SpotServe: Serving Generative Large Language Models on Preemptible Instances

Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia.

Presenter : Yi Liu

Author

Research Direction:

- machine learning systems
- data management
- distributed computing

Selected Publications:

- SpotServe(ASPLOS’2024)
- SpecInfer(ASPLOS’2024)
- Galvatron(VLDB 2023)

Xupeng Miao
https://hsword.github.io/

Outline

- Background
- Design
- Evaluation
- Thinking

Background:The Scale Of LLM

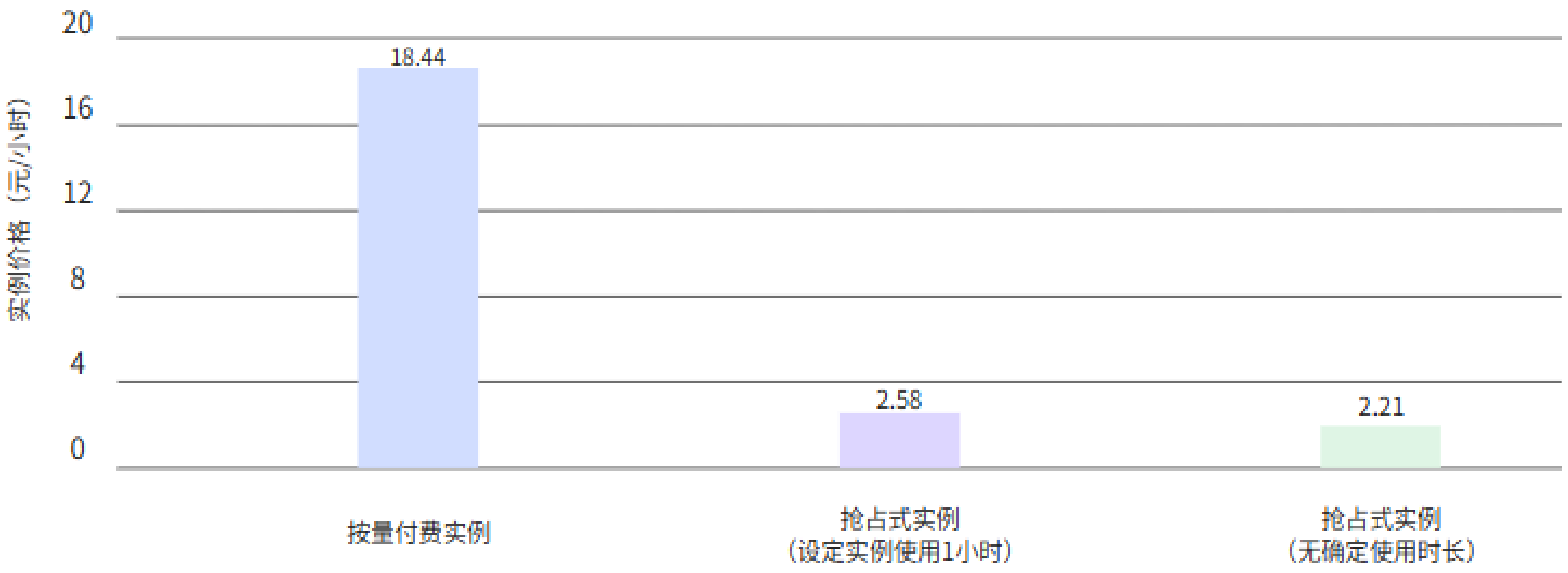
- high computational and memory requirements of LLMs make it challenging to serve them on hardware platforms

model	parameters	device	num	type
GPT-3	175 bilion	Nvidia A100-40GB	16	float

Background : Spot Instances

Modern clouds offer:

- on-demand instances
- spot instances

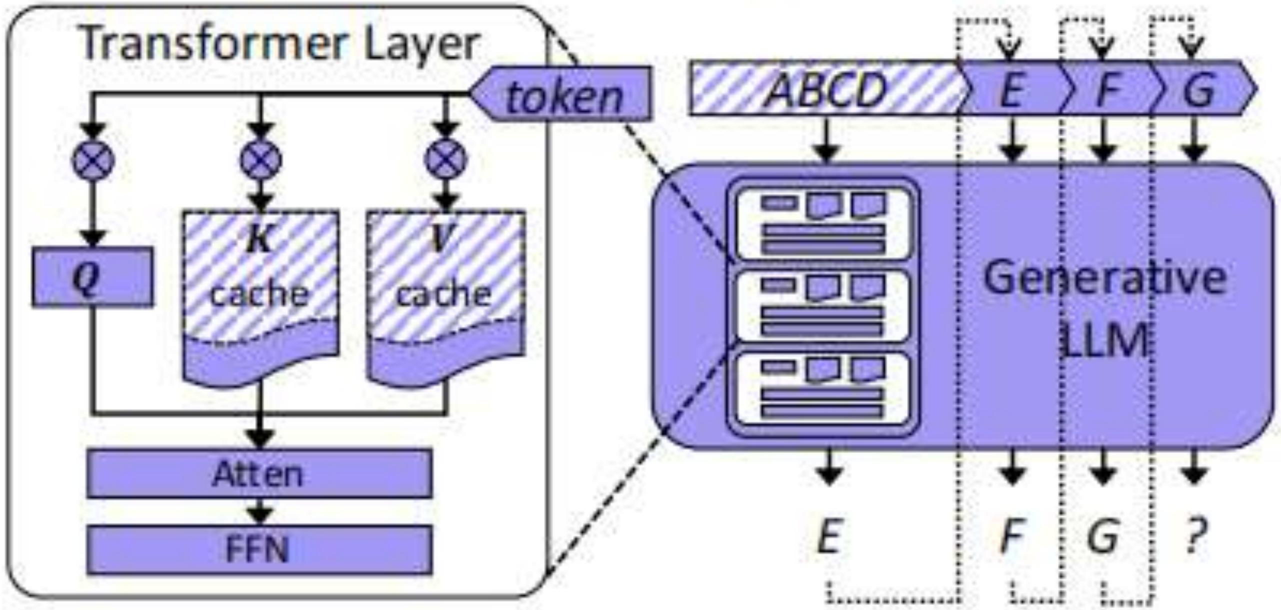


Background:Generative LLM inference

- l_{exe} is execution latency
- t_{exe} is the LLM’ s execution time

Execution Latency:

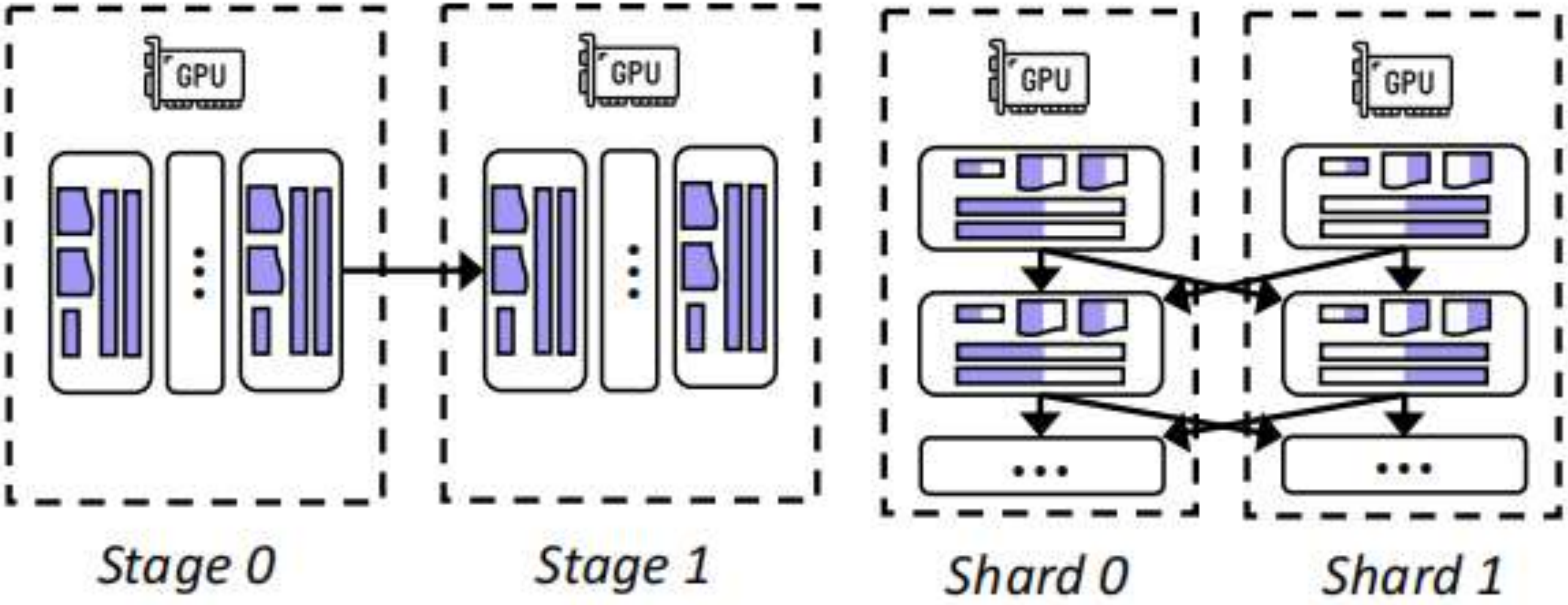
$$l_{exe}(3|4) = t_{exe}(4) + t_{exe}(1) + t_{exe}(1) + t_{exe}(1)$$



$$l_{exe}(S_{out}|S_{in}) = t_{exe}(S_{in}) + \sum_{i=1}^{S_{out}} t_{exe}(S_{in}+i)$$
$$\approx t_{exe}(S_{in}) + S_{out} \times t_{exe}(1)$$

Background:Distributed Infernece

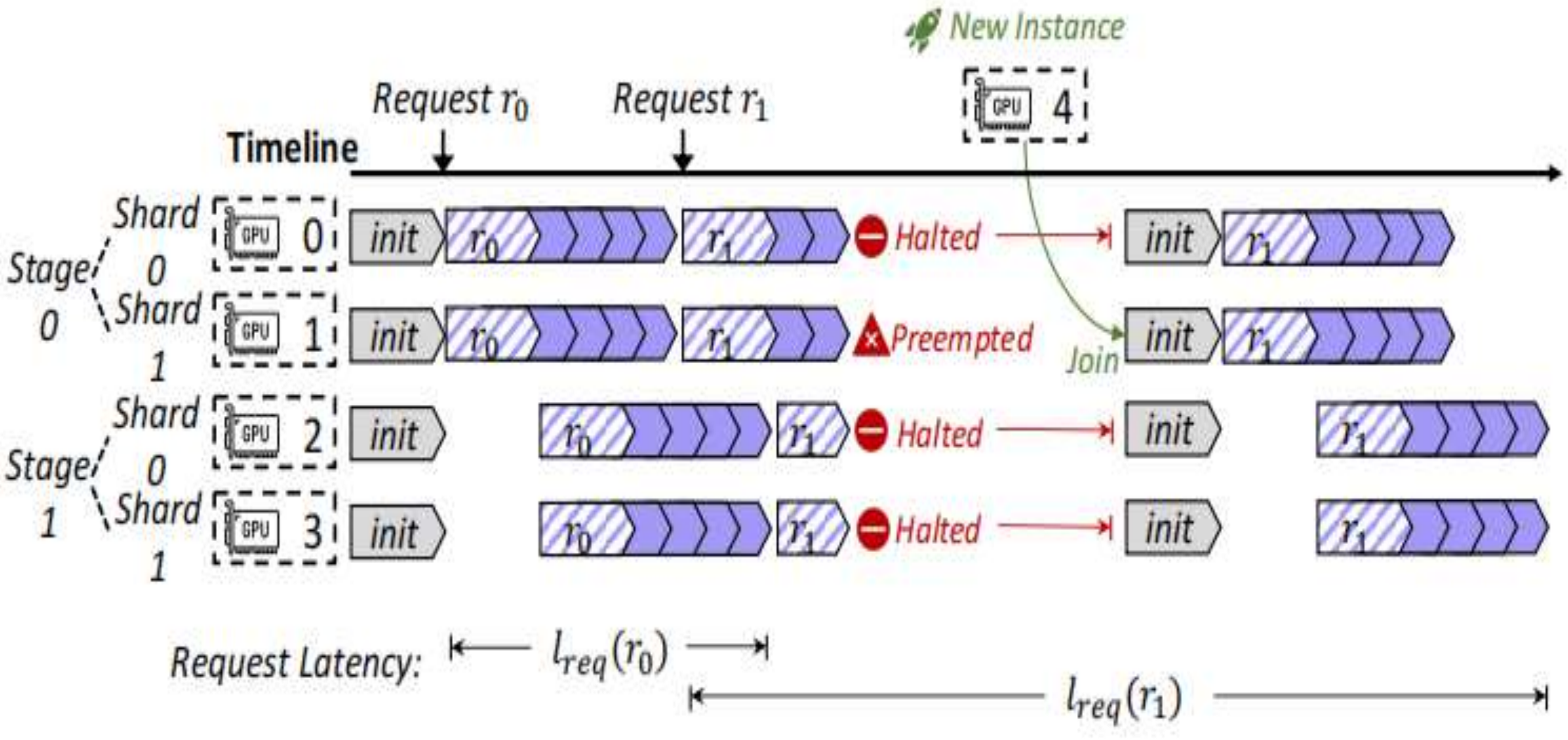
- Inference pipeline combine pipeline model parallelism and tensor model parallelism



(a) Pipeline model parallelism (b) Tensor model parallelism

Background:Preemptible LLM Inference

- One Preempted hang all the other instances
- New instance joins need initialization costs



(b) Distributed LLM inference (P=2, M=2, batch size=1) on preemptible instances

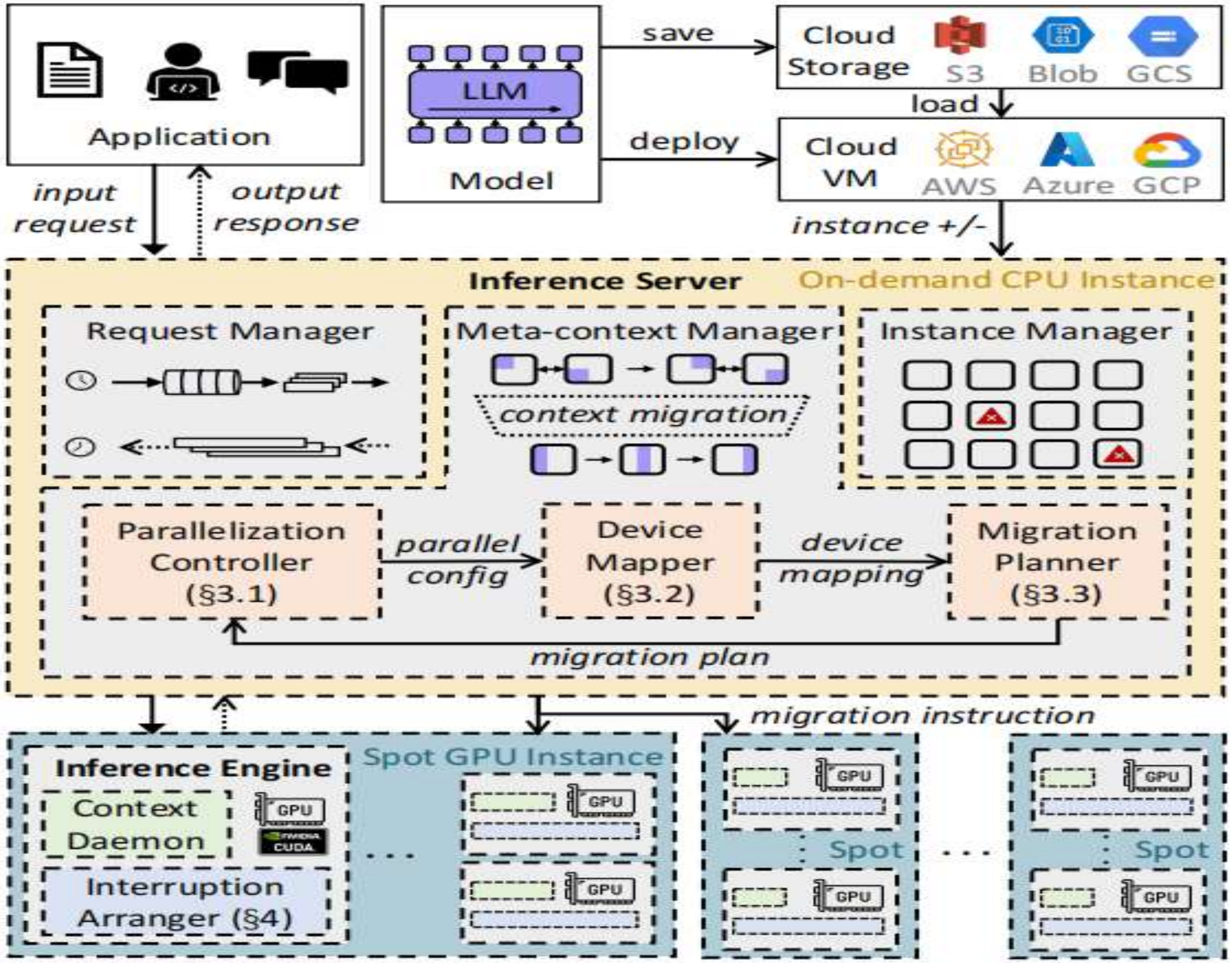
Design:SpotServe

Inference Server:

- Request Manager
- Meta-context Manager
- Instance Manager

Inference Engine :

- Context Daemon
- Interruption Arranger



Design:Parallelization Controller

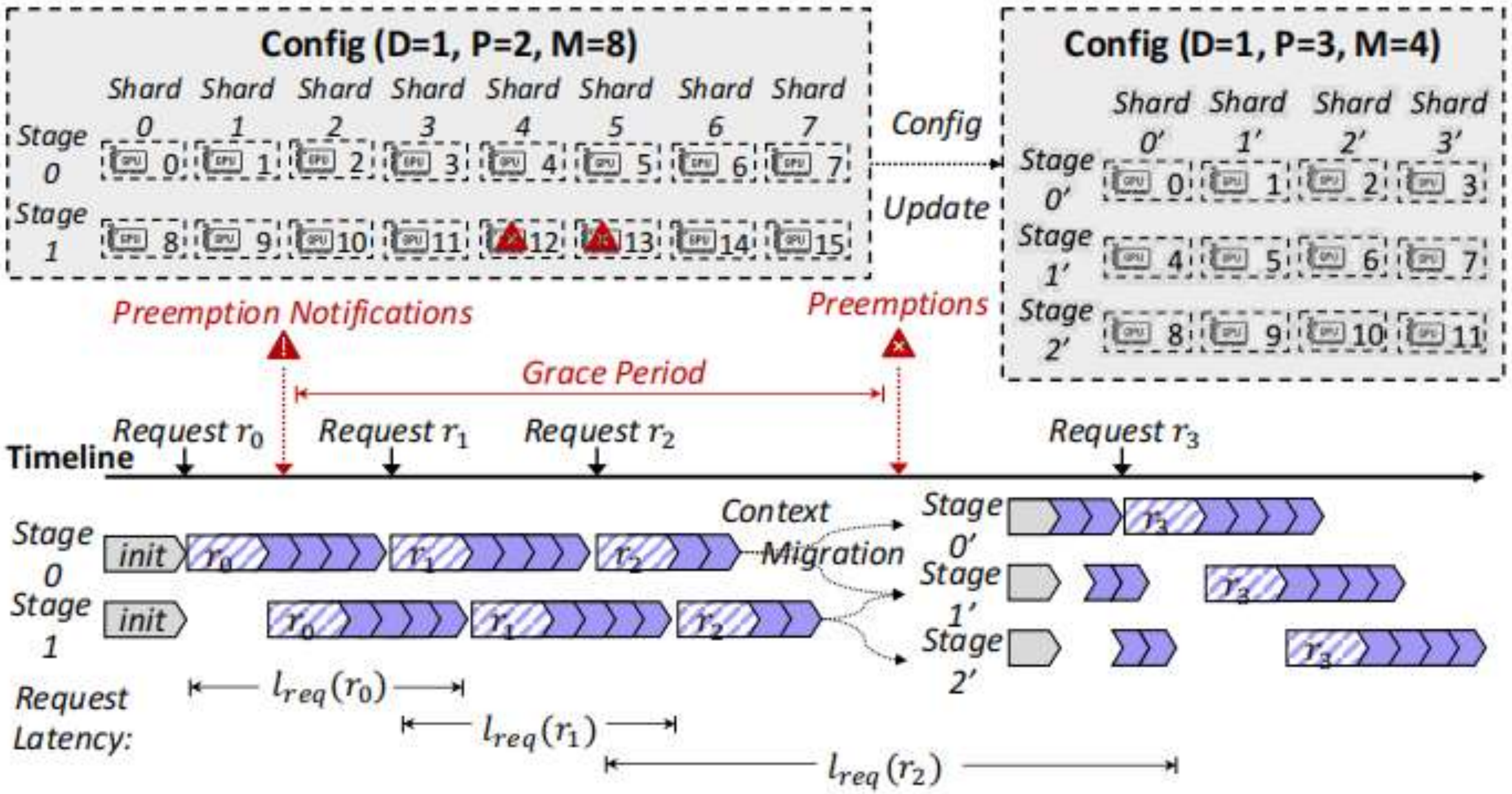
- Parallelization Controller adjusts the parallelization configuration
- Algorithm 1 balance the trade-off among throughput latency, and cost

Algorithm 1 Adaptive configuration optimizer.

```
function ConfigOptimizer(N_t, C_t, α_t)
    if ∃C.ϕ(C) ≥ α_t and cloud has enough instances for C then
        C_{t+1} ← arg min_{C|ϕ(C)≥α_t} l_req(C)
    else
        C_{t+1} ← arg max_{C|N_t} ϕ(C)
    Δ ← #Instances(C_{t+1}) − N_t
    if Δ > 0 then
        InstanceManager.alloc(Δ, ondemand_and_spot)
    else
        InstanceManager.free(−Δ, ondemand_first)
    ConfigUpdate(C_t, C_{t+1})
```

Design:Device Mapper

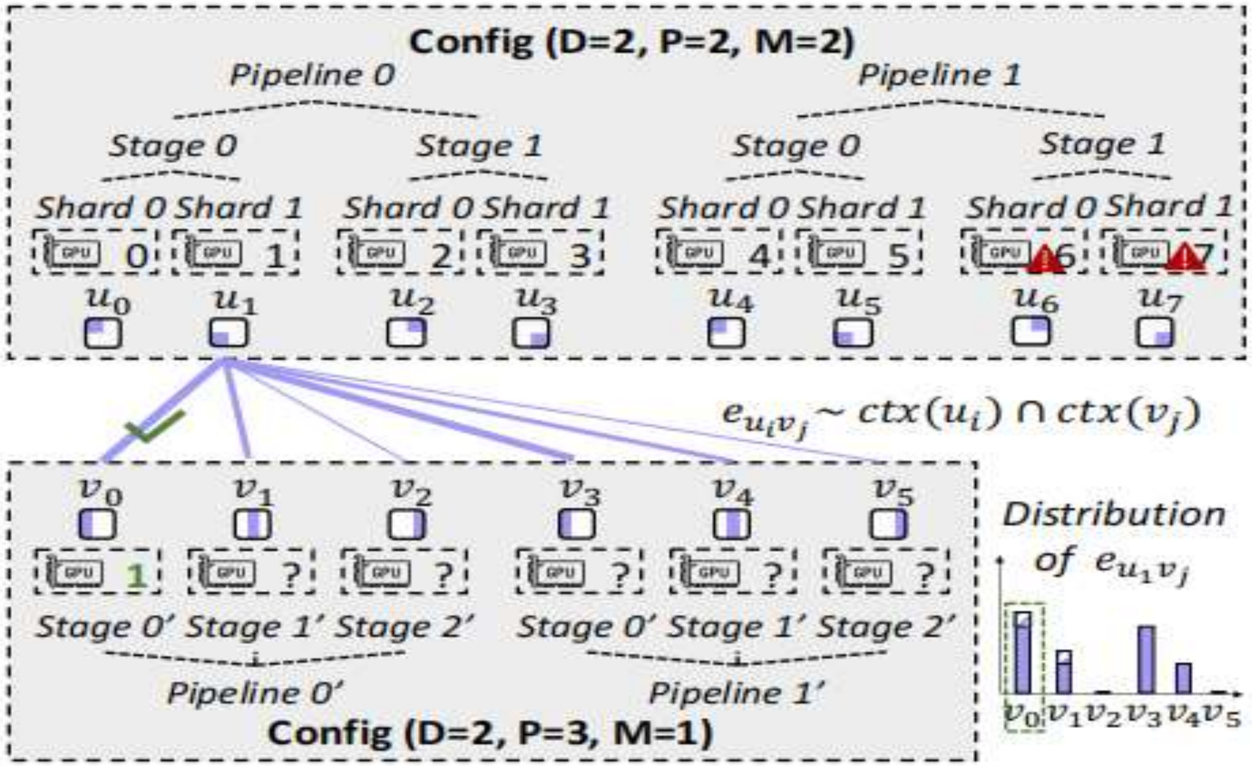
- A context migration mechanism can resume interrupted requests’ inference



(a) Illustration of configuration update and context migration in SpotServe.

Design:Device Mapper

- Device mapping problem → bipartite graph matching task
- Maximally reuses the model parameters and KV cache and minimizes the total data transmission.



(b) Illustration of device mapping in SpotServe.

Design:Migration Planner

- Algorithm2 is a progressive migration schedule
- Consider the memory usage during the progressive migration

Algorithm 2 Workflow of the SpotServe migration planner.

```
▷ Progressive Migration
function MigrationPlanner(context ctx, plan = [ ])
    plan.append(<migrate, ctx.cache>)
    O ← Layer migration order from MemOptMigPlanner
    for layer index i in range(0, #layers) do
        plan.append(<migrate, ctx.weight[O_i]>)
        p ← Get pipeline stage index of layer O_i
        if stage p completes all context migration then
            plan.append(<start, instances of stage p>)
▷ Memory Optimized Migration
function MemOptMigPlanner(maximum buffer size U_max)
    O ← [ ], X ← { }
    Instance buffer memory usage U ∈ {0}^N
    for layer index i in range(0, #layers) do
        if (migrate, ctx.weight[i]) doesn't exceed U_max then
            Update buffer memory usage U
            O.append(i)
        else
            X.add(i)
    while X is not empty do
        x_opt ←
            arg min_{x∈X} max_{0≤i≤N−1} {U_i | (migrate, ctx.weight[x])}
        O.append(r_opt)
        X.remove(x_opt)
```

Design:JIT Arrangement

Utilize grace period

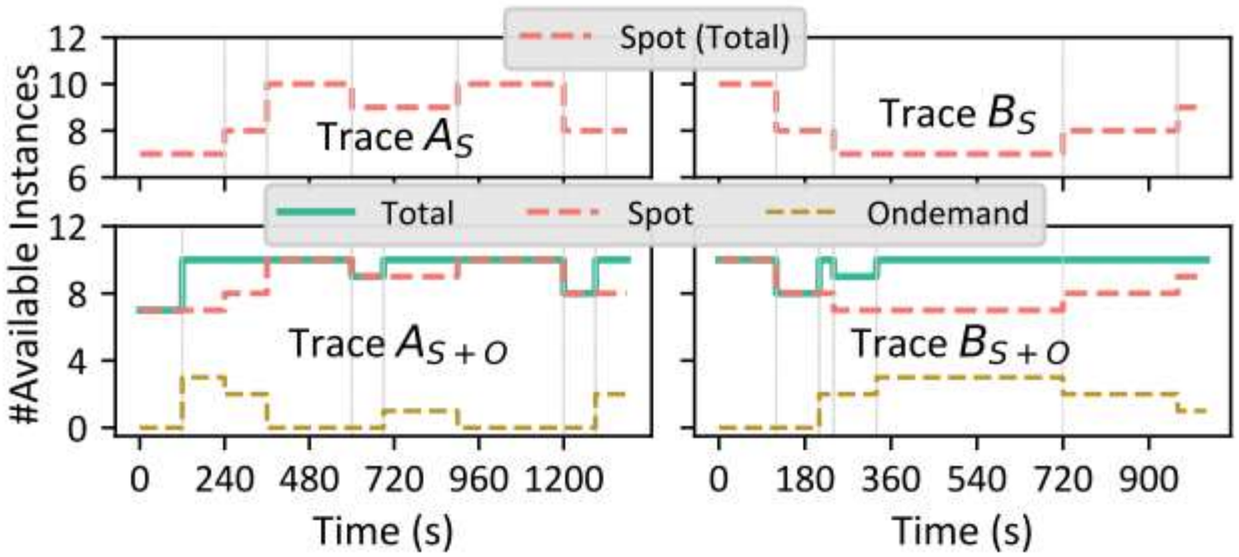
- Each spot GPU instance includes an interruption arranger
- Just-in-time (JIT) arrangement decide when to stop decoding

$$S_t = arg\ max_{0 \le S \le S_{out}} \{l_{exe}(S|C_t) < T^{-} - T_{mig}\}$$
$$\cap\ T_{mig} < l_{exe}(S_t|C_t)$$

Evaluation:Experiment Setup

Baseline:

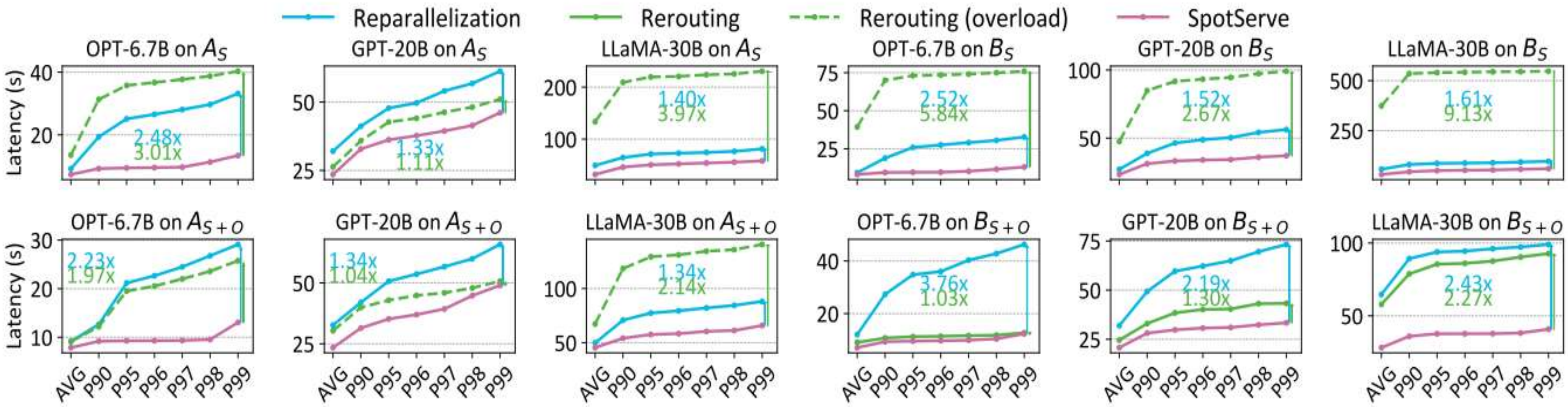
- Request rerouting
- Model reparallelization



Models:

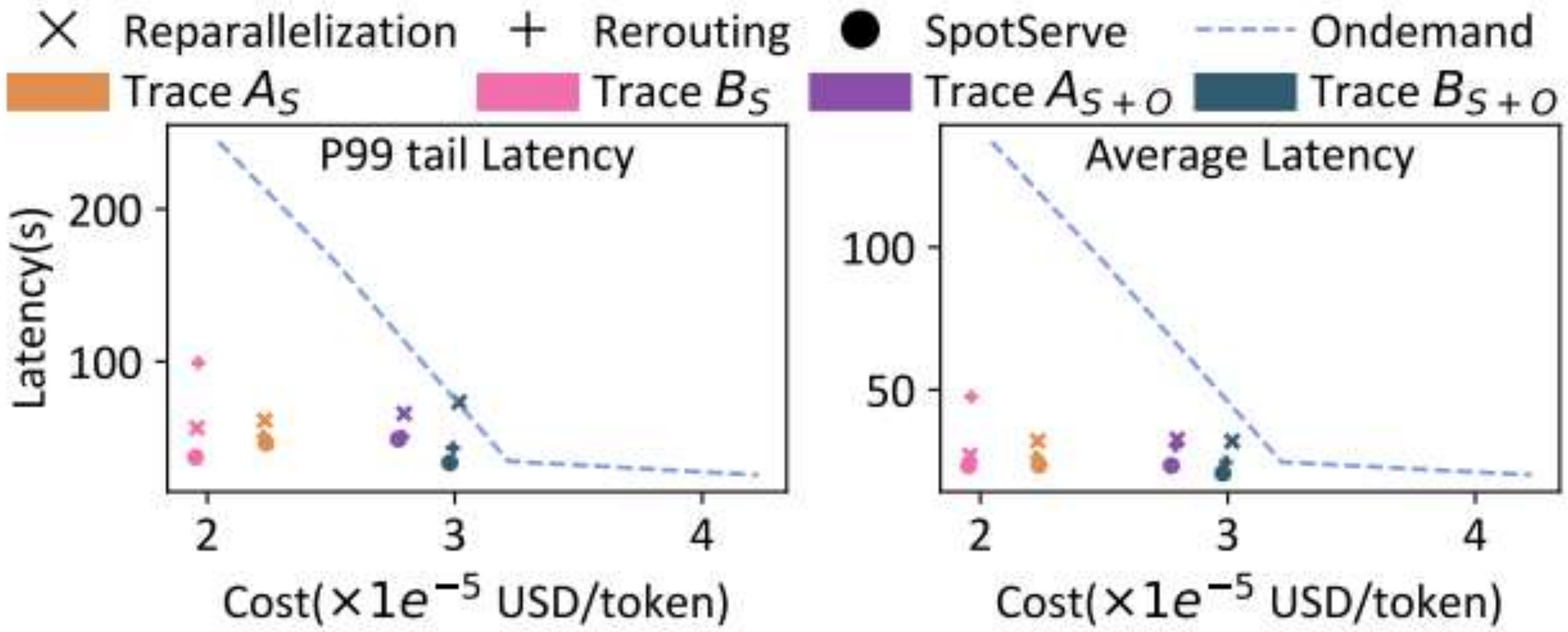
- OPT-6.7B
- GPT-20B
- LLaMA-30B

Evaluation

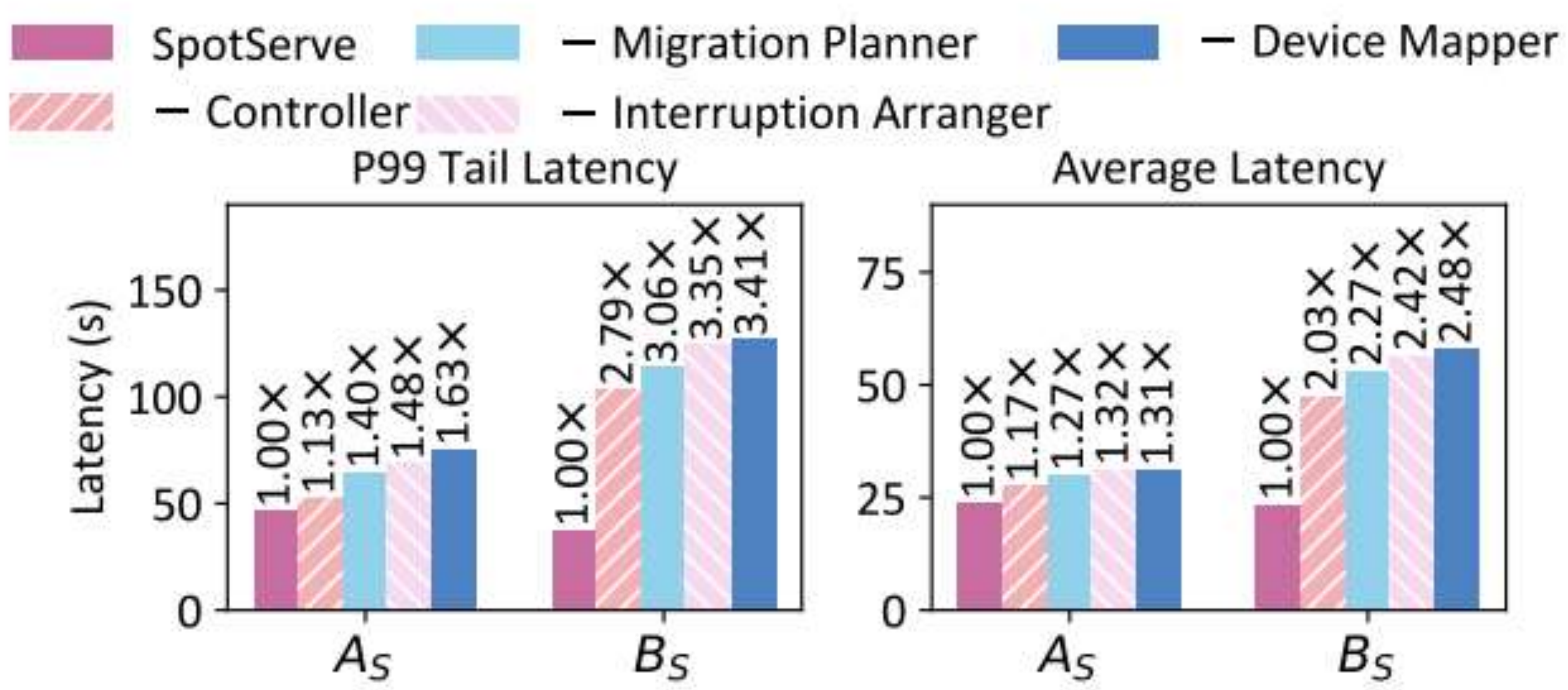


Reparallelization
Rerouting
Rerouting (overload)
SpotServe
OPT-6.7B on A_S
GPT-20B on A_S
LLaMA-30B on A_S
OPT-6.7B on B_S
GPT-20B on B_S
LLaMA-30B on B_S
OPT-6.7B on A_{S+O}
GPT-20B on A_{S+O}
LLaMA-30B on A_{S+O}
OPT-6.7B on B_{S+O}
GPT-20B on B_{S+O}
LLaMA-30B on B_{S+O}
Latency (s)
2.48x
3.01x
1.33x
1.11x
1.40x
3.97x
2.52x
5.84x
1.52x
2.67x
1.61x
9.13x
2.23x
1.97x
1.34x
1.04x
1.34x
2.14x
3.76x
1.03x
2.19x
1.30x
2.43x
2.27x
AVG
P90
P95
P96
P97
P98
P99

Evaluation


Reparallelization
Rerouting
SpotServe
Ondemand
Trace A_S
Trace B_S
Trace A_{S+O}
Trace B_{S+O}
P99 tail Latency
Average Latency
Latency(s)
200
100
100
50
2
3
4
Cost($\times 1e^{-5}$ USD/token)
Cost($\times 1e^{-5}$ USD/token)

Evaluation



SpotServe
− Migration Planner
− Device Mapper
− Controller
− Interruption Arranger
P99 Tail Latency
Average Latency
Latency (s)
150
100
50
0
75
25
1.00×
1.13×
1.40×
1.48×
1.63×
2.79×
3.06×
3.35×
3.41×
1.17×
1.27×
1.32×
1.31×
2.03×
2.27×
2.42×
2.48×
A_S
B_S

Thinking

- 文章有什么问题

1. 该系统的并行策略中考虑的都是相同类型的GPU实例，并行策略上也是将其按照同构资源处理
2. 在模型映射部分，只将模型参数的可复用程度作为边权的考虑，没有考虑设备间数据传输的能力差异

- 该方法能不能应用到别的场景

　对于智能家居的分布式计算来说，对于一个有处理任务的设备来说，其他当前状态下空闲的设备的资源对它来说视作可抢占资源，利用目前可用资源配置出最优的并行推理流水线

Thanks