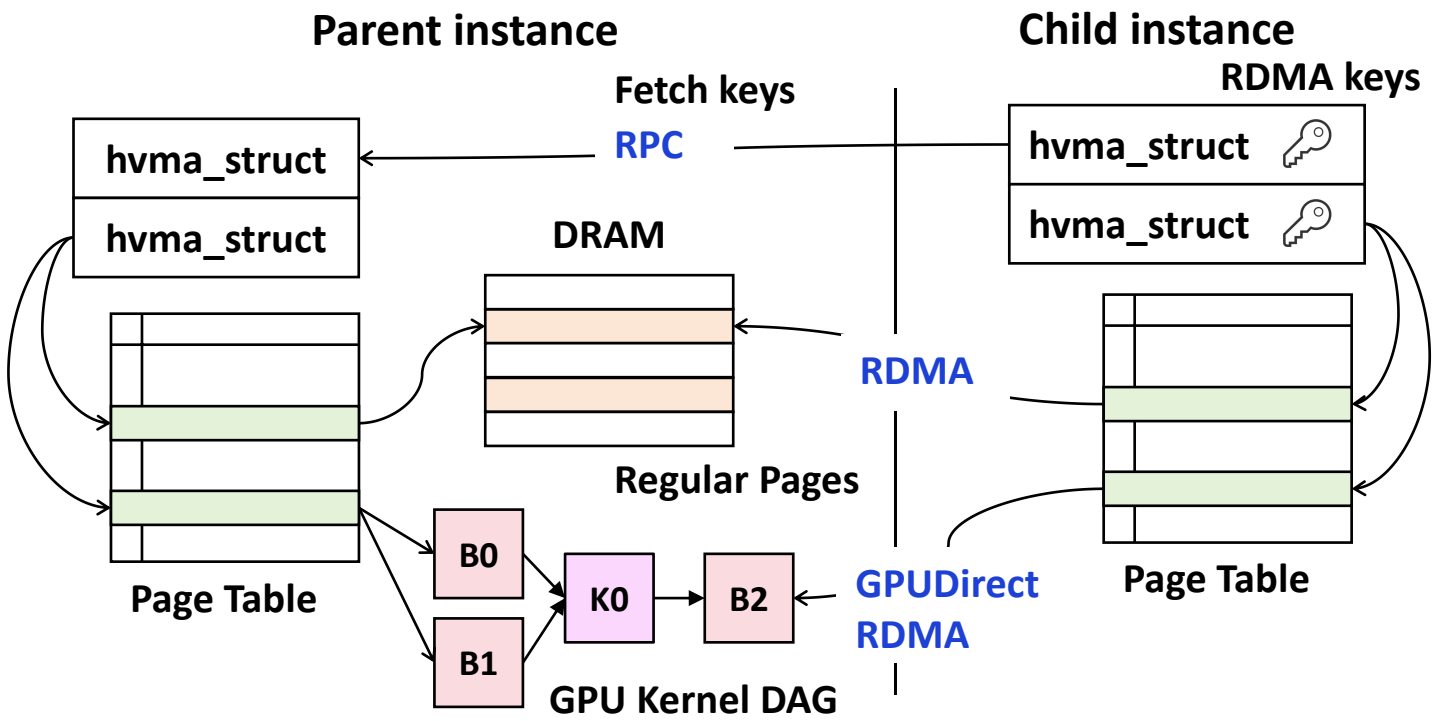
```
1  p_describer = coordinator.recv()      3
2  fork_resume(p_describer) #D starts    4
   ( continus runTransformer(...) ... )
```



- **物理内存跨机映射：**父实例的物理地址映射到子实例的虚拟地址空间
- **按需远程换页：**子实例执行期间，Page Fault (PF) 会触发 RDMA READ获取Page
- **消除序列化/反序列化开销：**父子实例间直接以Page为粒度通信

1. 图来源Liu, Mingxuan, Jianhua Gu, and Tianhai Zhao. "ServerlessPD: Fast RDMA-Codesigned Disaggregated Prefill-Decoding for Serverless Inference of Large Language Models." 2025 IEEE International Conference on Web Services (ICWS). IEEE, 2025.

2.2 GPUDirect RDMA赋能推理实例快速Remote Fork

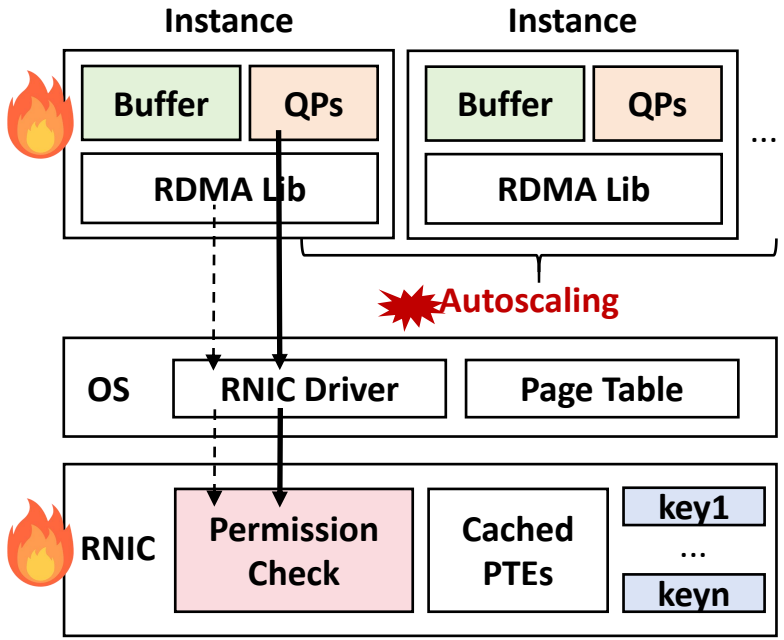


- 统一异构CPU/GPU虚拟内存地址空间（hvma）
- 正在运行的子推理实例：
  - CPU部分遇到PF，根据describer直接**RDMA READ**
  - GPU decoding时遇到由于KV Cache等引发的PF，根据describer直接**GDR READ**

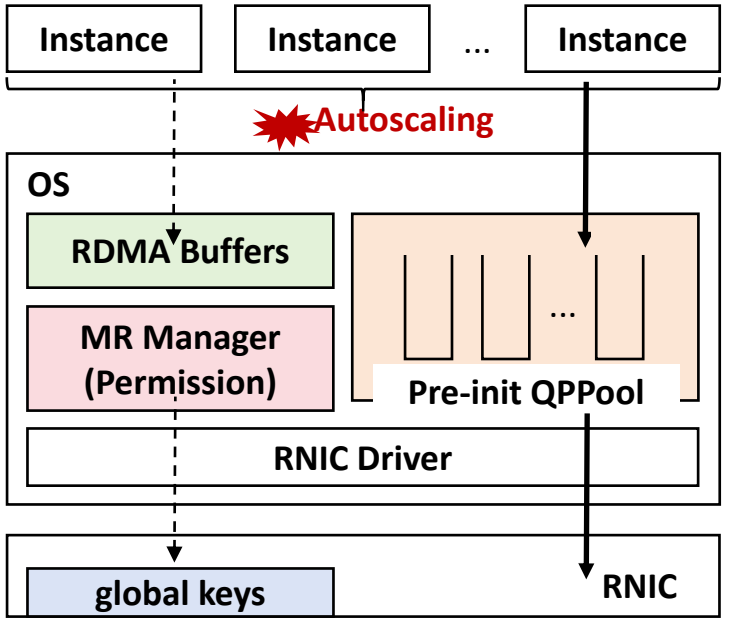
1. 图来源Liu, Mingxuan, Jianhua Gu, and Tianhai Zhao. "ServerlessPD: Fast RDMA-Codesigned Disaggregated Prefill-Decoding for Serverless Inference of Large Language Models." 2025 IEEE International Conference on Web Services (ICWS). IEEE, 2025.

2.3 内核空间RDMA构建Remote Fork基座[1, 2]

Traditional RDMA (e.g. k8s-rdma-shared-device-plugin[3])



Kernel-space RDMA (our design)



- **预分配RDMA池化资源（QP Pool）：** 消除冷启动的RDMA控制面初始化开销
- **内核空间共享RDMA：** RDMA用户库剥离容器，通过轻量系统调用操作RDMA
- **硬件功能软件化：** 远程内存访问权限控制从RNIC移至内核（CPU），缓解RNIC的Scalability瓶颈

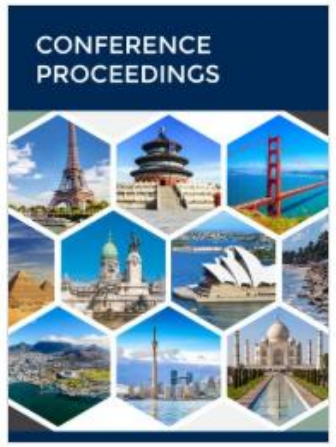
1. Tsai, Shin-Yeh, and Yiying Zhang. "LITE kernel RDMA support for datacenter applications." Proceedings of the 26th Symposium on Operating Systems Principles. 2017.
2. Wei, Xingda, et al. "{KRCORE}: A microsecond-scale {RDMA} control plane for elastic computing." 2022 USENIX Annual Technical Conference (USENIX ATC 22). 2022.
3. k8s-rdma-shared-device-plugin: https://github.com/Mellanox/k8s-rdma-shared-dev-plugin

PART 03 我们的工作：ServerlessPD

2025 IEEE International Conference on Web Services (ICWS)

ServerlessPD: Fast RDMA-Codesigned Disaggregated Prefill-Decoding for Serverless Inference of Large Language Models

Year: 2025, Pages: 305-315
DOI Bookmark: 10.1109/ICWS67624.2025.00045



3.1 LLM推理： Prefill vs Decoding

Prefill: generate KV cache

- Generate KV Cache
- **Compute-bound**

Decode: generate next token

- Fetch KV Cache from HBM
- **Memory-bound**

	TTFT	TPOT
Chat	Low (< 1s)	Match read speed (~ 100ms)
Search	Very Low (~ 200ms)	Match read speed (~ 100ms)
Program	Very Low (~ 200ms)	Very Low (~ 50ms)

- **PD分离的四个层次[1]**
  - L-1： 一张GPU卡上融合PD
  - L0： 一张GPU卡上分别调度PD
  - L1： 在一个节点内不同GPU间调度PD
  - L2： 在高速网络互连的同构节点间调度PD
  - **L3： 在高速网络互连的异构节点间调度PD**
    - **如H800节点运行P实例+H20节点运行D实例**

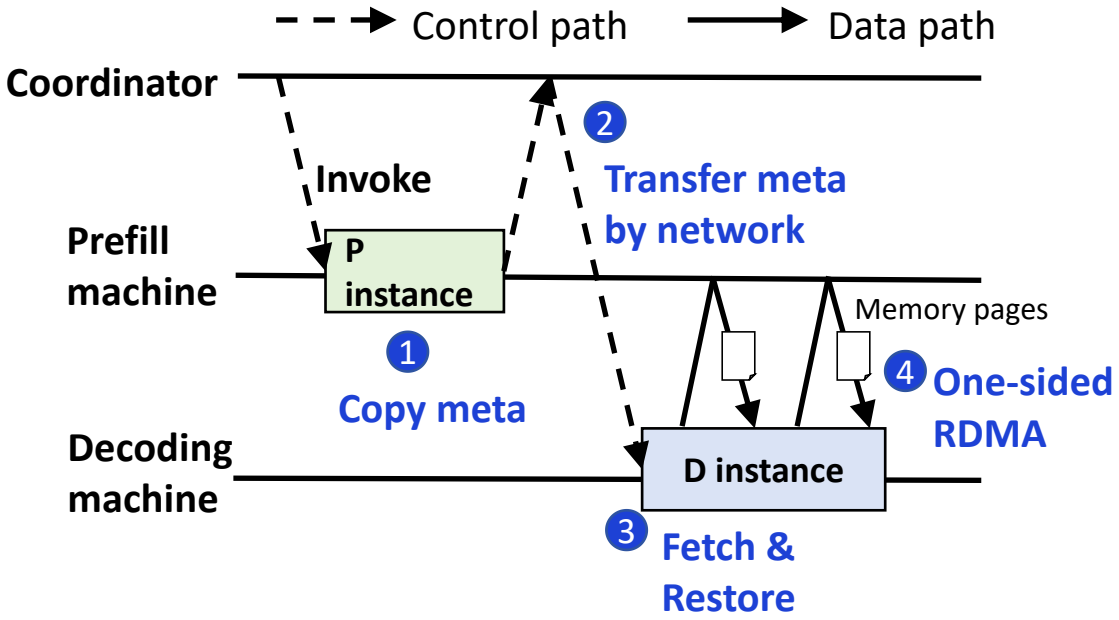
H800	H20
80GB VRAM， 3.3 TBps	96GB VRAM， 4 TBps
~ 1 PFLOPS	~ 200 TFLOPS

1. PD分离的五个层次来源：https://zhuanlan.zhihu.com/p/8056351077
2. 图参考Zhong, Yinmin, et al. "{DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving." 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). 2024.

3.2 思路： Remote Fork for Serverless LLM

```
1  def runTransformer(...):
           ...
2  runTransformer(...)
3  p_describer = fork_prepare()      1
4  coordinator.send(p_describer)     2
5  runTransformer(...)
```

```
1  p_describer = coordinator.recv()      3
2  fork_resume(p_describer) #D starts    4
   ( continus runTransformer(...) ... )
```

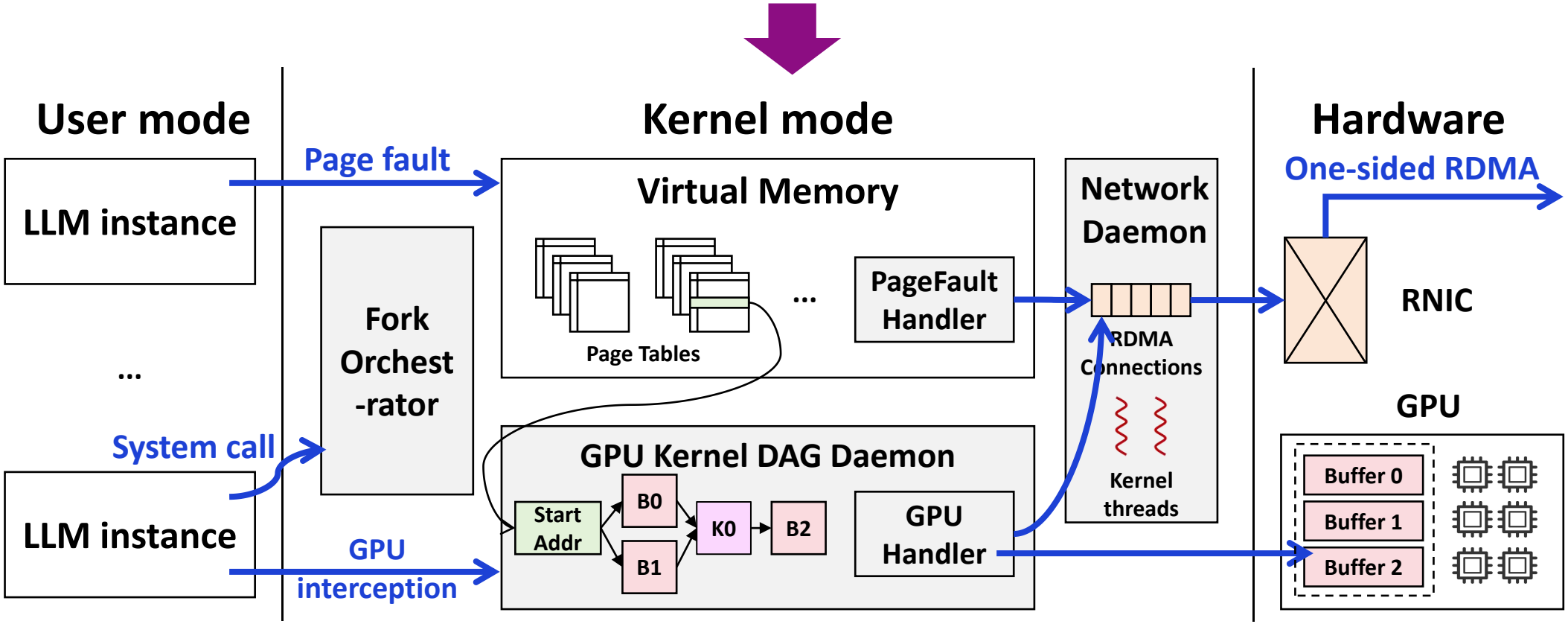


- **物理内存跨机映射：**父实例的物理地址映射到子实例的虚拟地址空间
- **按需远程换页：**子实例执行期间，Page Fault (PF) 会触发 RDMA READ获取Page
- **消除序列化/反序列化开销：**父子实例间直接以Page为粒度通信

1. 图来源Liu, Mingxuan, Jianhua Gu, and Tianhai Zhao. "ServerlessPD: Fast RDMA-Codesigned Disaggregated Prefill-Decoding for Serverless Inference of Large Language Models." 2025 IEEE International Conference on Web Services (ICWS). IEEE, 2025.

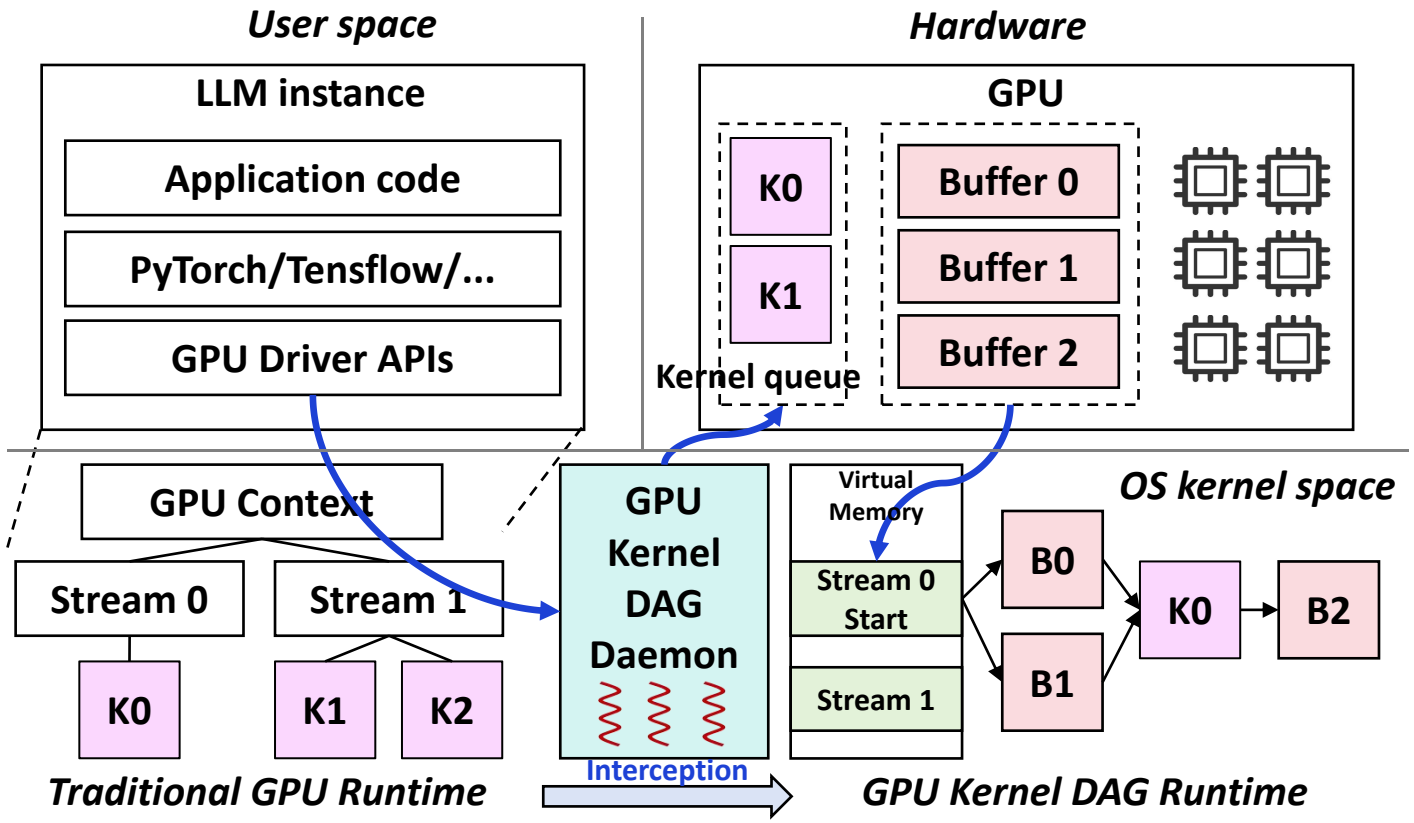
3.3 Challenges & ServerlessPD Architecture

- **C#1：** How to achieve efficient GPU context integration for remote fork?
- **C#2：** How to enable high-speed KV cache transfer for elastic scaling?
- **C#3：** How to determine the optimal launch point for decoding instances?



3.3.1 GPU-Aware Remote Fork

Challenge 1: How to achieve efficient GPU context integration for remote fork?

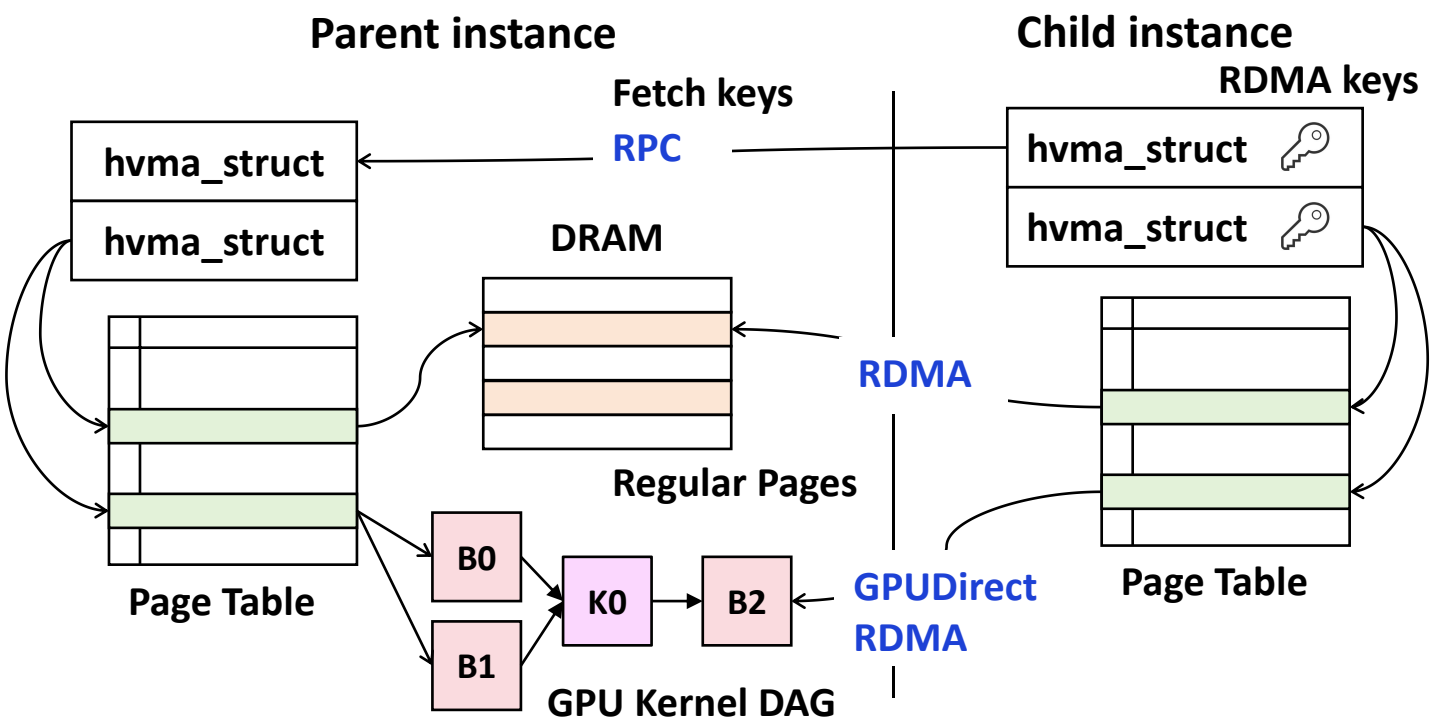


- **OS-Kernel GPU Kernel DAG[1]**
  - GPU Remoting[2]
- **API Interception & DAG Maintenance**
  - Non-Blocking GPU Checkpointing
  - COW Restoration

1. Wei, Xingda, et al. "PhoenixOS: Concurrent OS-level GPU Checkpoint and Restore with Validated Speculation." Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles. 2025.
2. Eiling, Niklas, et al. "GPU acceleration in unikernels using cricket GPU virtualization." Proceedings of the SC'23 Workshops of the International Conference for High Performance Computing, Network, Storage, and Analysis. 2023.

3.3.2 KV Cache Transmission

Challenge 2: How to enable high-speed KV cache transfer for elastic scaling?



- **Fast path**
  - control-path w/ RDMA-based RPC[1]
  - data-path w/ Kernel-space RDMA
- **Heterogeneous virtual memory address space**
  - RDMA for CPU part
  - GDR for GPU part
- **Other optimizations**
  - Prefetching and caching

1. Kaminsky, Anuj Kalia Michael, and David G. Andersen. "FaSST: Fast, Scalable and Simple Distributed Transactions with Two-sided RDMA Datagram RPCs."

3.3.3 Launch-Point Optimization

Input: Prefill instance P_i, Cluster state C, KV cache size K
Output: Target GPU g^*, Launch time t^*
// Phase 1: Three-Dimensional Time Estimation
1. Query DAG tracker for residual prefill time:
   $T_{\text{prefill}} \leftarrow P_i.\text{getResidualTime}()$
2. Retrieve candidate GPUs with RDMA connectivity:
   $G_{\text{cand}} \leftarrow \text{FilterGPUs}(C, \text{RDMA=True})$
3. **foreach** $g_j \in G_{cand}$ **do**
   a. Estimate GPU context initialization:
      $T^j_{\text{init}} \leftarrow \text{ModelInitEstimator}(g_j)$
   b. Calculate KV cache transfer time:
      $T^j_{\text{xfer}} \leftarrow \text{TransferEstimator}(P_i, g_j, K)$
   c. Compute ready time:
      $t^j_{\text{ready}} \leftarrow \max(T_{\text{prefill}}, T^j_{\text{init}} + T^j_{\text{xfer}})$
end
// Phase 2: Optimal Target Selection
4. Select GPU with minimal ready time:
   $g^* \leftarrow \arg\min_{g_j \in G_{\text{cand}}}(t^j_{\text{ready}})$
5. Determine launch timestamp:
   $t^* \leftarrow \max(0, T_{\text{prefill}} - T^*_{\text{xfer}})$
// Phase 3: State Synchronization
6. Lock g^* in distributed KV store:
   etcd.Put(g^*.id, "locked")
7. Dispatch launch command via kernel daemon:
   DAGDaemon.Send(g^*, t^*, K)
8. Update GPU telemetry:
   $C \leftarrow \text{UpdateClusterState}(C, g^*)$
return g^*, t^*

Challenge 3: How to determine the optimal launch point for decoding instances?

- **Phase 1: Three-Dimensional Time Estimation**
  - Establishes the temporal foundation for scheduling decisions.
- **Phase 2: Optimal Target Selection**
  - Applies constrained optimization.
- **Phase 3: State Synchronization**
  - Finalizes the scheduling decision through atomic updates.

PART 04 前景展望

4 展望：超越Serverless LLM -> Serverless Agent

- **超越Serverless LLM： Serverless Agent**
  - 基本问题：安全容器/快速启动
  - 增大GPU复用（Multiplexing）密度：构造GPU OS服务更多租户[1]
  - 多租户托管复杂种类Agentic Workflow： 如何高效通信/编排？
- **RDMA for Serverless Agent的潜力**

RDMA特性	Cloud-Naive AI潜在瓶颈	RDMA的潜力
One-side RDMA	Agent/Tool冷启动	基于Remote Fork的**Agent安全容器**快速启动
内核空间RDMA	A2A/A2Tool第三方通信	基于RDMA构建**A2A/A2Tool共享内存地址空间**[2]
构建全局内存池化	Agentic Workflow资源弹性	**Agent/Tool拆分成小单元**，细粒度弹性伸缩[3]

1. Ideas源自：Coppock, Patrick H., et al. "LithOS: An operating system for efficient machine learning on GPUs." Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles. 2025.
2. Ideas源自：Lu, Fangming, et al. "Serialization/deserialization-free state transfer in serverless workflows." Proceedings of the Nineteenth European Conference on Computer Systems. 2024.
3. Ideas源自：Ruan, Zhenyuan, et al. "Quicksand: Harnessing Stranded Datacenter Resources with Granular Computing." 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25). 2025.

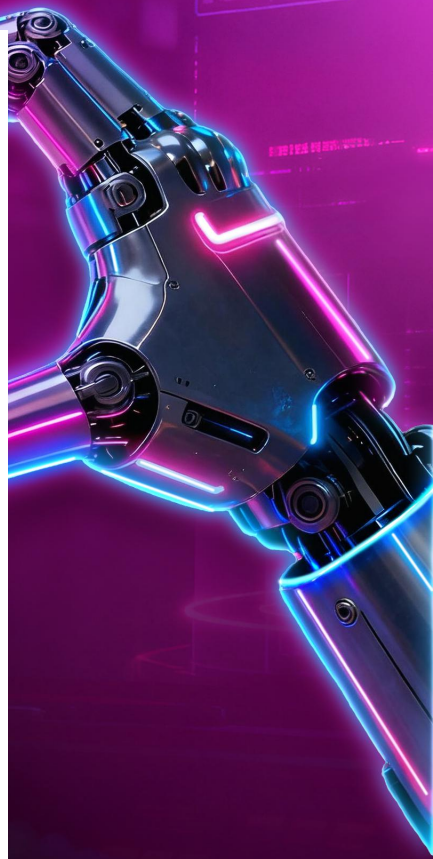
开源社 kaiyuanshe

云原生社区 Cloud Native Community

COSCon'25
第十届中国开源年会

众智开源 | Open Source, Open Intelligence

Thanks



刘明轩 西北工业大学计算机学院 应届博士生

liumingxuan@mail.nwpu.edu.cn