北京 | Beijing

SGLang在微信搜一搜的应用实践

曹皓 | 腾讯专家工程师

联合主办

SGL 亚马逊云科技 startups 美团 Meituan 文心大模型 Omni-AI 趋境科技 APPROACHING.AI MONOLITH NVIDIA

微信搜一搜

连接优质内容

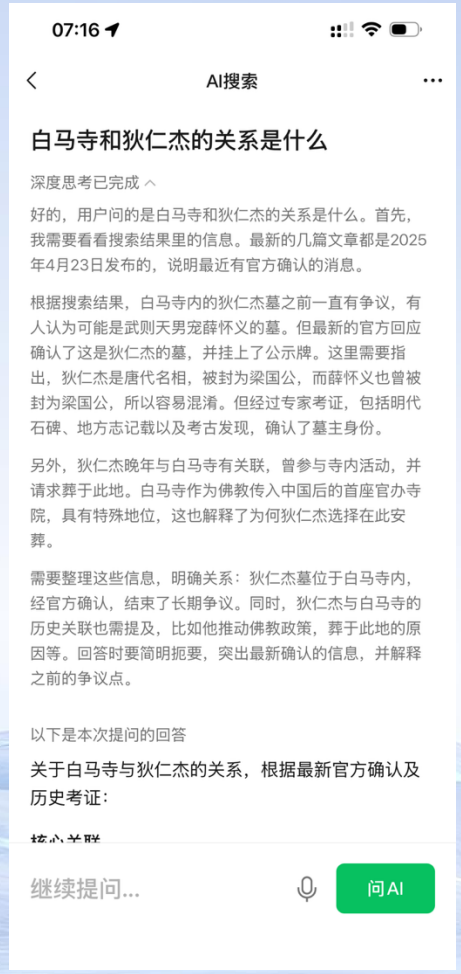
服务美好生活

新兴媒介：视频搜索

经典载体：图文搜索

微信特色：账号与服务搜索

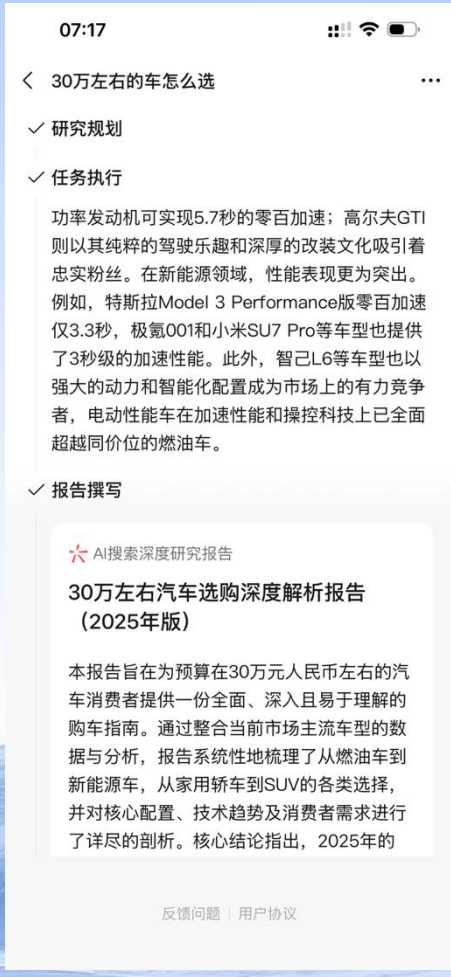
LLM在搜一搜中应用举例



深度思考

快速回答

视频号前链



深度研究

应用特点和推理目标

基础体验

- LLM 替换 BERT，改善长尾体验
- 自研 1B dense 模型
- 只生成一个 token

智能问答

- LLM RAG 替代 MRC
- 几十B dense - 几百B MoE模型
- 输入长，输出短

目标1：高效支撑海量推理流量

- 在有限 GPU 资源下**提升推理吞吐**（QPS）、**降低单位请求成本**

目标2：在高吞吐约束下实现首字快、生成流畅的推理响应体验

- 搜索场景偏**即时消费**，用户养成**秒级响应心理预期**
- 要求响应快（**首字速度TTFT）**，生成速度流畅（**生成速度TPOT**）
- 流畅一般标准：一秒50字+（20ms/字）

SGLang为微信搜一搜提供高性能LLM推理支持

实践1：生成流畅度优化

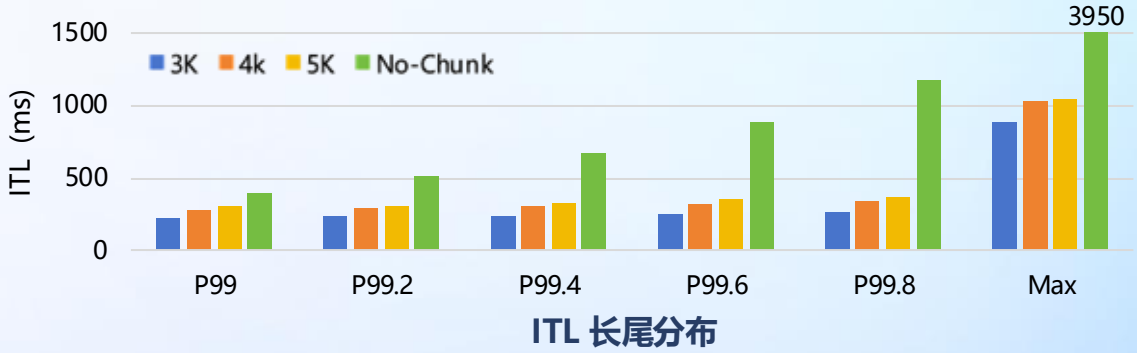
现有方法的局限性

Chunked prefill

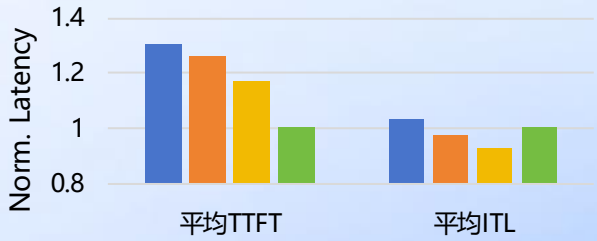
- TTFT 上升 ~25%
- 卡顿原因：chunked prefill 用 TTFT 换取 TPOT

No-chunk 混合部署

- P99 ITL 上升 ~35%，卡顿率高
- 卡顿原因：prefill 优先导致 decode 执行后挪



ITL 长尾分布



平均时延（归一化）

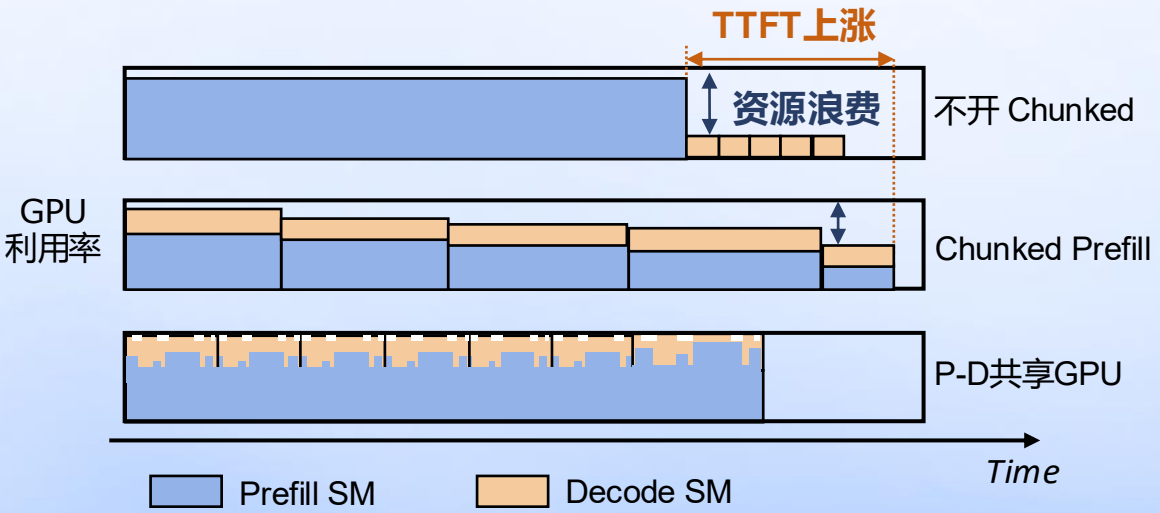
请求卡顿率

如何降低P-D的相互影响?

Chunked Prefill 分析

Chunked prefill 特征

- 用吞吐和 TTFT 换取 TPOT
- Chunk 冗余访存：平均 GPU 资源利用率降低



性能剖析

- Prefill 阶段 **Attention** 的 **Tensor Core 利用率**只有 **40%-80%**，显著低于 GEMM
- Prefill 阶段**小算子**及**通信**操作时间占比超15%，造成GPU资源浪费
- Decode 阶段访存密集，**SM**（Streaming Multiprocessor）**利用率低**

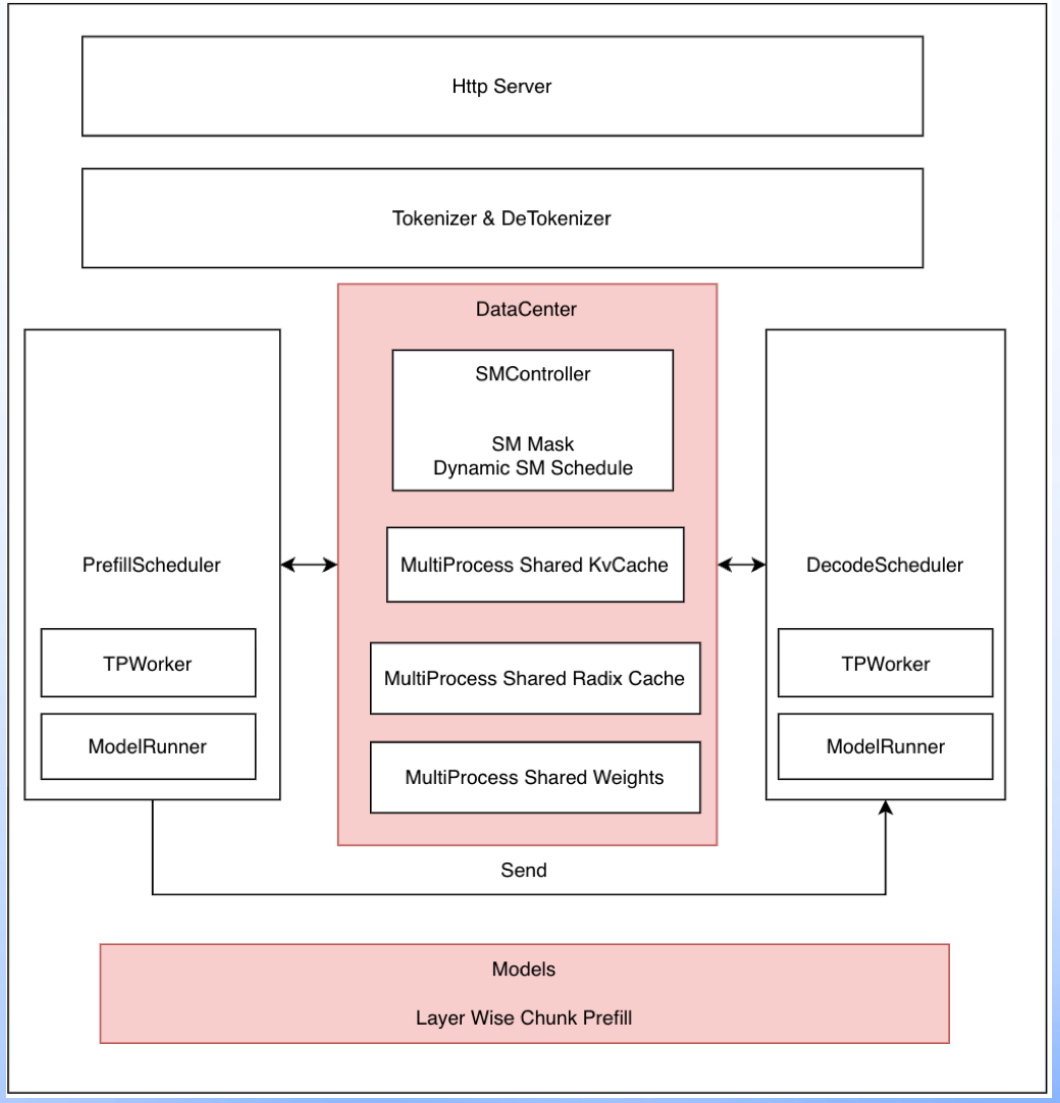
解决思路

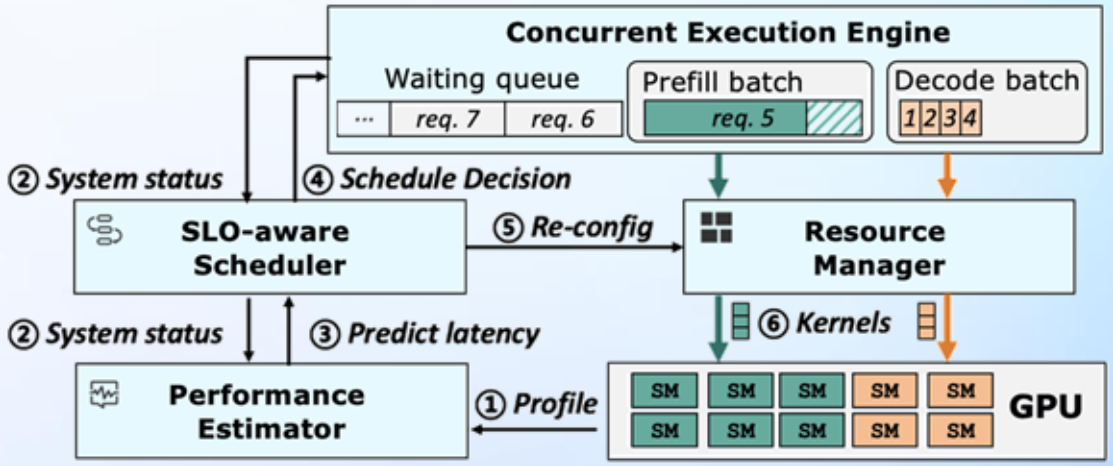
P-D并发执行共享GPU

充分利用计算和带宽

https://arxiv.org/abs/2504.19516

整体方案





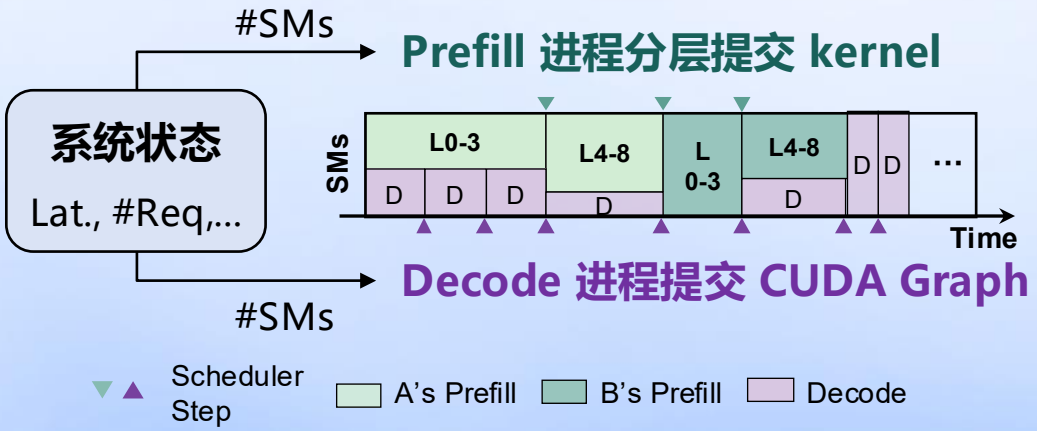
P/D 分离到同 GPU 的两个进程并发执行

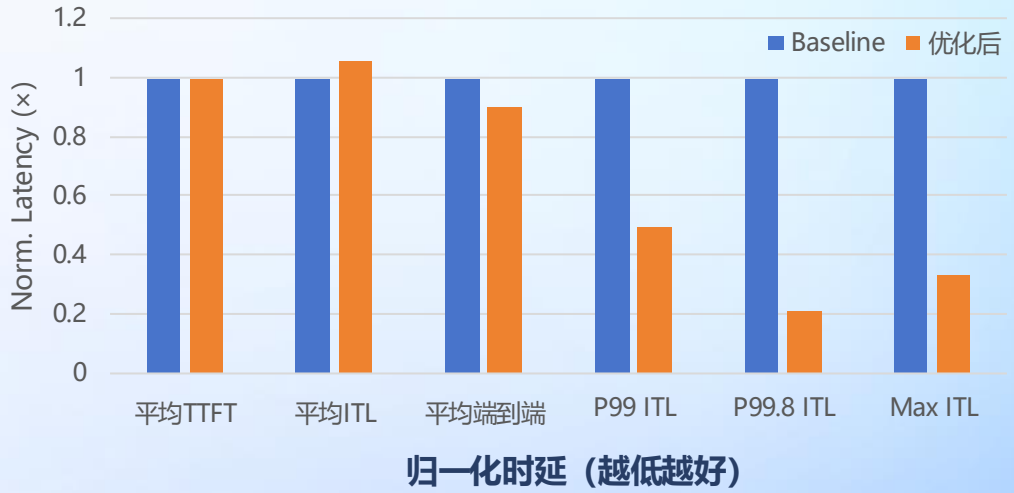
- **实时性能预测：**指导调度和资源管理
- **SLO感知调度：**决策所需 SM 数量
- **GPU资源管理：**SM 划分及进程间元数据交换
- **并发执行引擎：**多进程避免 GIL，跨进程共享KV Cache 和模型权重

https://arxiv.org/abs/2504.19516

执行流程

- **进程模型：**P/D 独立进程，通过 Nvidia MPS (Multi-process Service) 实现 GPU 空间共享
- **调度层：**P/D 进程共享系统状态，根据动态调整 SM 配比
- **显存管理**：CUDA IPC 实现KV cache、权重共享，P/D之间只需传递请求元数据





优化效果

- 端到端和长尾延迟降低
- 请求卡顿率 81% -> 14%
- GPU利用率提升

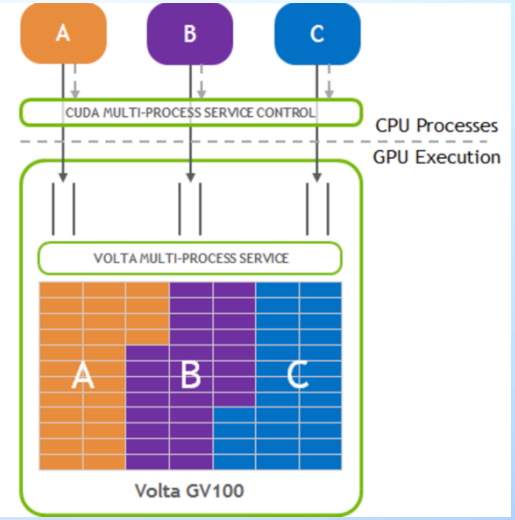
https://arxiv.org/abs/2504.19516
https://docs.nvidia.com/deploy/mps/index.html

多进程共享 GPU

GPU 空间共享

- 多进程并发使用 GPU 默认使用时间片划分
- 开启 MPS 后可进行空间划分
- MPS 对应用透明，一行命令即可开启

```
# 开启MPS
nvidia-cuda-mps-control -d
# 关闭MPS
echo quit | nvidia-cuda-mps-control
```



GPU 空间划分

- 多任务共享 GPU 时，设定资源配额（#SM）能**缓解**任务间**资源竞争**

	粒度	作用域	重划分方式和开销	动态划分	原生支持
MPS 环境变量	TPC (2 个 SM)	进程	重启进程	×	✓
CUDA Green Context	2-8 个 SM	CUDA stream	创建 Context 有较大时间和显存开销	✓	✓
TPC 掩码划分 (libsmctrl)	TPC	CUDA stream	Kernel launch前修改掩码，几乎无开销	✓	×

https://docs.nvidia.com/deploy/mps/index.html
http://rtsrv.cs.unc.edu/cgit/cgit.cgi/libsmctrl.git/about/
https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__GREEN__CONTEXTS.html

跨进程数据共享

GPU 零拷贝显存共享

- CUDA IPC Memory Handler：几 kb 的可序列化对象
- 发送方调用 CUDA API 获取 handler 并发送
- 接收方使用 handler 将该显存**映射**到自己地址空间
- 实现进程间 KV cache、权重共享，读/写**无性能开销**

发送方

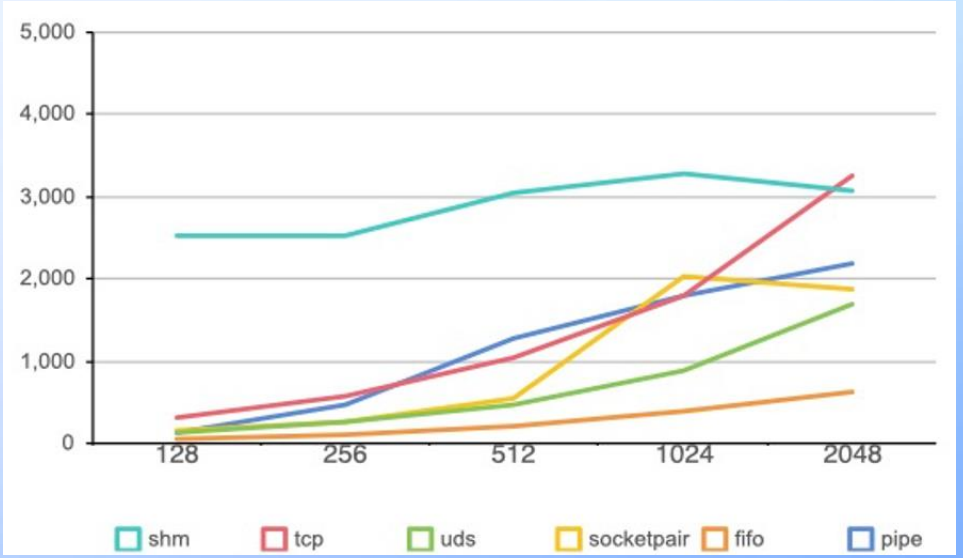
```
reconstructor, handle = torch.multiprocessing\
                                    .reductions.reduce_tensor(tensor)
send(reconstructor, handle)
```

接收方

```
reconstructor, handle = receive()
tensor = reconstructor(handle)
```

CPU 元数据传输

- 调度信息 (req. len., bs., 等) 需在P/D进程间共享，且高频读写
- 使用共享内存 (/dev/shm) 高效操作元数据



不同IPC方式吞吐量

https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__VA.html

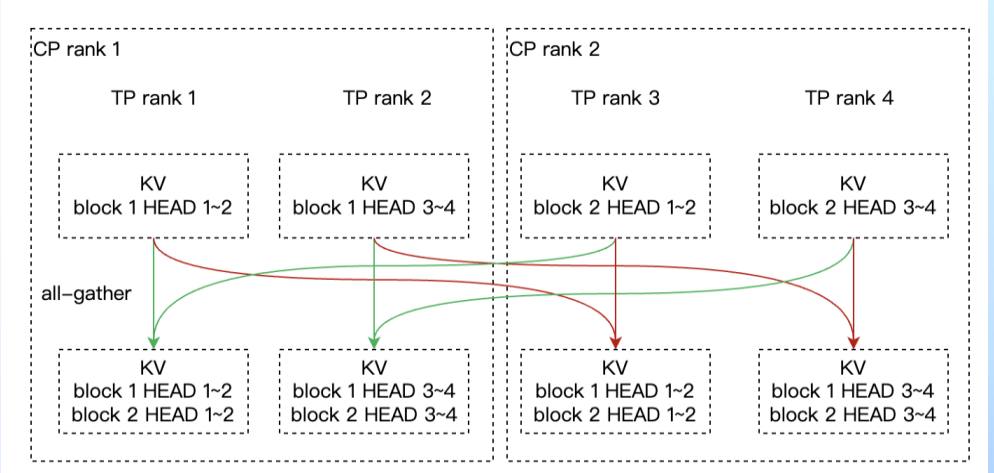
实践2：长输入场景TTFT 优化

CP×TP 优化 TTFT

长输入场景如何进一步降低 TTFT?

常规情况模型 TP 拆分的瓶颈

- TP size 受限于模型 Head 超参，不能无限扩大，无法处理超长序列（QwQ: 40 head；DeepSeek: 128 head）
- GQA/MLA 下，TP 存在重复的 KV cache 计算
- TP>8 时，存在跨机通信，依赖网络带宽



CP2 × TP2 示意

解决方案：CP×TP

- 水平扩展，突破 Head 超参限制
- 减少 KV cache 重复计算
- 减少全局网络通信

	Mean	P50	P90	P99	QPS
TP8 CP2	1.03	0.88	1.98	2.40	0.97
TP16	1.23	1.03	2.35	2.80	0.82

DeepSeek-R1 TTFT（秒）

CP×TP 通信量分析

CP2×TP4 vs TP8: Prefill

- CP 通信量下降 54%
- CP 在长序列场景总耗时更低

维度	TP8	CP2TP4	优势方	说明
单卡通信量	4480	2080	**CP2TP4** ↓54%	单卡通信量减半，直接降低单卡通信延迟
网络总流量	17,920	8320	**CP2TP4** ↓54%	整体网络负载减半，降低网络拥塞风险
通信次数/层	2次	3次	**TP8**	CP 多 1 次通信
通信类型	All-reduce	All-reduce + All-gather	**CP2TP4**	通信量已做了折算

CP2×TP4 vs TP8: Decode

- CP 通信量略微下降

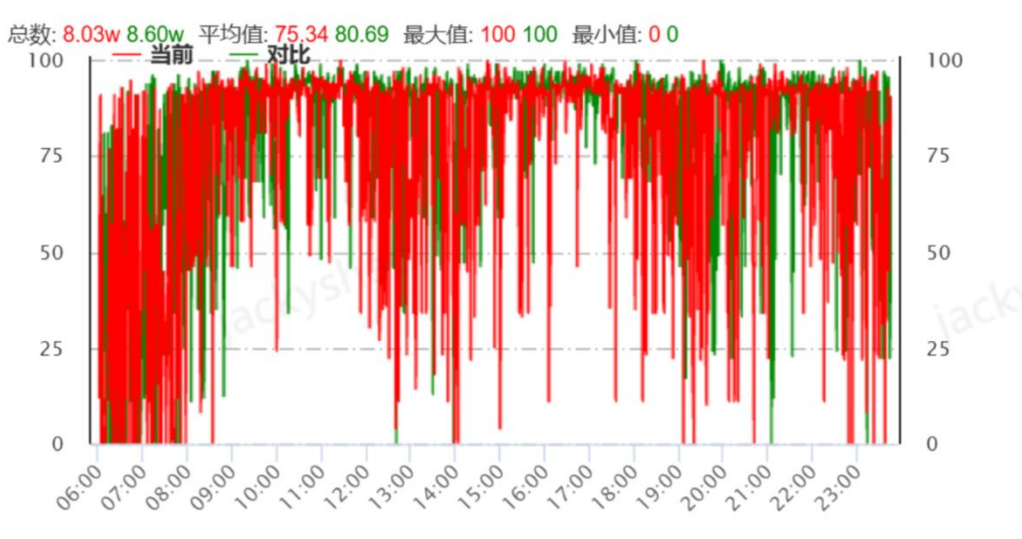
维度	TP8	CP2TP4	优势方	说明
单卡通信量	448	416 ↓7%	**CP2TP4**	单卡通信量轻微下降
网络总流量	1,792	832 ↓54%	**CP2TP4**	整体网络负载减半，降低网络拥塞风险
通信次数/层	2次	3次	**TP8**	CP2TP4 多 64 次启动开销
通信类型	All-reduce	All-reduce + All-gather	**CP2TP4**	通信量已做了折算

实践3：长耗时流量负载均衡

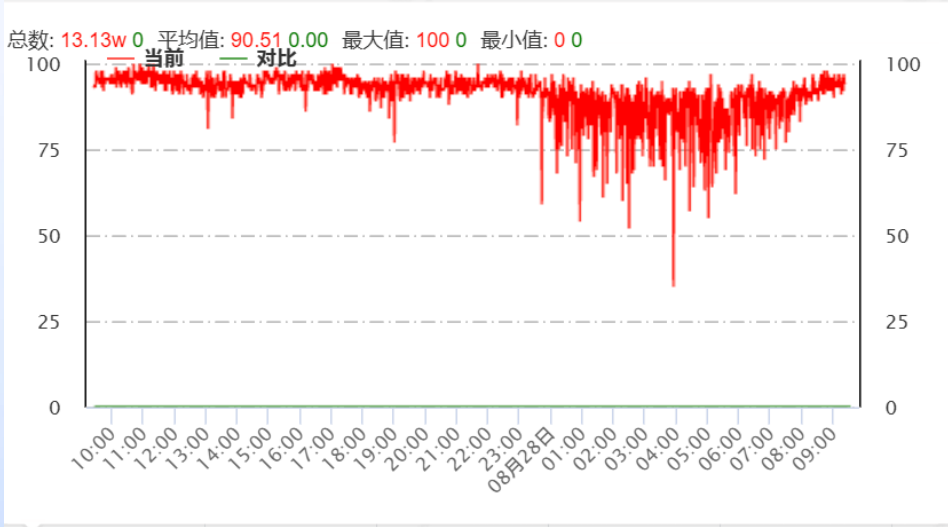
长耗时流量负载均衡

长耗时稀疏流量下如何保证调度均匀?

稀疏流量下的极致调度均衡



优化



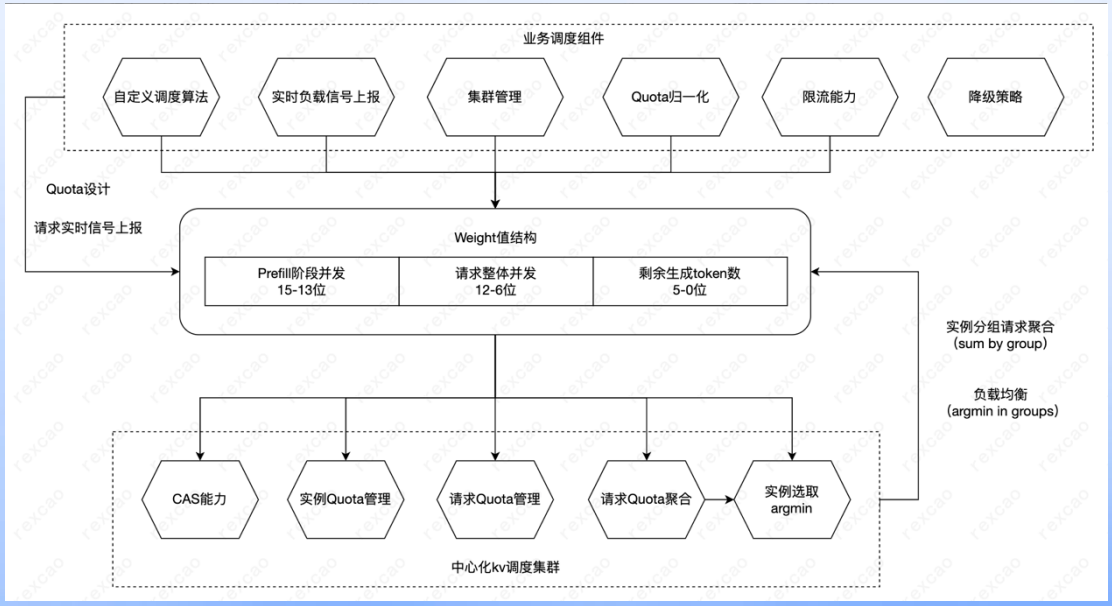
调度优化关键点

- **中心化**调度：集中管理所有节点负载情况并进行最优指派
- **P/D协同**调度：综合考虑P/D负载情况，平衡P/D性能
- **实时感知**调度：实时搜集实例负载，实时决策请求调度

长耗时流量负载均衡

中心化调度层

- **Quota 管理能力**：集群 - 机器 - 请求三级 Quota 管理
- **动态权重择优机制**：支持按机器分组聚合请求维度的 Quota，并以此作为实例权重进行节点择优选取
- **CAS 原子操作**：提供 Compare-And-Swap 机制，确保 Quota 更新和节点择优过程的原子性与一致性



业务逻辑层

- **实时信号上报**：在请求处理的不同阶段（Prefill 开始/结束、Decode 开始/结束）实时上报负载信号
- **Quota 映射设计**：将多维信号归一化为请求维度的 Quota 整数值，并对底层调度系统屏蔽业务细节
- **Quota 位域规划**：
  - 高位 (15-13位)：Prefill 阶段并发度
  - 中位 (12-6位)：请求并发度
  - 低位 (0-5位)：预估剩余生成token数

实践4：PD分离规模化部署

PD分离规模化部署

🧩 架构背景

- **目标**：在高并发、高动态变更场景下，保障**服务连续性**、**容灾隔离性**与**运维可控性**
- **挑战**：流量波动剧烈、模型频繁切换、依赖链复杂，稳定性成为核心考验

🎯 稳定性目标

- **高可用**：多级限流+降级保障主路径不中断
- **高自愈**：节点自愈与自动重拉恢复机制
- **可验证**：演练体系化、问题可量化、机制可复用

风险类型	典型问题	防护思路
容量风险	流量突增、实例宕机、限流配置异常、重试扩散	集群限流 + 快速分流 + 自动重拉机制
代码/下游风险	ttlkv 异常、北极星故障、接口配置不合理	调度降级（随机下发）+ 缓存路由 + 重试治理
变更风险	模型上下线中断、扩缩容失效、配置漂移	优雅退出 + 静态组网 + 灰度变更机制

PD分离规模化部署

业务层

- 集群级限流：QPS 控制 + 随机丢弃 + 快速开关配置
- 快速下线、跨集群重试能力
- 审查 HTTP 超时 & 重试策略

平台层

- 安全扩缩容与静态组网
- 节点自愈 + 屏蔽机制
- 模型灰度变更流程

Conductor / 引擎层

- PD节点限流与调度降级策略：
  - ttlkv异常 → 降级为随机下发
  - 北极星异常 → 启用历史成功IP缓存
- 优雅退出：GPU 空闲后安全下线
- 集群条带化 & QuotaKV 屏蔽机制

容灾策略

- 模型上下线演练：耗时与成功率稳定
- 大版本变更演练：Conductor 与北极星状态一致
- 宕机演练：实例可快速下线
- 降级演练：
  - 请求维度降级
  - 引擎维度降级：全量切换验证
  - 调度降级：随机路由 + 请求丢弃
- 过载演练：系统稳定，限流与成功率符合预期

Q&A

Thanks