LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism

SOSP'24

Bingyang Wu Shengyu Liu Yinmin Zhong

Peng Sun Xuanzhe Liu Xin Jin

Peking University

https://github.com/LoongServe/LoongServe

YiFan Hu

2025.5.16

Contents

- Background & Motivation
- LoongServe
- Evaluation
- Thinking

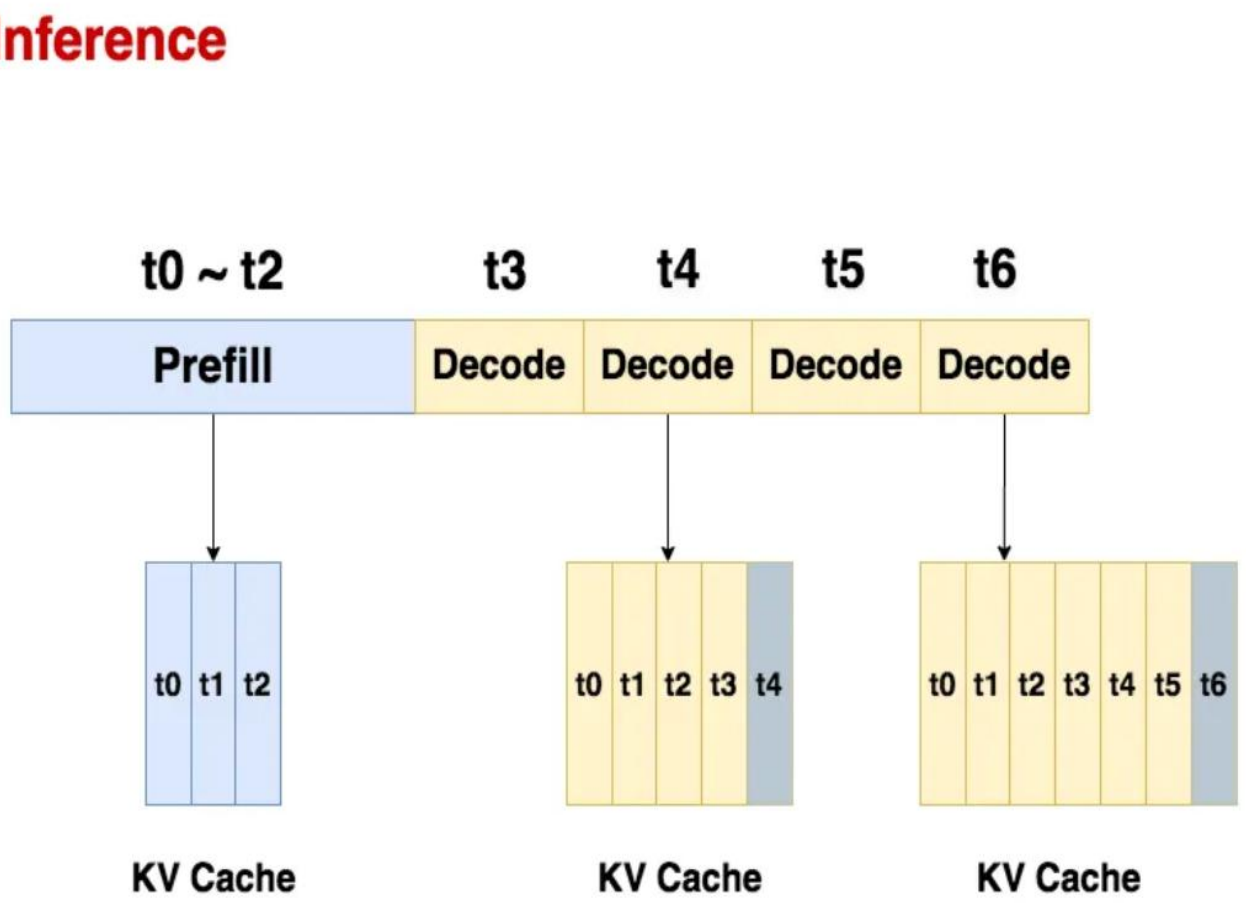
Background

Prefill

- 输入prompts生成qkv
- 存入KV Cache

Decode

- 新产生的 tokens 生成 qkv
- 计算与之前tokens的attention



Prefill和Decode负载不均衡

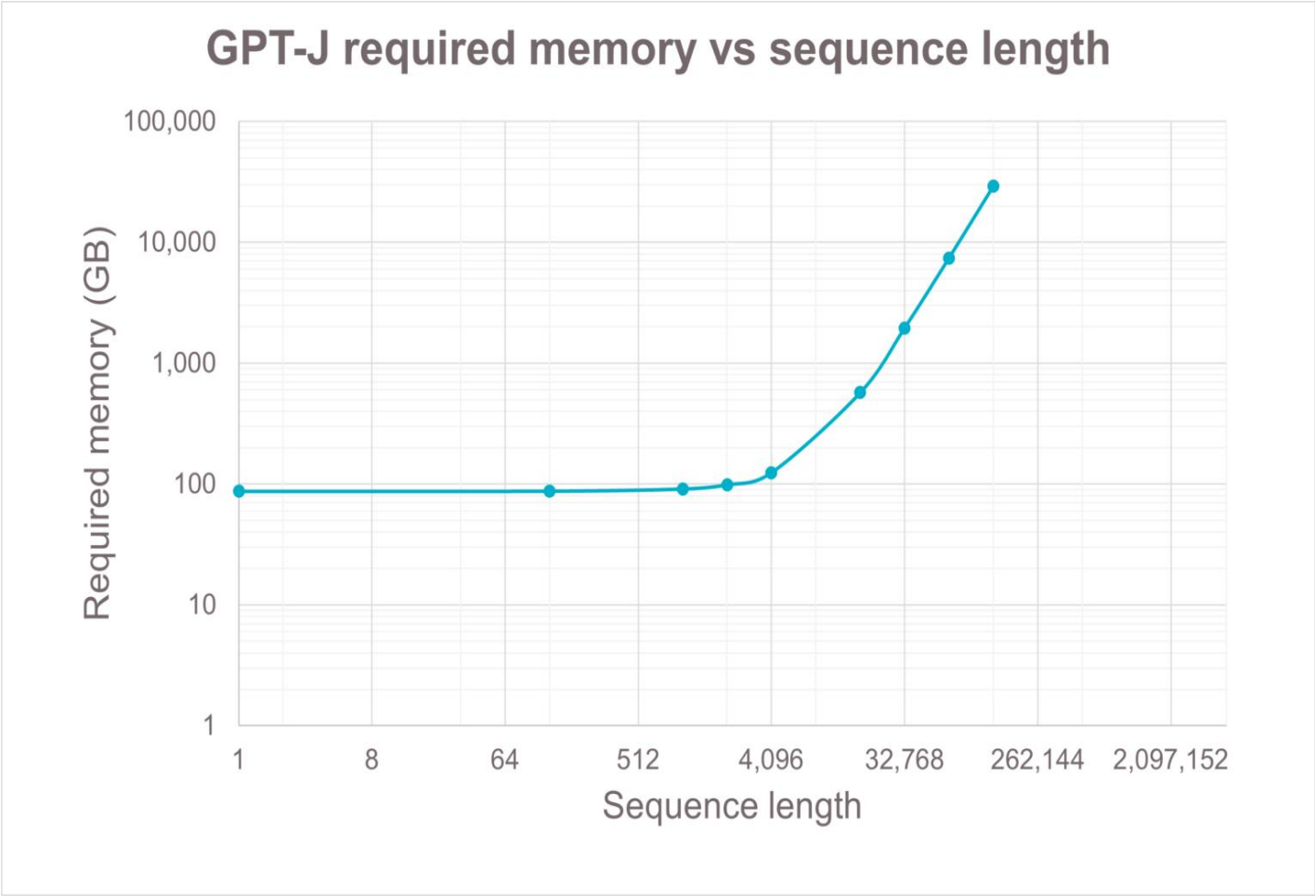
Background

长上下文LLM的崛起

- 上下文窗口2K -> 1M

推理显存不足

- 1M tokens = 488GB



长上下文导致Prefill和Decode更加不均衡

加速推理方案

技术	作用类别	Prefill	Decode	典型局限
Flash Attention	内核级加速	✔	✔	- 单卡优化
Flash decoding	内核级加速	✖	✔	- Prefill 不适用 -单卡优化
Tensor Parallelism (TP)	模型并行	✔	✔	- 并行度固定 - 短句利用率低

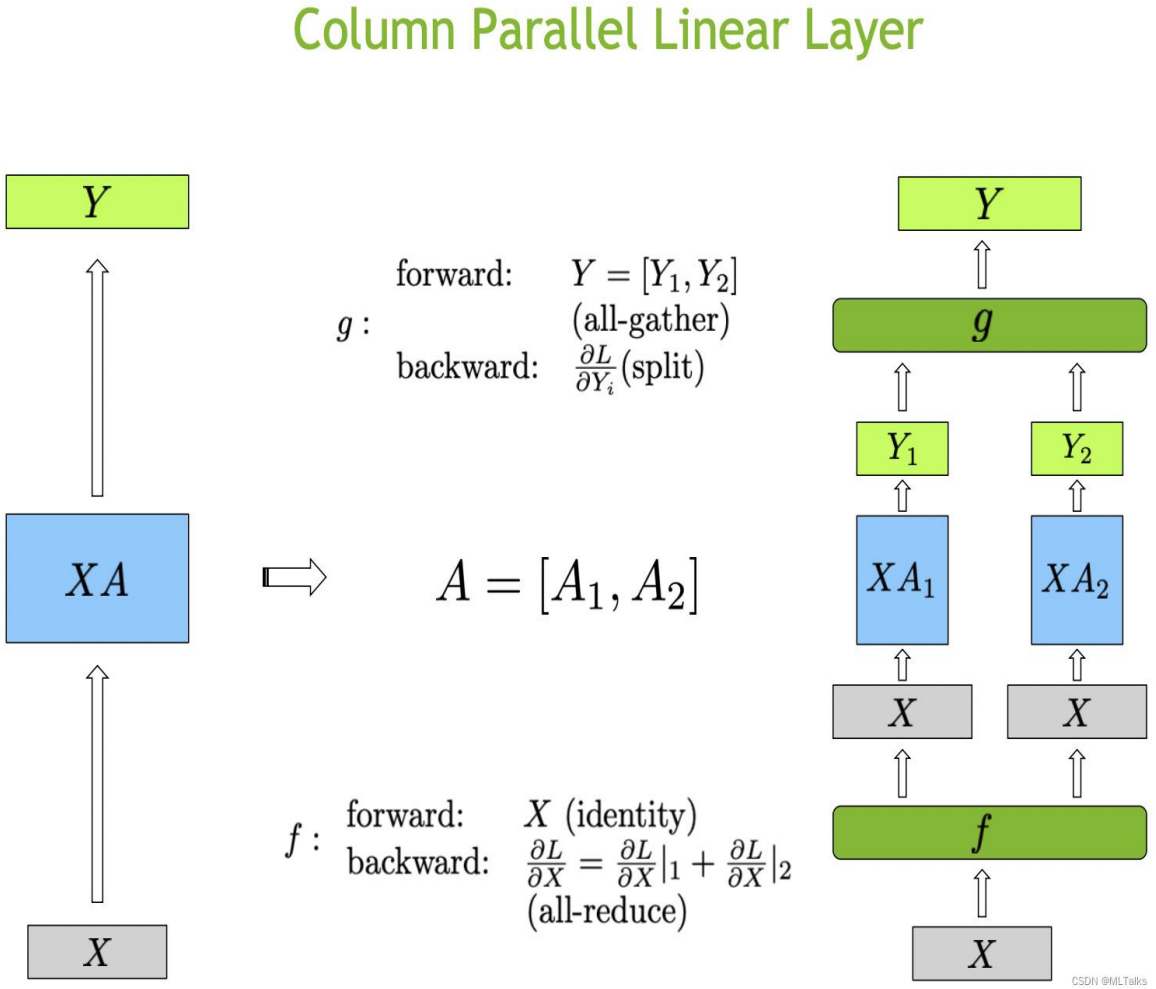
Tensor Parallelism

思想

- 划分模型参数

缺点

- 在部署前预先确定并行度，若修改并行度，需要重新切分，再加载到GPU中
- 通信量与tokens数无关，1000tokens和10tokens使用相同的GPU，短句的利用率更低



长上下文处理方案

技术	作用类别	Prefill	Decode	典型局限
Chunked Prefill	Prefill 优化	✔	✔	- 通信开销大 - 资源争用
Prefill-decoding Disaggregation	阶段隔离	✔	✔	- 通信开销大 - GPU内存碎片化

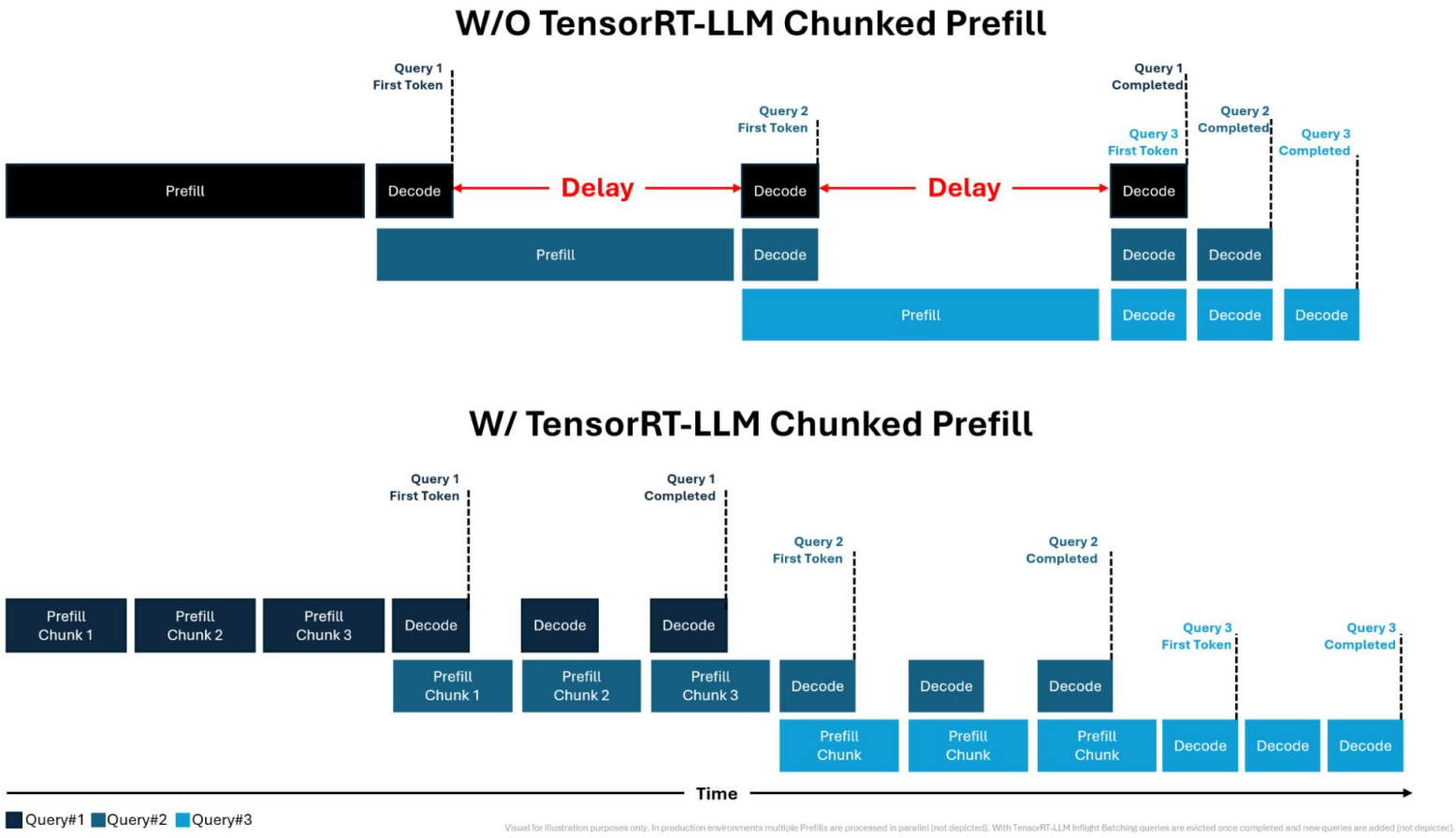
Chunked Prefill

思想

- 拆块 Prefill，Decode 插在 Chunk 之间执行

缺点

- 每个Chunk结束后同步KV，Prefill与Decode不在同一组时传递KV开销大
- 两个阶段在同一组GPU抢资源



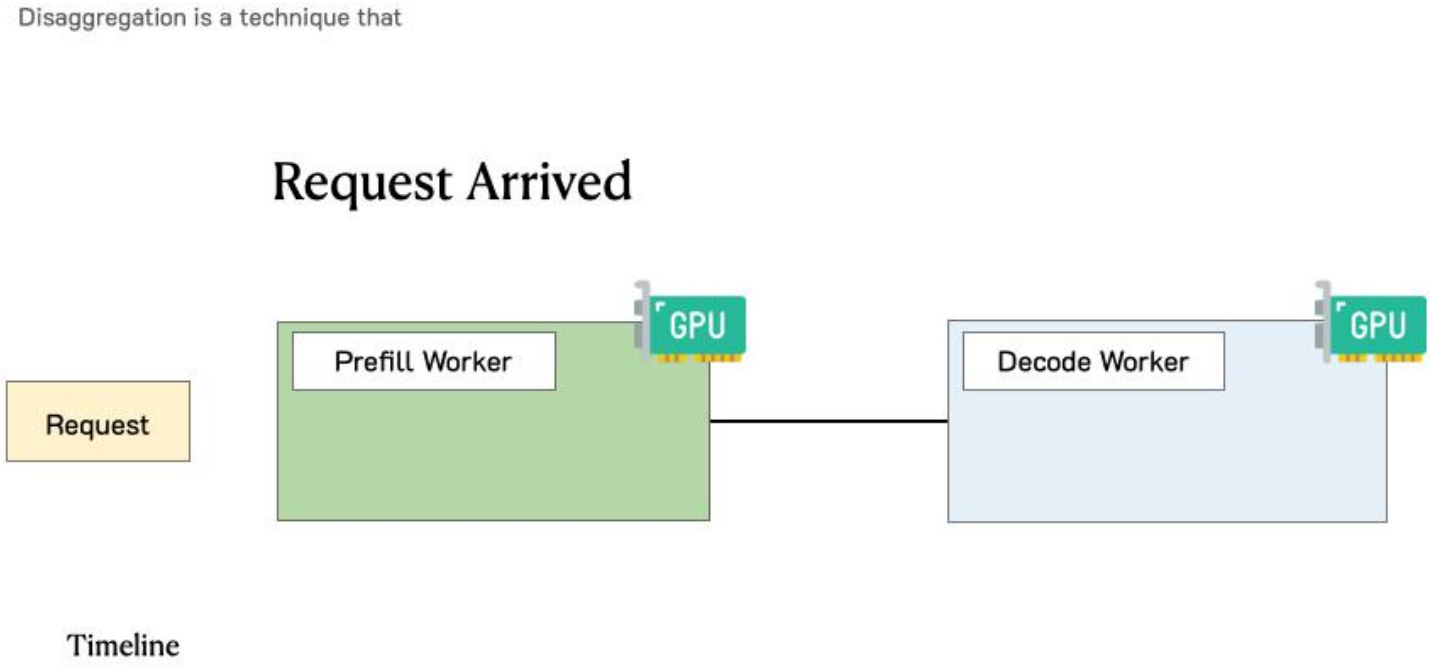
Prefill-Decode Disaggregation

思想

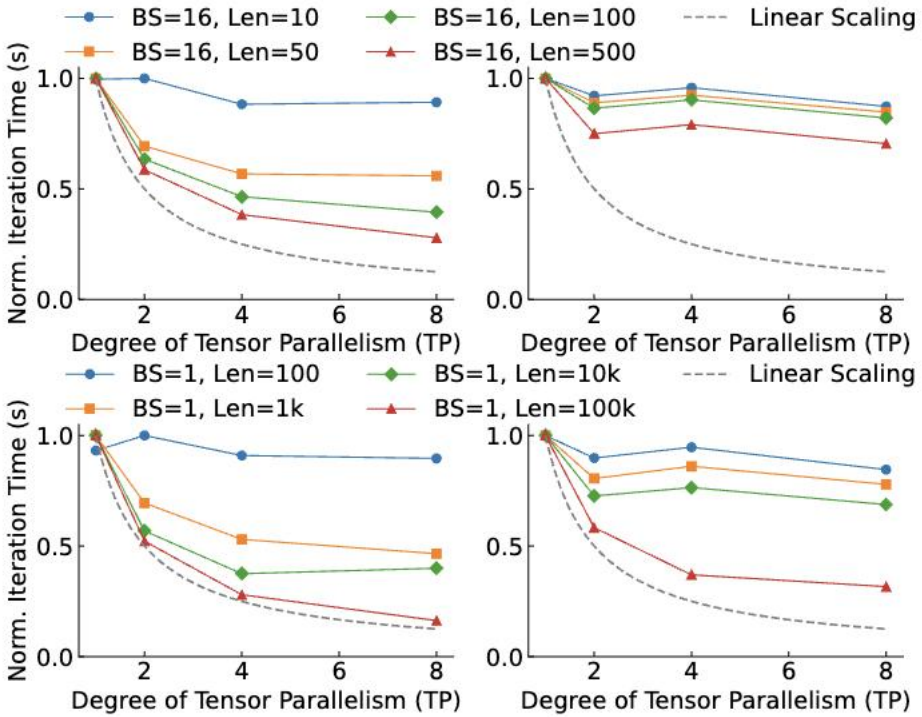
- Prefill和Decode使用不同的GPU组

缺点

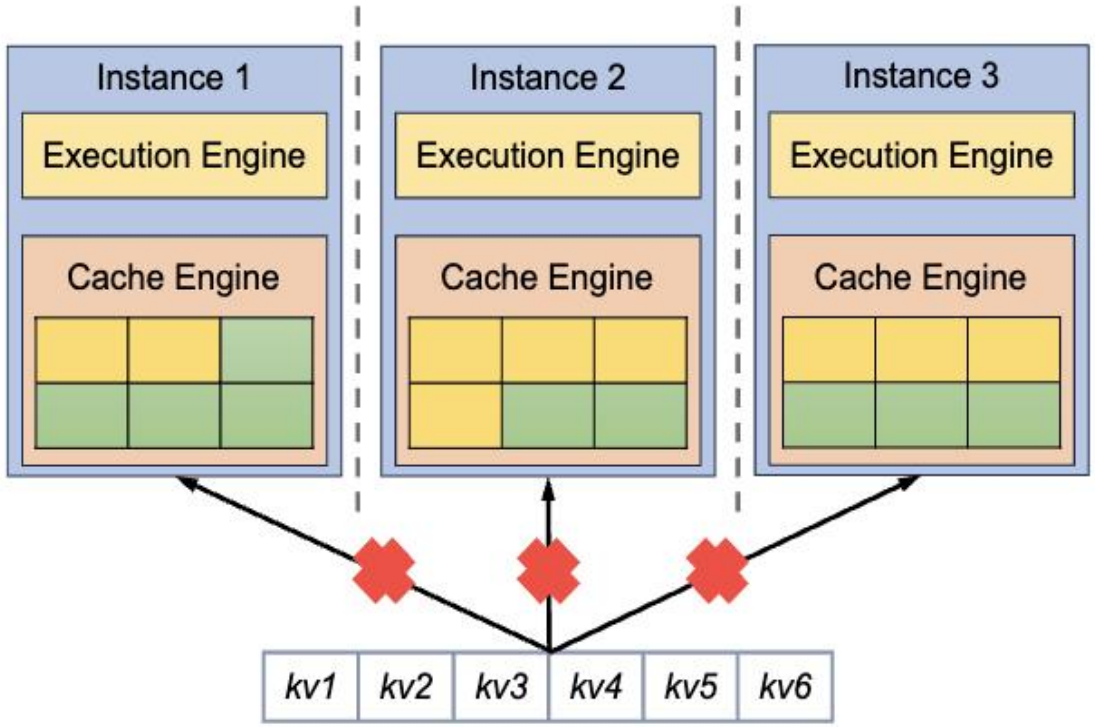
- 迁移完整的kv，产生巨大的通信开销
- 组与组的隔离导致显存碎片化



Motivation



并行效率曲线
左（Prefill）右（Decode）



GPU碎片化

Motivation-Sequence Parallelism

思想

- 输入序列划分并置换，分派

给不同的instance（包含多个GPU）

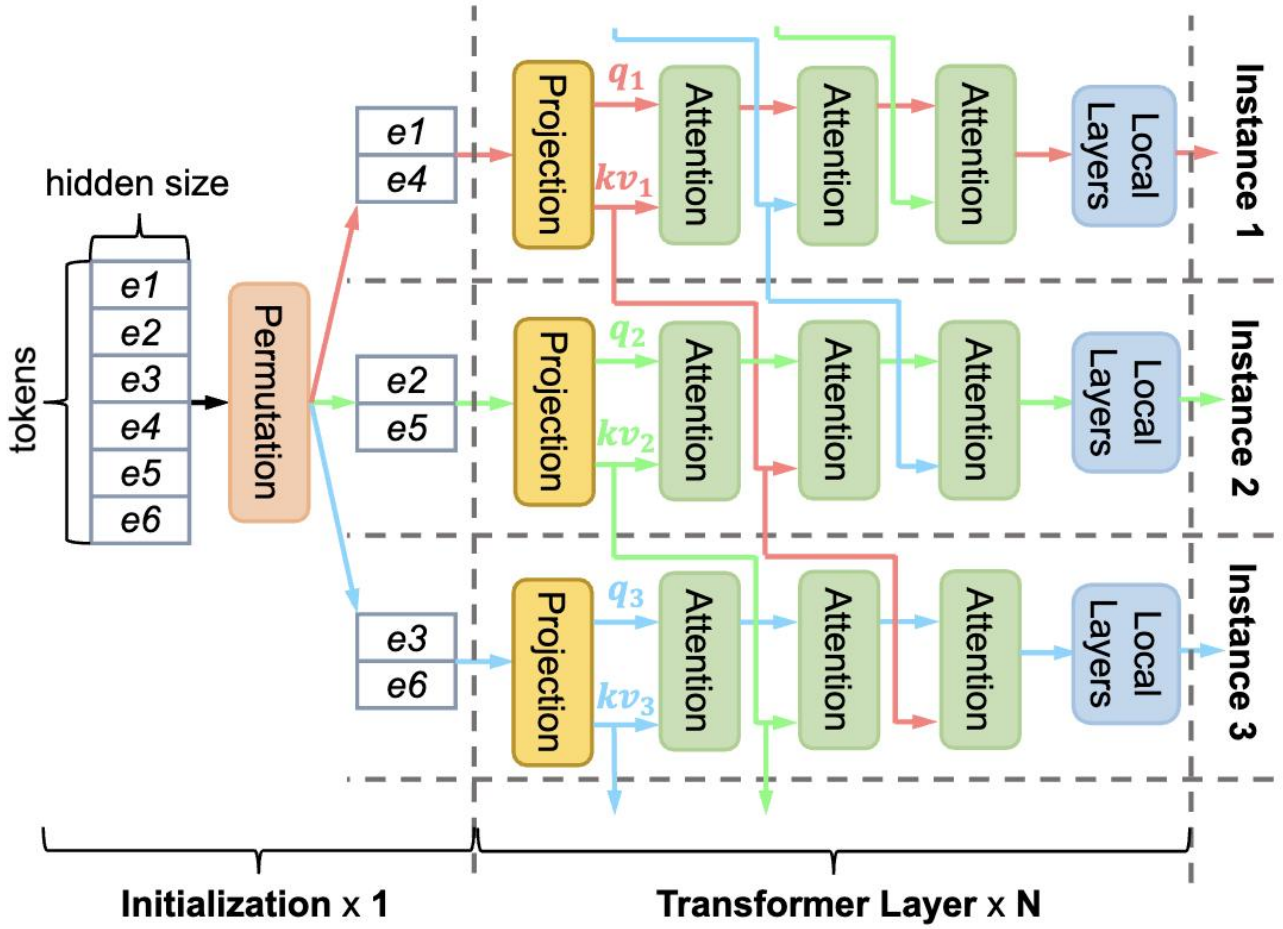
- 轮转计算注意力

优点

- 计算复杂度低
- 每个instance保存的kv小
- 无额外通信

缺点

- 不可用于Decode
- 并行度不可变



Contents

- Background & Motivation
- LoongServe
- Evaluation
- Related Work
- Thinking

LoongServe

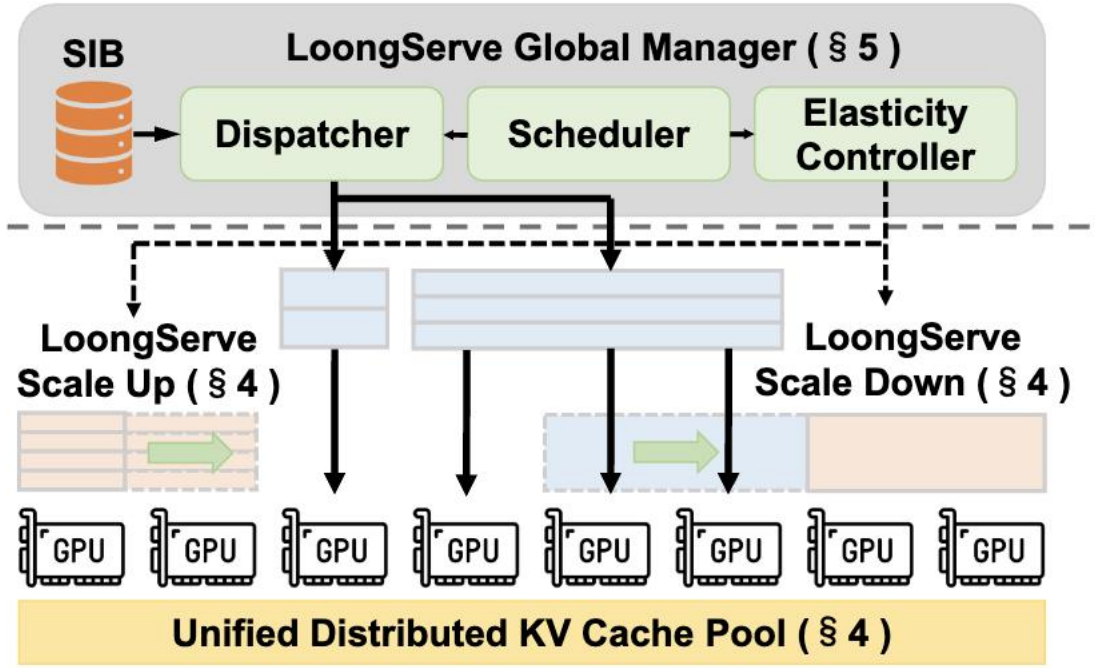
核心： Elastic sequence parallelism （ESP）

优点：

✓ Decode支持

✓ 动态可调的并行度（DoP）

✓ 灵活管理KV

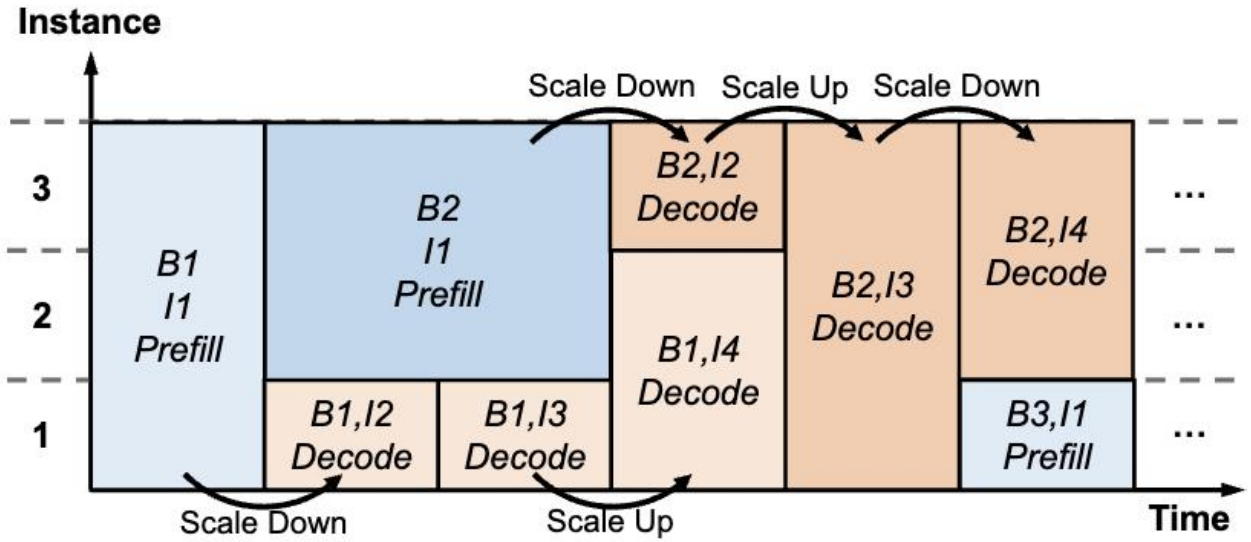


方法： ESP = Elastic Instances + Global Manager

Elastic Instances

每一个Instances具有

- 完整模型权重副本，避免权重重新切分的开销
- 相同的GPU数量，降低了调度复杂性
- 内部执行TP， Scale-up 阶段，可以并行执行多个 Master



Scale Up：一个并行组增加Instance
Scale Down：一个并行组减少Instance

Elastic Scale-down

发生阶段：

- Prefill阶段结束后，KV从一个并行组 R 迁移到一个新的并行组 R ′
- 为Prefill留下更多资源

原始方法：

- 全部KV Cache 从并行组A(规模大)->并行组B(规模小)，传输的数据量极大

挑战：

- 减少最后的传输消耗

并行组 R：包含多个Instance

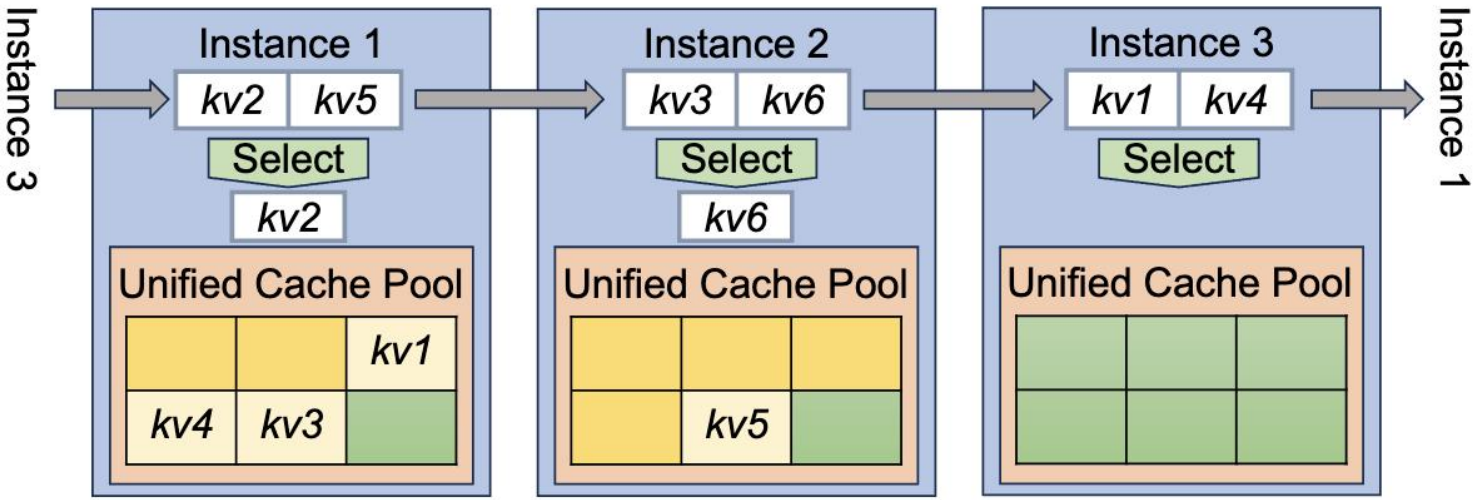


Figure 7. Elastic scale down in the prefill phase.

现方法：Proactive Migration，在轮转的过程中保存KV，不必在结束后一键搬运

Elastic Scale-up

发生阶段：

- Decode产生的KV Cache超过显存
- 减少延迟，希望快速结束现有的Decode任务

原始方法：

- 将KV Cache从InsA (资源少) -> InsB (资源多)

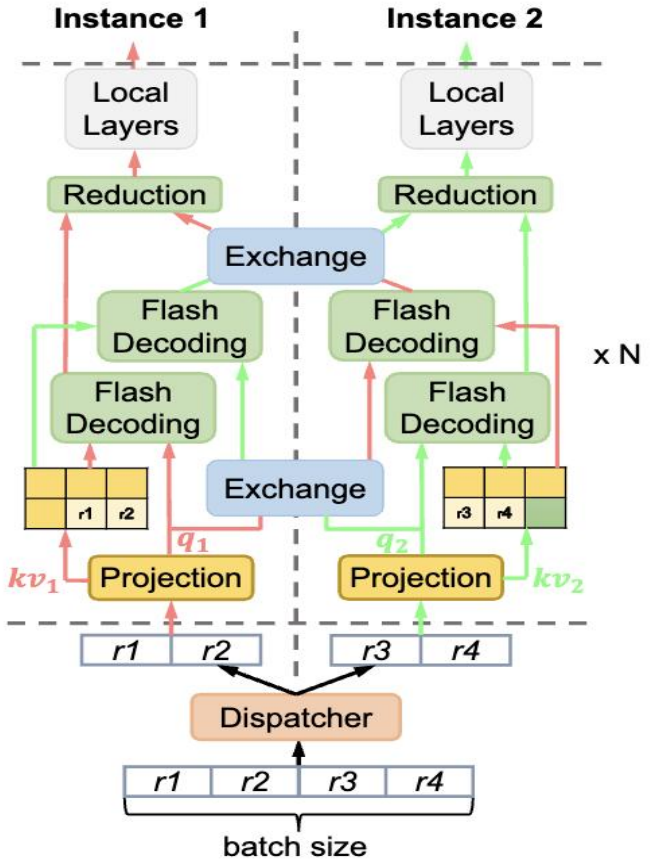
但是需要传输完整的KV Cache，带来通信开销

挑战：

减少通信开销

Single-master：主卡 InsA 保留并持续追加全部 KV Cache，

遇到显存吃紧时再临时拉一张副卡 InsB 来帮忙



multi-master distributed Decode

一个并行组多master，负责不同请求

Global Manager

负责：

- 挑选请求，进行评估，进入Prefill
- 为这些请求分配多少Instance，

先用空闲，不够可以抢占

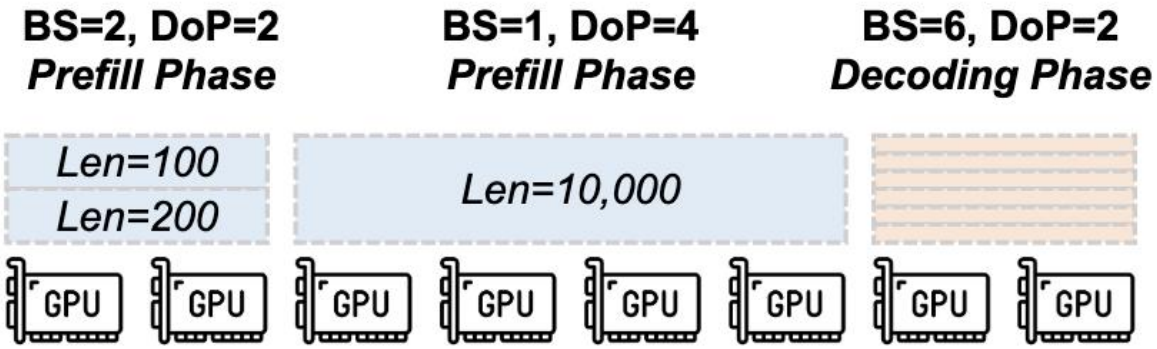
- 这些请求划分为哪几个batch，怎

么分配并行度(DoP)，能最好的节约资源

- Decode是否需要scale-up与

scale-down

batch：Prefill 或 Decode 时 在同一并行内同步执行的一组请求



BS：BatchSize　　DoP：并行度

蓝色Prefill　　粉色：Decode

Dispatching

1. GPU Memory约束

挑待处理队列 P 里的请求按照FCFS放进预取集合 Rp，不能因为显存不够导致以后把长序列驱逐出去重算，造成资源浪费。

评估：

现在：当前所有GPU的KV Cache 可放下请求所需 → OK

未来：对每个请求估算生成的最大长度，即最差占多少KV

现在 + 未来 < 总 KV Cache　满足条件

Dispatching

2. GPU计算约束

Prefill阶段，GPU拥有越多请求越划算，直到拐点，GPU不再负担得起请求，拐点参照SIB预先 profiling 数据（也负责计算T）

评估：

用分析模型估算Rp (请求)在Ep (instance)上跑一轮的时间T(Rp, Ep)，若T超过拐点，则停止塞入请求

$$T_p(R) = \alpha_p + \beta_p \cdot \sum_{r \in R} r.\text{input\_len} + \gamma_p \cdot \sum_{r \in R} r.\text{input\_len}^2 \quad (7)$$

T = 固定开销 + ffn耗时 * 请求长度 + attention耗时 * 请求长度²

Dispatching

2. GPU计算约束

定义请求X，当前迭代原本不能直接放进来，但可以尝试通过抢占Decode阶段instance中的空闲KV槽位插入

评估：

收益-代价分析，Gain为插入X带来的性能提升，Cost为被抢占的请求被延迟的代价

$$\text{Gain} = \sum_{r \in R'_{p,i}} \frac{(\text{AvgLat}_d - \min(B_{p,i}.\text{exec\_time}))^+}{r.\text{input\_len}}$$

$$\text{Cost} = \sum_{r \in B_{p,i}} \frac{T(R_p \cup R'_{p,i}, E_p \cup G_{p,i})}{r.\text{output\_len}}$$

收益 = (平均Decode时延 - 最小执行时间)/请求X的输入token长度

代价 = T(所有加入的请求, 占据的GPU)/被抢占的请求已经生成的token长度

Elastic Instance Allocation

作用：

为进入Prefill阶段的请求初步确定各自可用的elastic instance。当KV不够或需要提升性能，挑选最空的instance (e_{min})抢占

评估：

收益-代价分析，Gain为把e_{min}给请求后节省的总时间，Cost为浪费的时间

$$\text{Gain} = \sum_{r \in R_p} \frac{T(R_p, E_p) - T(R_p, E_p \cup e_{min})}{r.\text{input\_len}}$$

$$\text{Cost} = \sum_{r \in R_p} \frac{V(e_{min})}{\text{avg\_bandwidth} \cdot r.\text{input\_len}}$$

收益 = (请求R在instance组E做一次迭代的时间 - 请求R在instance_E+instance_e_{min}做一次迭代的时间) / 请求长度

代价 = e_{min} instance已有的KV Cache / (instance搬运的平均带宽 * 请求长度)

Batching

思路：在确定请求集合Rp和instance集合Ep后，需要将请求分成若干batch，并分配GPU，要求是所有请求的Prefill时间最短，采用动态规划计算

预处理：1.请求从大到小2.instance从KV Cache少到多排

状态定义：D[j, i] = 从请求j 到i 的token数 V[l, k]= instancel到k的KV容量

f[i][k] = 前k个instance，处理前i个请求达到的最小Prefill时间

状态转移：

$$f[i][k] = \min_{\substack{0<j\le i, 0<l\le k, \\ D[j,i]\le V[l,k]}} (f[j][l] + T(R[j,i], E[l,k]))$$

朴素枚举为$O(n^2m^2)$, 采用四边形不等式优化，降低为$O((n+m)^2)$

$$split_{req}[i][k-1] \le split_{req}[i][k] \le split_{req}[i][k+1],$$

$$split_{ins}[i-1][k] \le split_{ins}[i][k] \le split_{ins}[i+1][k].$$

Contents

- Background & Motivation
- LoongServe
- Evaluation
- Thinking

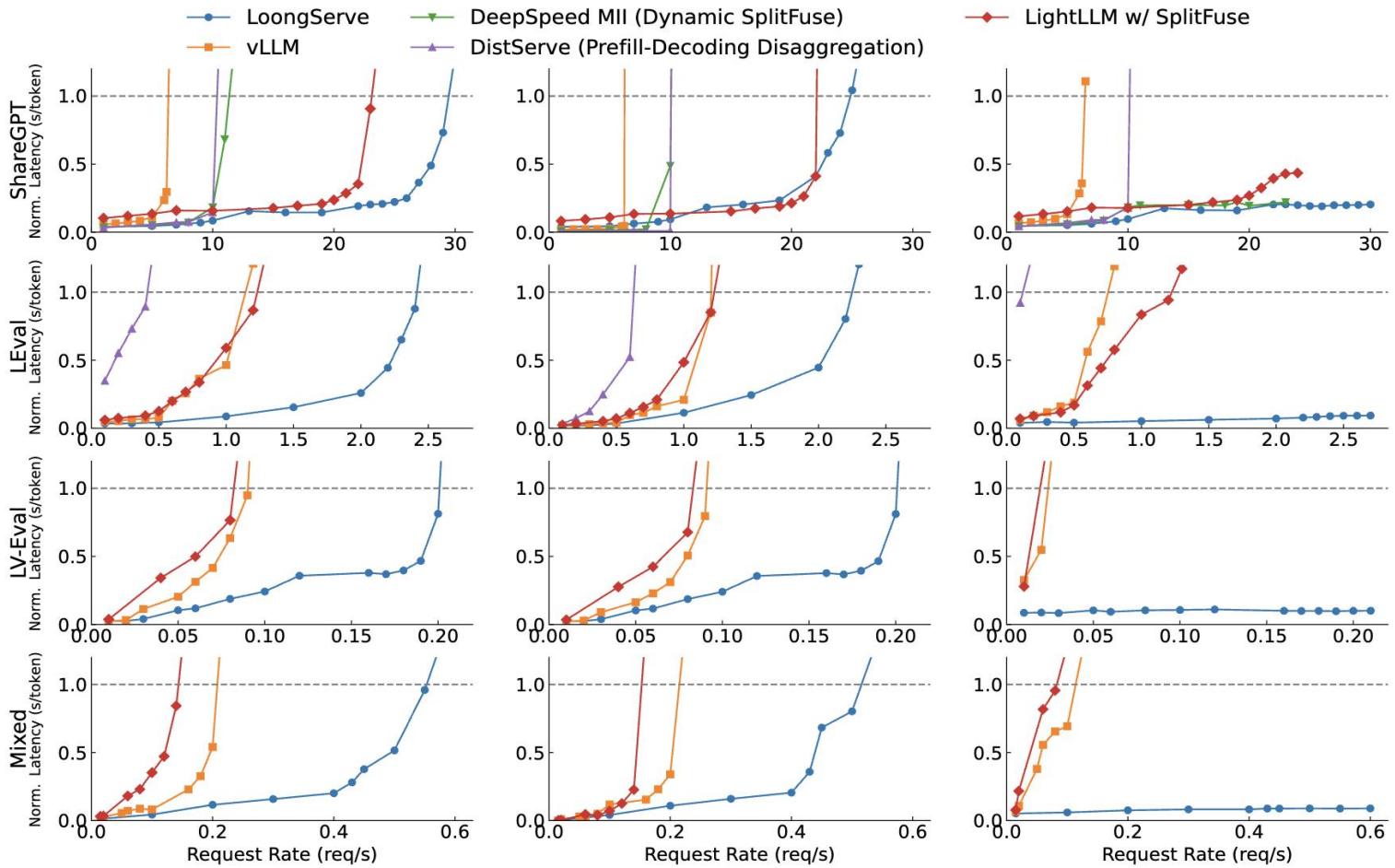
Evaluation

设置类别	具体配置
模型	LWM-1M-Text（1 M context， Llama-2-7B 架构）
单机硬件	8× NVIDIA A800 128 CPU 2048 GB 主存 4×200 Gbps InfiniBand NVLink 400 GB/s
软件栈	PyTorch 2.0.0; CUDA 12.2
流量模型	ShareGPT / L-Eval / LV-Eval / Mixed
对比基线	vLLM (TP=8) LightLLM+SplitFuse (TP=8) LoongServe (TP=2, ESP=4) DeepSpeed-MII (TP=8， SplitFuse， 仅限 ShareGPT) DistServe (prefill 4 GPU+decode 4 GPU, DoP=4)
评价指标 & SLO	归一化端到端延迟 / token归一化 prefill 延迟 / input_token归一化 decode 延迟 / output_tokenSLO=25×inference latency

Evaluation

端到端吞吐&延迟

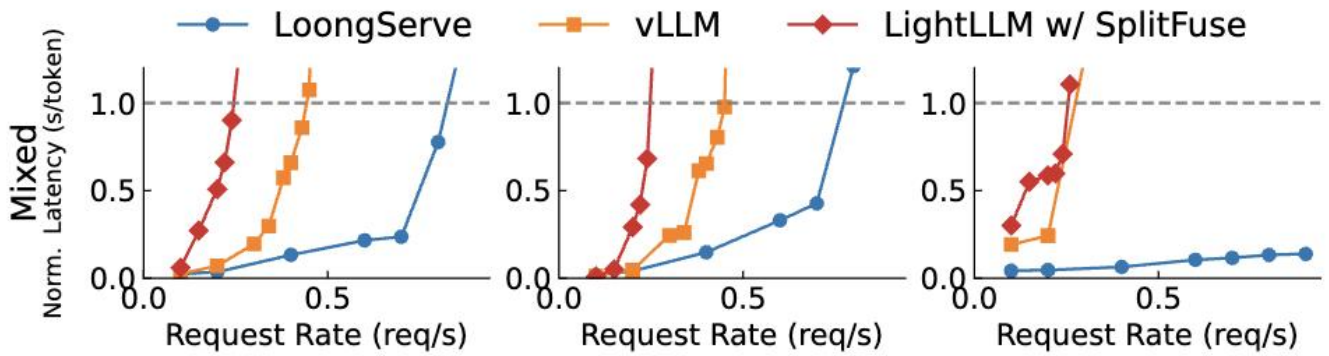
Baseline	Total 吞吐提升 (×)	Prefill 吞吐提升 (×)
vLLM	4.64	4.00
DeepSpeed-MII	3.85	3.37
LightLLM	3.85	3.37
DistServe	5.81	3.58



Avg. per-token latency. Avg. input token latency. Avg. output token latency.

Evaluation

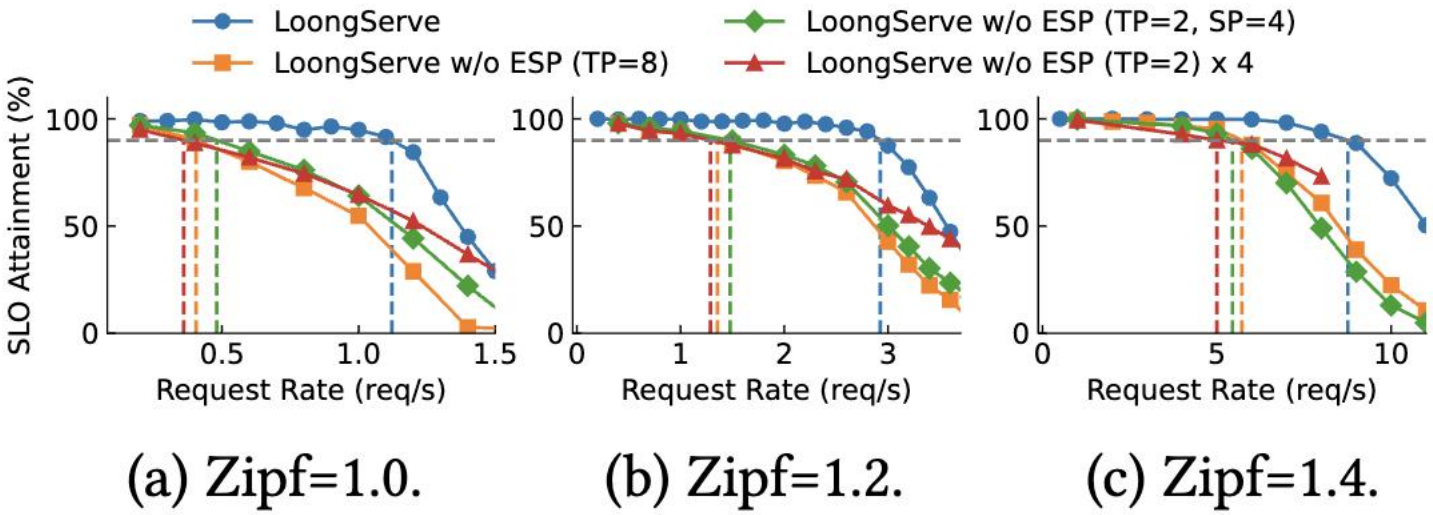
跨节点吞吐&延迟



Baseline	总吞吐提升 (×)	Prefill 吞吐提升 (×)
vLLM (TP=8)	1.86	1.72
LightLLM w/ SplitFuse (TP=8)	3.37	3.11

Evaluation

消融实验：ESP 贡献



(a) Zipf=1.0. (b) Zipf=1.2. (c) Zipf=1.4.

LoongServe w/o ESP (TP=8)：纯张量并行

LoongServe w/o ESP (TP=2, SP=4)：2-way 张量 + 4-way 序列并行

LoongServe w/o ESP (TP=2)×4：4 个副本各用 2-way TP

LoongServe（TP=2, ESP=4）：启用弹性序列并行的完整方案

Zipf=1.0 提升 2.33× Zipf=1.2 提升 1.98× Zipf=1.4 提升 1.53×

ESP是核心动力

Evaluation

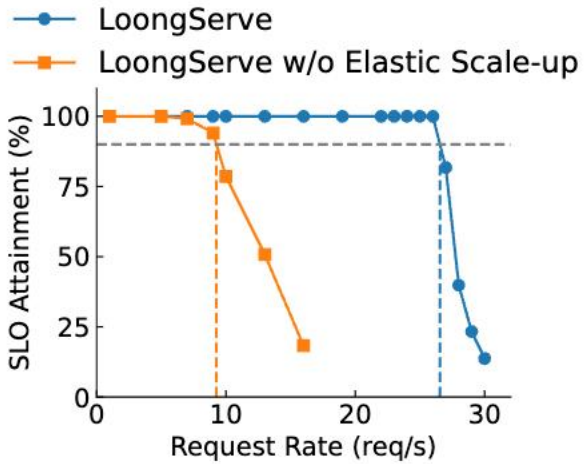
消融实验：Scale-up 贡献

1. P90 Goodput 对比

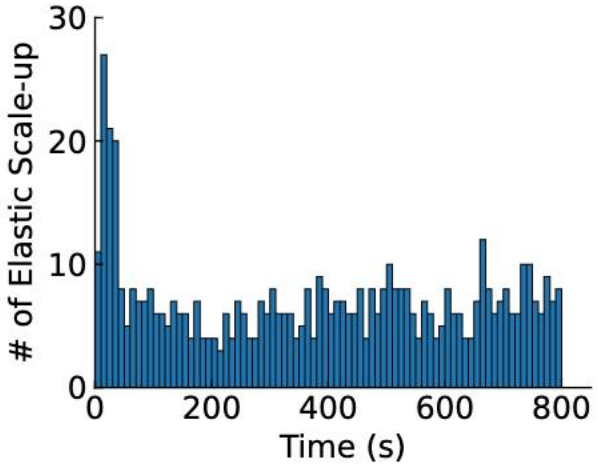
- 开启提升2.87×

2. Scale-up 触发频率

- 0.7次/s的频率，说明Decode阶段必须有Scale-up才能避免吞吐下降



(a) Effectiveness of elastic scale-up.



(b) Frequency of elastic scale-up.

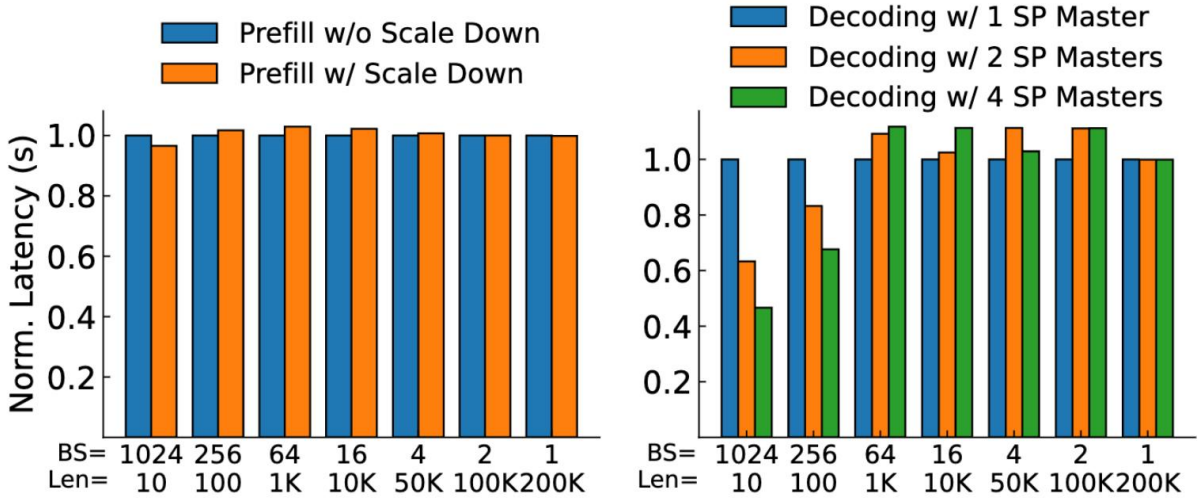
Evaluation

Scaling 开销

1. Scale-down 开销

- 不同的 batch sizes 和输入长度下

前向一次的延迟只增加了不到 2%



(a) Scale down overhead. (b) Scale up overhead.

2. Scale-up 开销

- 对于大 batch每次迭代延迟减少约 2×
- 对于小 batch带来小于10% 的额外开销

Contents

- Background & Motivation
- LoongServe
- Evaluation
- Thinking

Thinking

这篇Paper能做哪些优化？

1. ESP在多个地方都使用了SIB里面的分析模型和预先profiling数据，这个数据是否可以更加通用？（可以使用强化学习持续调整数据）

2. 论文强制要求每个instance能力相同，这不大符合实际要求，如果Instance用异构集群，那e_{min}和拐点都不准确。（赋予权重）

3. 动态抢占虽然可以提升Prefill阶段的吞吐，但是会恶化已有请求的输出延迟，可能会导致有些长的请求频繁退后。（很像操作系统的进程调度，为每个请求设置优先级）

Thinking

这篇Paper能用在自己的工作吗?

1. LoongServe 的 Multi-master 思路可以直接迁移到异构边缘集群的调度上，动态分配计算任务，最大化利用不同设备的计算资源。

这篇Paper能不能泛化?

1. 兼容图像、音频、视频等多模态数据，可以把 KV Cache 扩展到支持多模态特征；例如在多模态 Transformer 中，可以将每个实例负责不同模态的计算任务

Thanks

YiFan Hu
2025.5.16