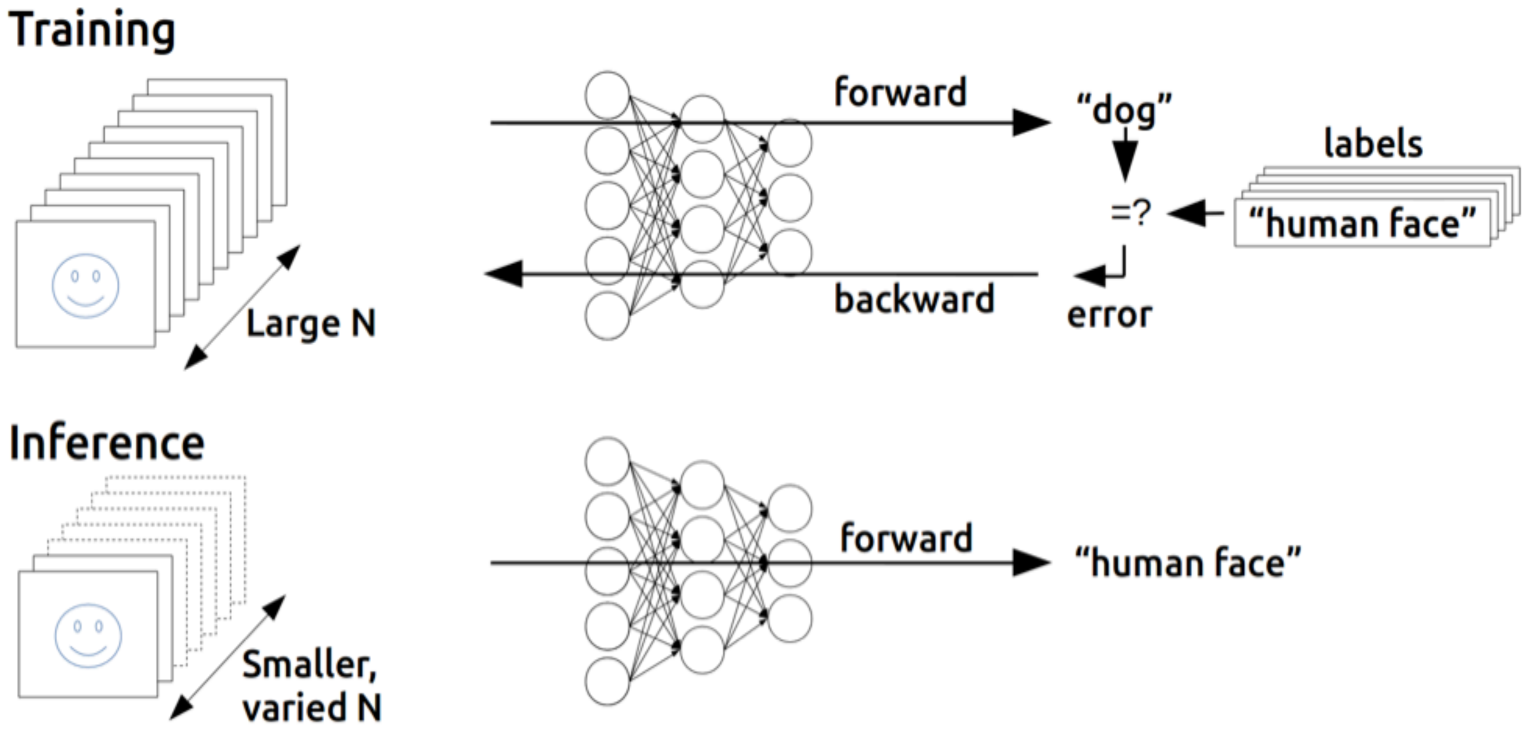
Lyra: Elastic Cluster Scheduling for Deep Learning

Jiamin Li[1], Hong Xu[2], Yibo Zhu[3], Zherui Liu[4], Chuanxiong Guo[5], Cong Wang[1]

[1]City University of Hong Kong, [2]The Chinese University of Hong Kong, [3]Google, [4]ByteDance Inc., [5]Unaffiliated

EuroSys 2023

Large-scale GPU Clusters for DNN jobs



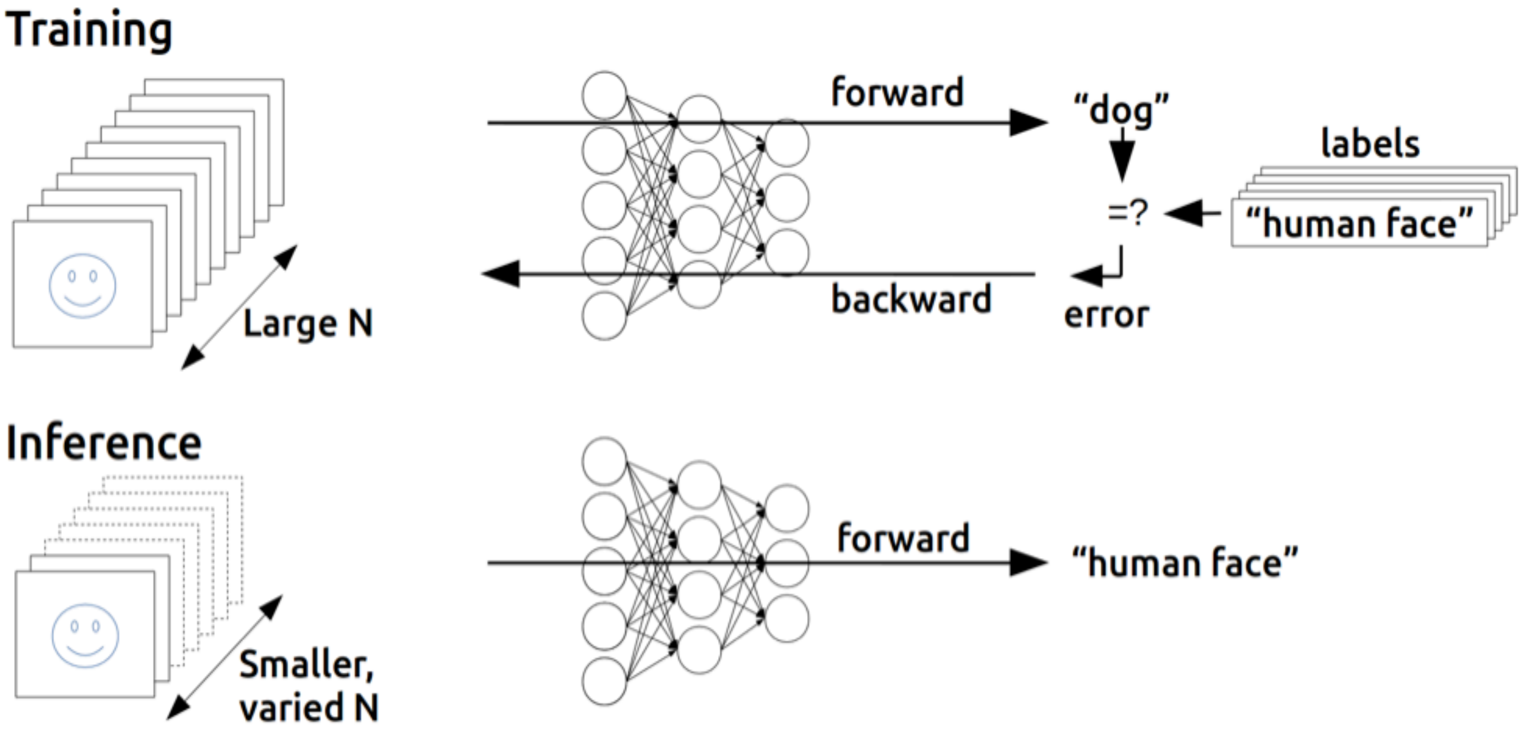
Resource-heavy
V100, A100

Latency-sensitive

T4, P4

* Figure credited to https://developer.nvidia.com/blog/inference-next-step-gpu-accelerated-deep-learning/

Large-scale GPU Clusters for DNN jobs



Resource-heavy
V100, A100

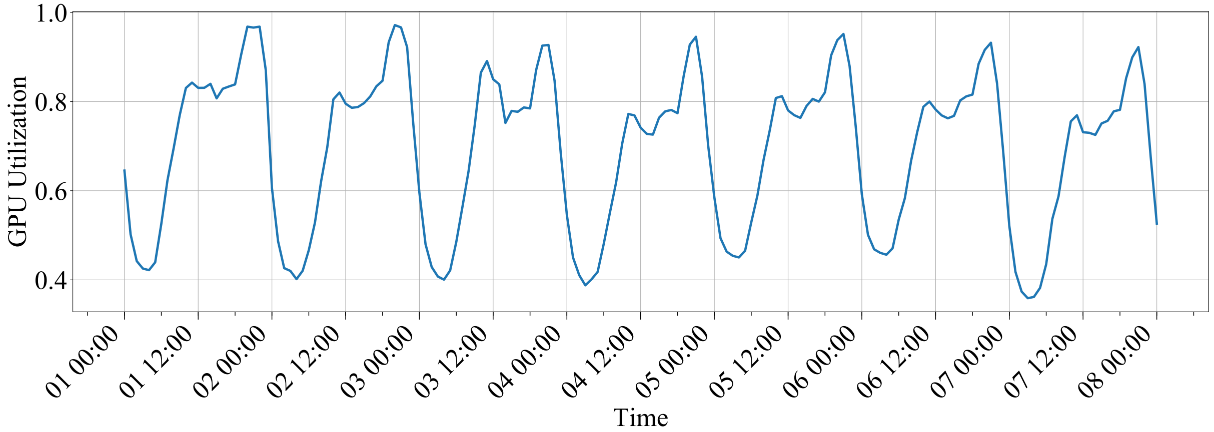
Latency-sensitive
T4, P4

Separate deployment of GPU clusters for training and inference.

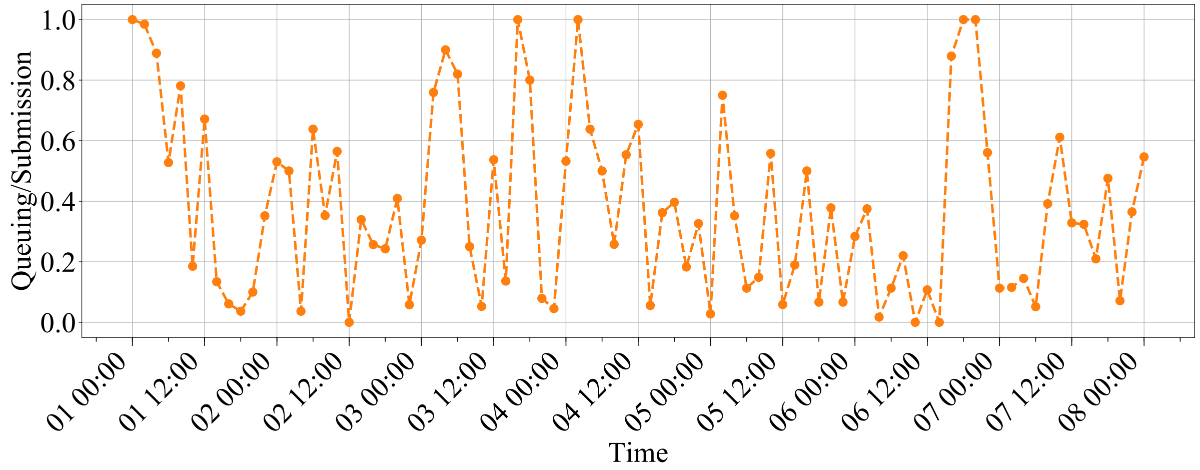
* Figure credited to https://developer.nvidia.com/blog/inference-next-step-gpu-accelerated-deep-learning/

Observations from the production environment

- Inference cluster: T4



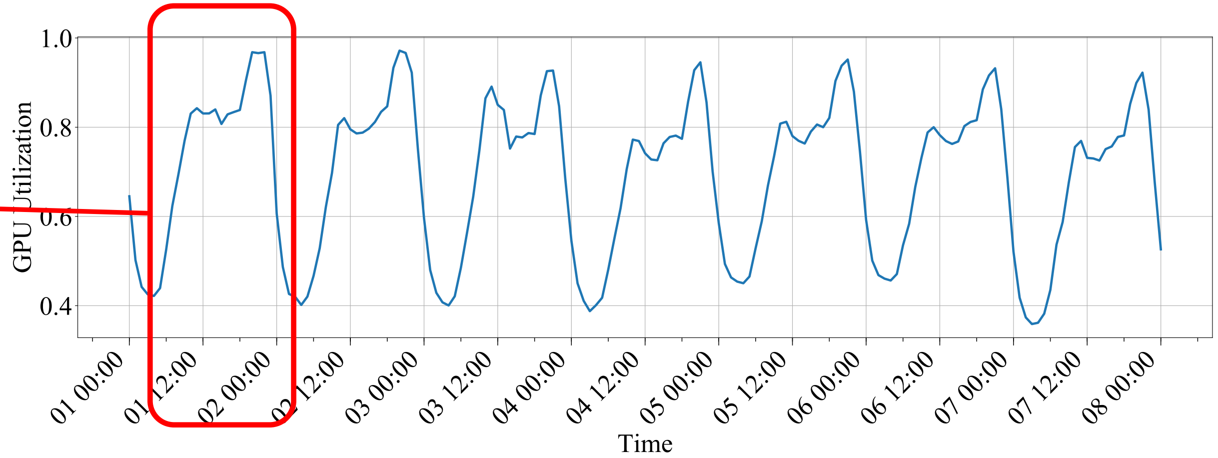
- Training cluster: V100



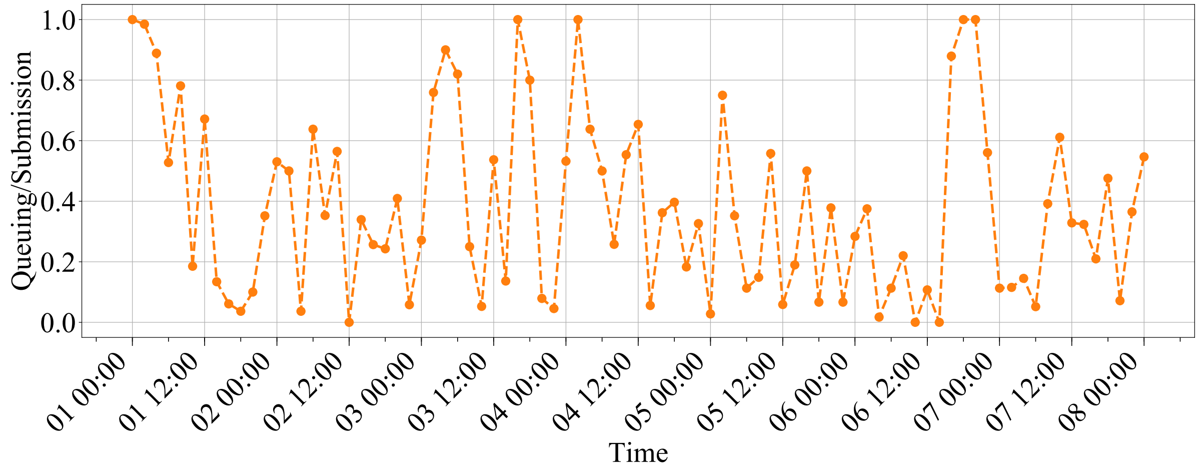
Observations from the production environment

- Inference cluster: T4

  Diurnal Patterns



- Training cluster: V100

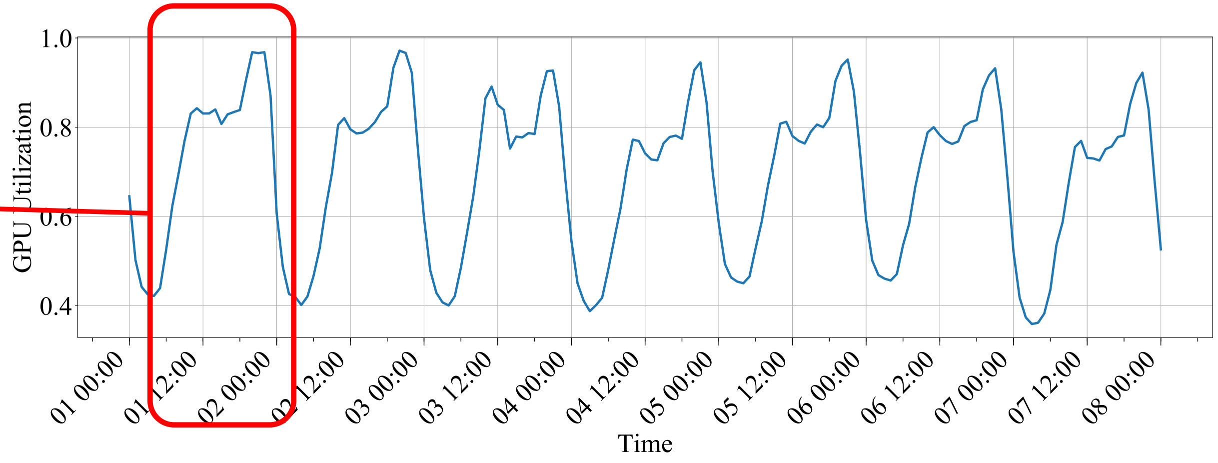


Observations from the production environment

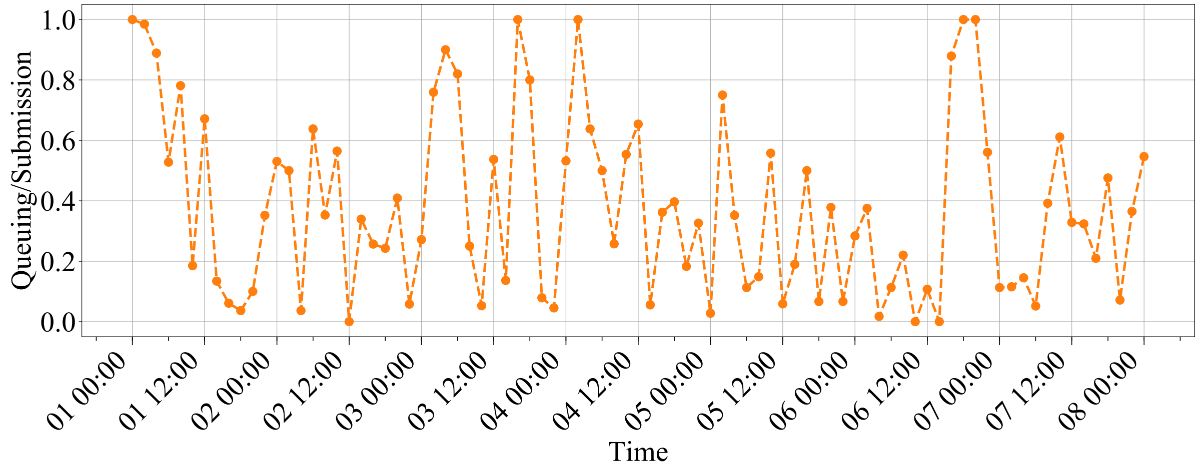
- Inference cluster: T4

  Diurnal Patterns

  Avg. Utilisation ~40%



- Training cluster: V100

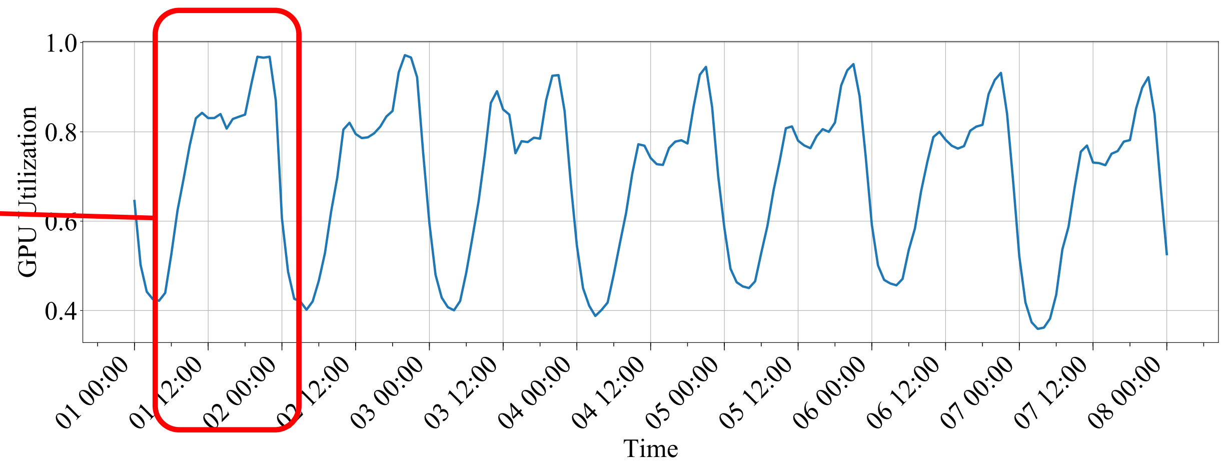


Observations from the production environment

- Inference cluster: T4

  Diurnal Patterns

  Avg. Utilisation ~40%

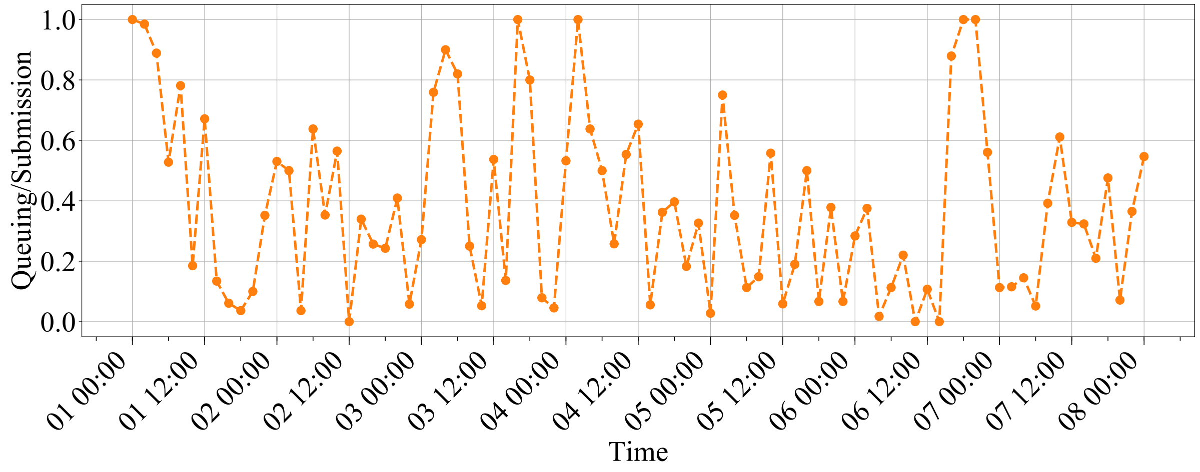


- Training cluster: V100

  Long queuing time

  Avg. 3000 s

  95%tile 10,000s

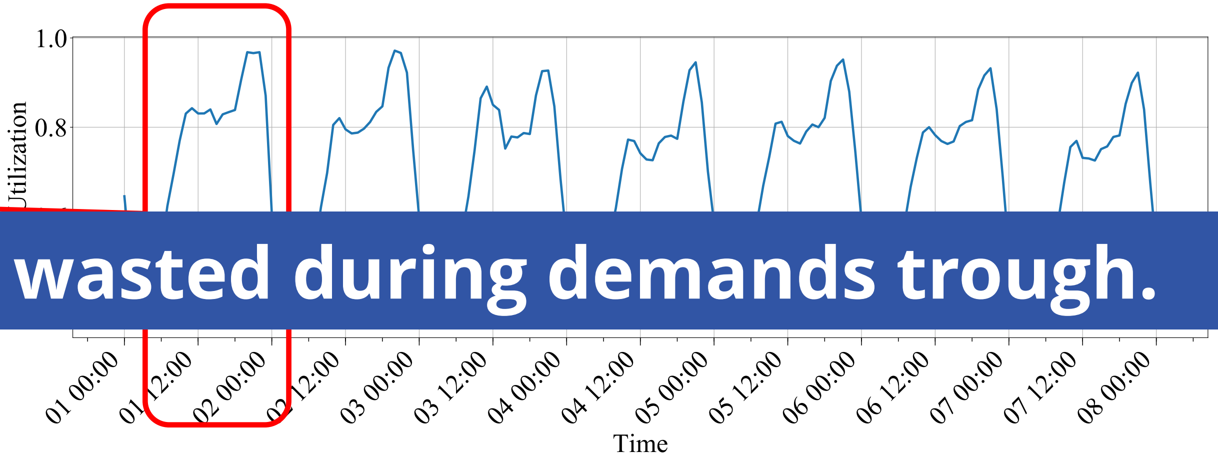


Observations from the production environment

- Inference cluster: T4

  Diurnal Patterns

  Av



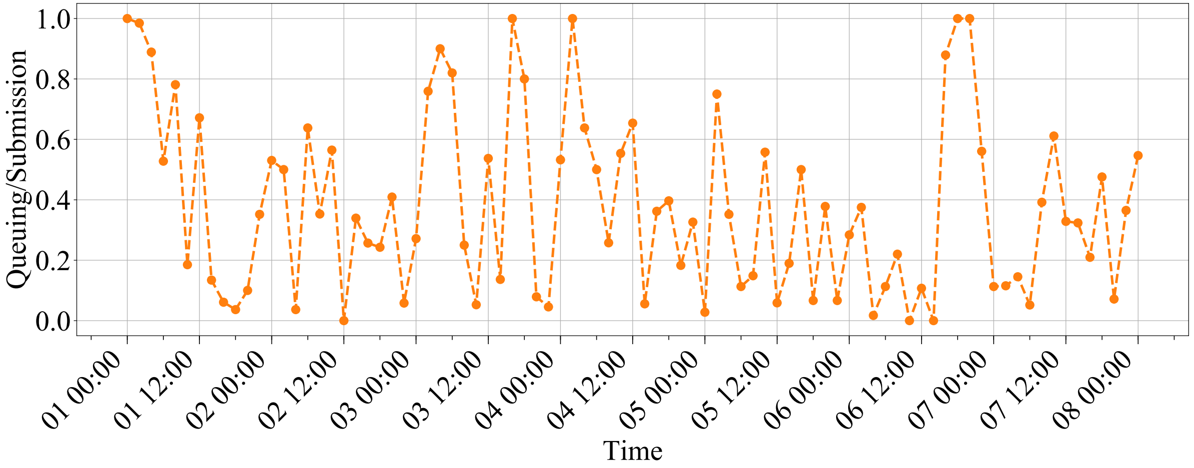
Abundant resources are wasted during demands trough.

- Training cluster: V100

  Long queuing time

  Avg. 3000 s

  95%tile 10,000s

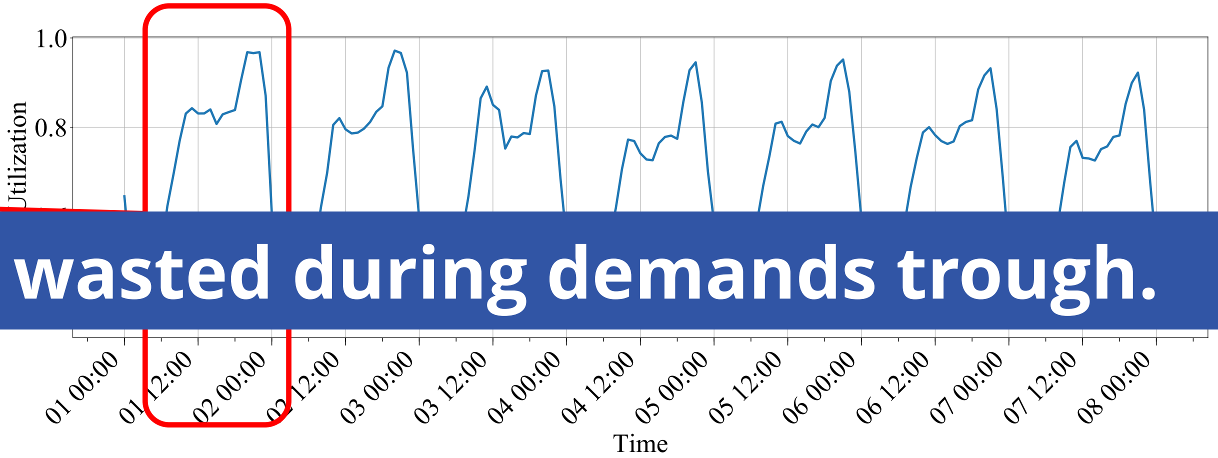


Observations from the production environment

- Inference cluster: T4

  Diurnal Patterns

  A



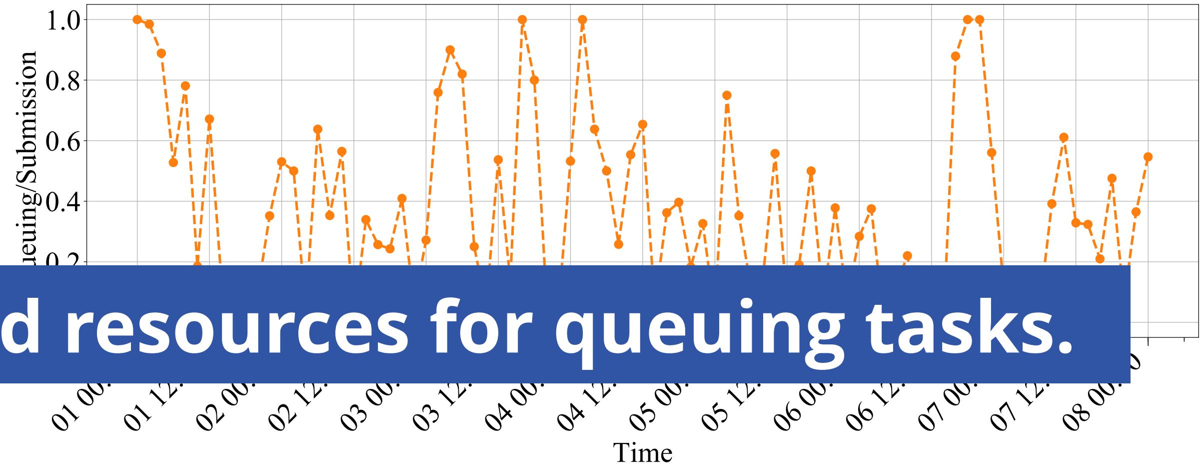
Abundant resources are wasted during demands trough.

- Training cluster: V100

  Long queuing time

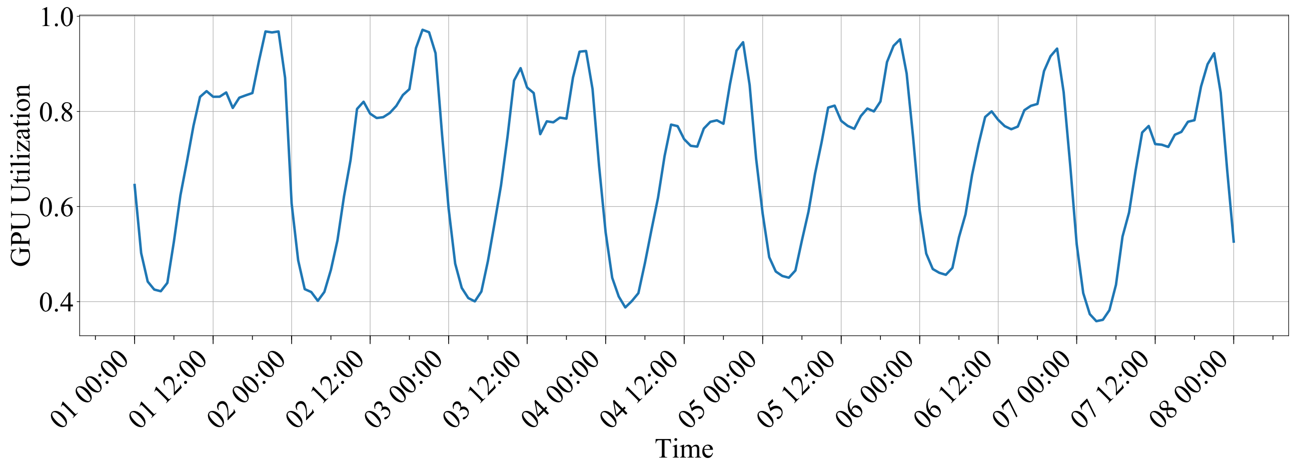
  Avg. 3000 s

  95%ti



Lack of unfragmented resources for queuing tasks.

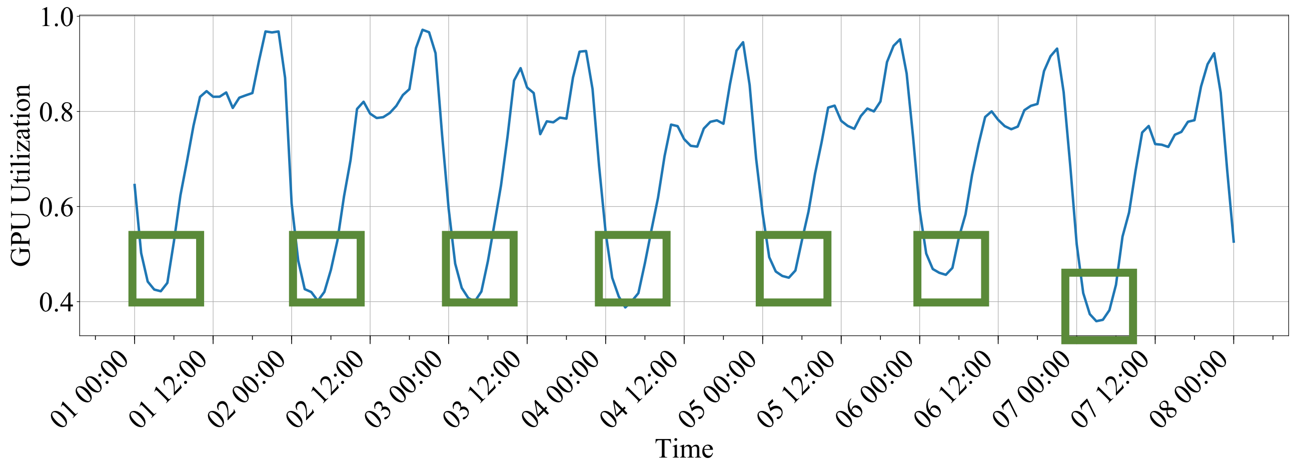
Capacity Loaning from Inference Cluster



How to make the best use of the servers loaned from the inference cluster?

- Opportunities
  - [O1] Fungible and heterogeneous workloads can utilise inference resources.
  - [O2] Elastic scaling workload match with the dynamically changing resources pool and reduce job preemption.

Capacity Loaning from Inference Cluster

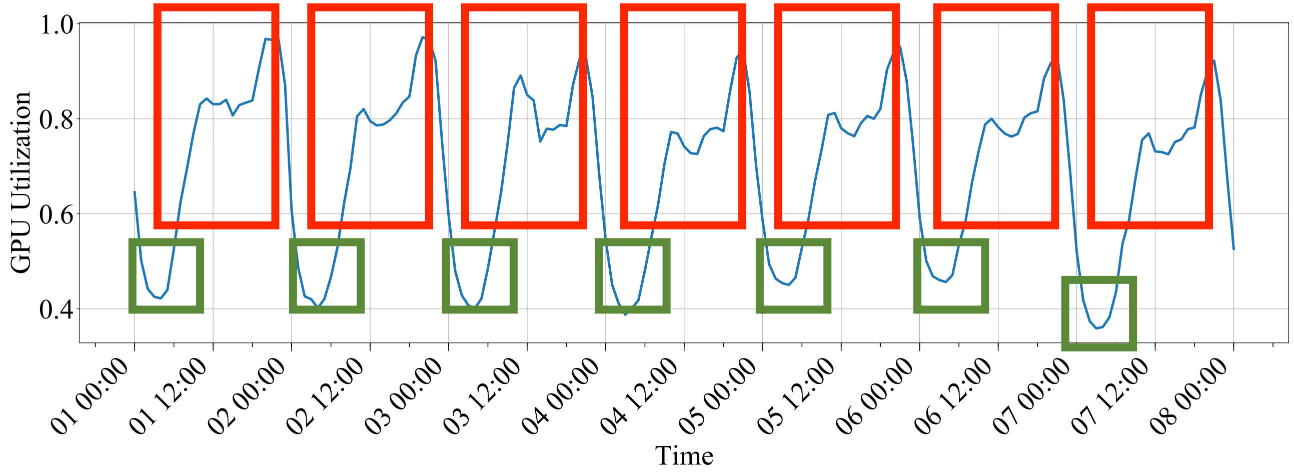


Low Traffic: loan resources to training jobs.

How to make the best use of the servers loaned from the inference cluster?

- Opportunities
  - [O1] Fungible and heterogeneous workloads can utilise inference resources.
  - [O2] Elastic scaling workload match with the dynamically changing resources pool and reduce job preemption.

Capacity Loaning from Inference Cluster



Traffic Spikes: reclaim resources.

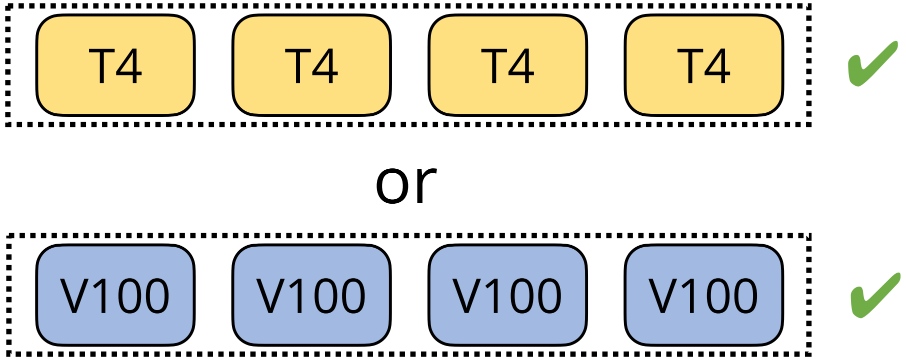
Low Traffic: loan resources to training jobs.

How to make the best use of the servers loaned from the inference cluster?

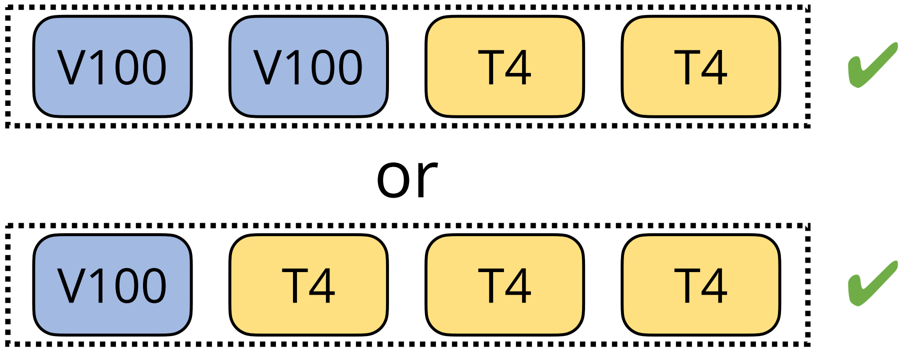
- Opportunities
  - [O1] Fungible and heterogeneous workloads can utilise inference resources.
  - [O2] Elastic scaling workload match with the dynamically changing resources pool and reduce job preemption.

Opportunity 1: Fungible and Heterogeneous Workloads

- 21% of the workload are <u>fungible</u> jobs:
  - Work with any GPU types in **different** execution runs.
  - GPU memory differences can be handled with batch size adjustment.

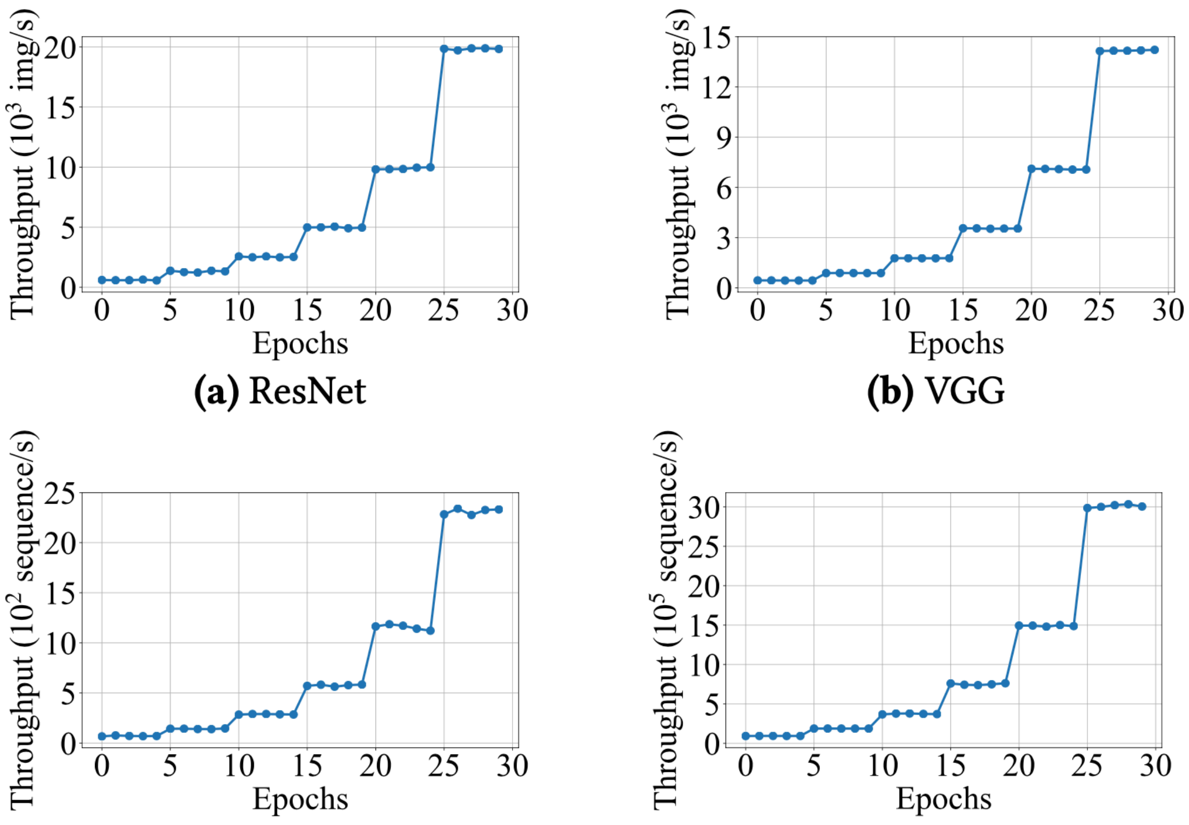


- 5% of the workload are <u>heterogeneous</u> tasks:
  - Use heterogeneous GPUs in one **single** execution run.
  - Extensive work support efficient training for the suitable models



Opportunity 2: Elastic Scaling Workload

- Elastic DNN training
  - Varied number of workers
  - PyTorch Elastic, Elastic DL…
- Linear scalability within a *limited* scaling range
  - Model families: ResNet, BERT…
  - ~5% of training jobs
  - **~36%** of training cluster resources



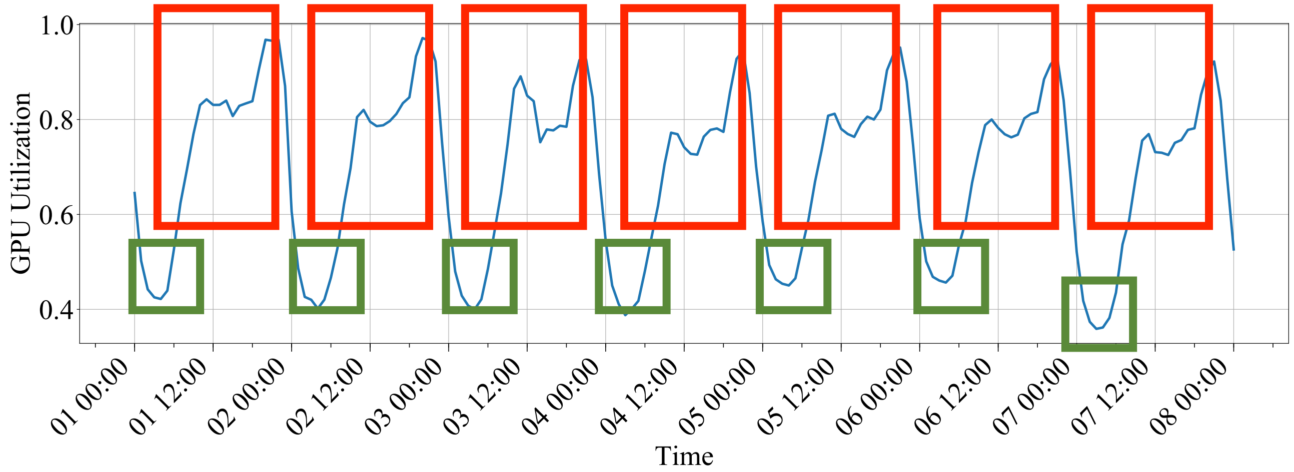
(a) ResNet **(b)** VGG **(c)** BERT **(d)** GNMT-16

Opportunity 2: Benefit of Elastic Training Workload

Traditional Training	Elastic Training
Gang scheduling	Launch with a smaller number of workers
Fixed resource allocation	Dynamically adjust resource allocation on-the-fly
Fixed running time	Scale out to reduce running time with low overhead

Capacity Loaning from Inference Cluster

- Opportunities
  - [O1] Fungible and heterogeneous workloads can utilise inference resources.
  - [O2] Elastic scaling workload match with the dynamically changing resources pool and reduce job preemption.

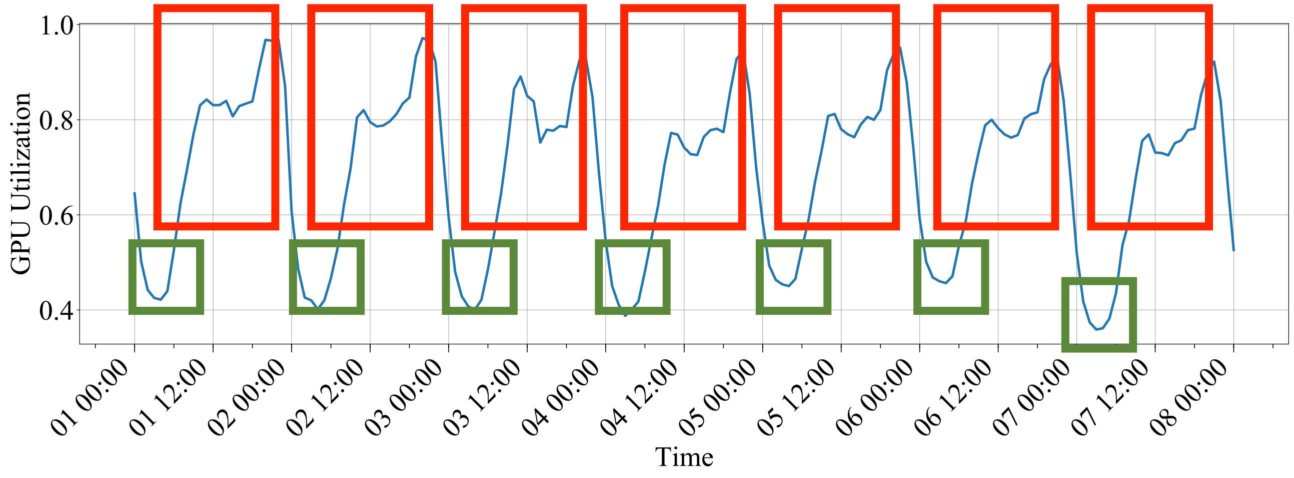


Traffic Spikes: reclaim resources.

Low Traffic: loan resources to training jobs.

Capacity Loaning from Inference Cluster

- Opportunities
  - [O1] Fungible and heterogeneous workloads can utilise inference resources.
  - [O2] Elastic scaling workload match with the dynamically changing resources pool and reduce job preemption.



[O2] Scale in elastic workload to vacate resources.

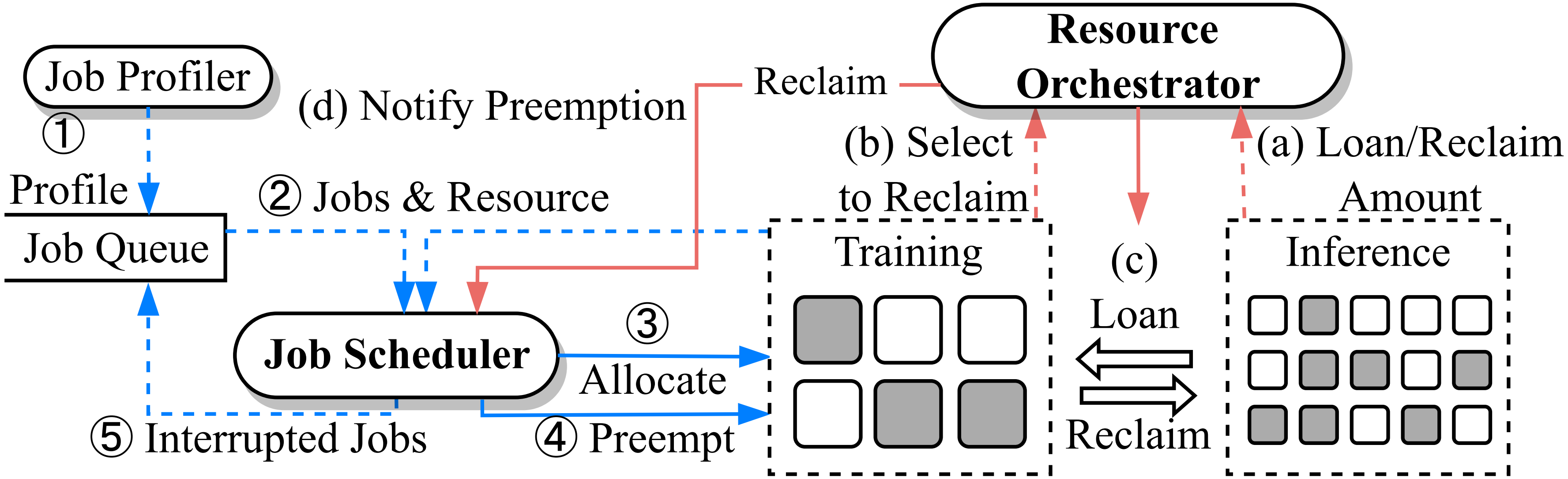
Traffic Spikes: reclaim resources.

Low Traffic: loan resources to training jobs.

[O1, O2] Schedule jobs on inference servers, scale out elastic workload

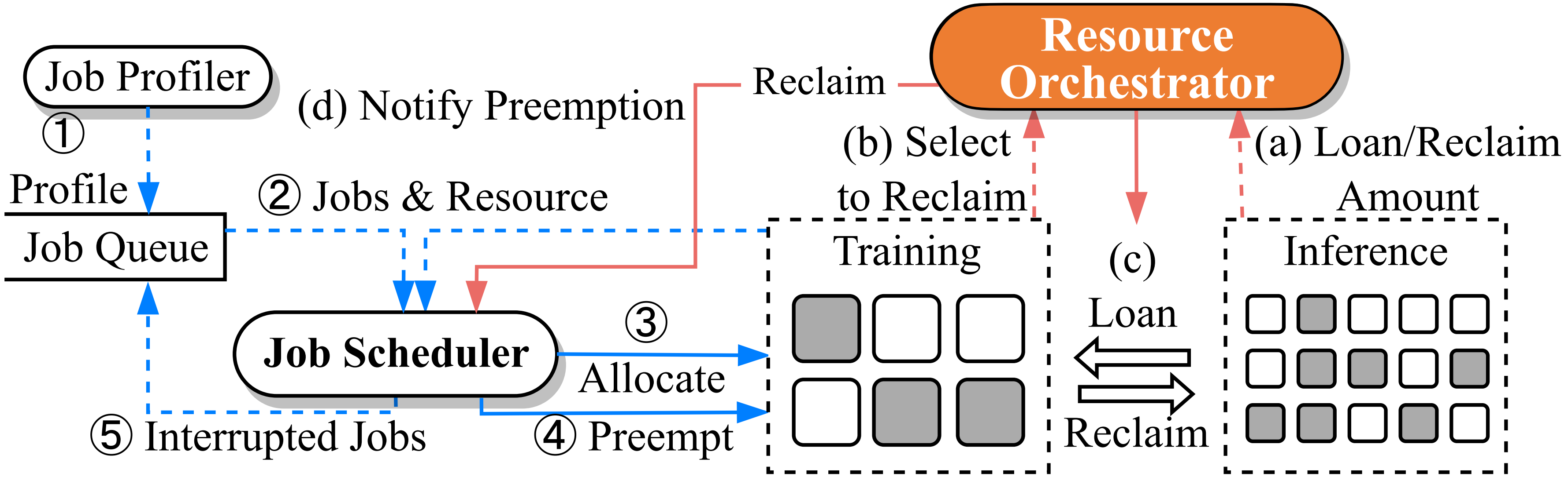
Lyra: System Architecture

- Lyra: an elastic cluster scheduler
  - Cluster-level elasticity handled by Resource Orchestrator
  - Job-level elasticity handled by Job Scheduler



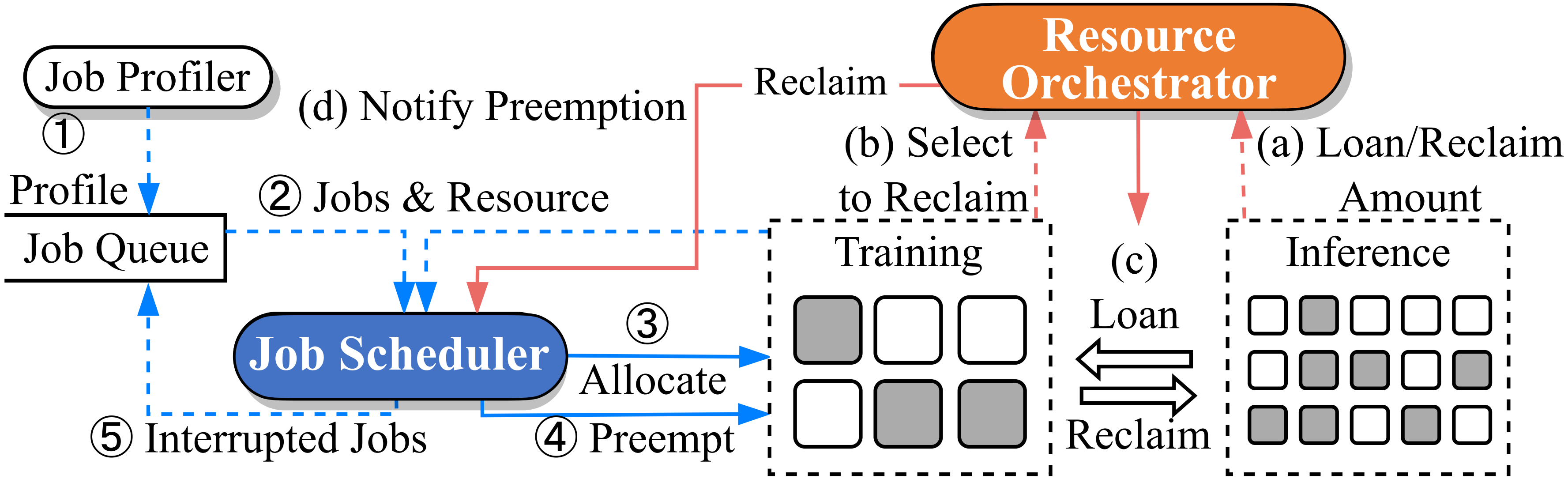
Lyra: System Architecture

- Lyra: an elastic cluster scheduler
  - Cluster-level elasticity handled by Resource Orchestrator
  - Job-level elasticity handled by Job Scheduler



Lyra: System Architecture

- Lyra: an elastic cluster scheduler
  - Cluster-level elasticity handled by Resource Orchestrator
  - Job-level elasticity handled by Job Scheduler

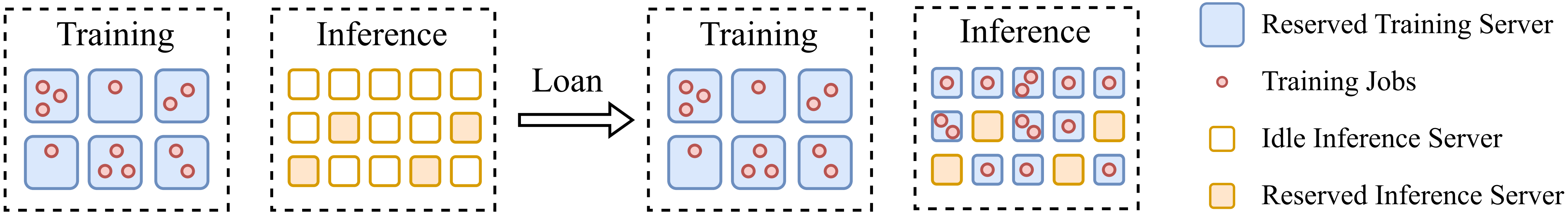


Key Questions

- Capacity Loaning: improve cluster efficiency

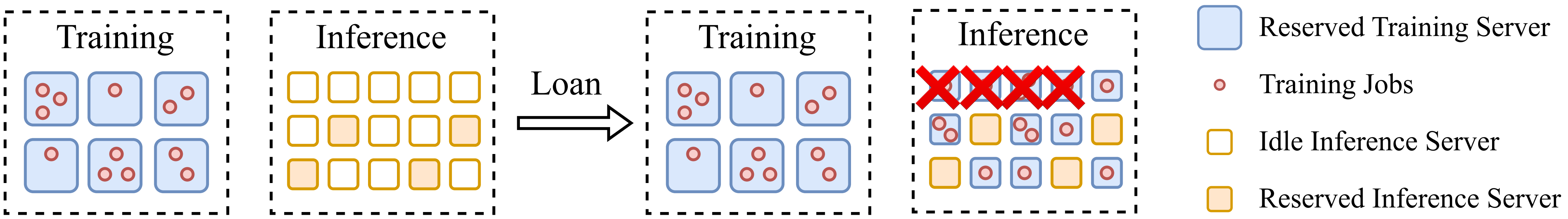
Key Questions

- Capacity Loaning: improve cluster efficiency
  - Loaning is simple. How about Reclaiming?
  - Which on-loan servers should be returned to minimise preemptions?



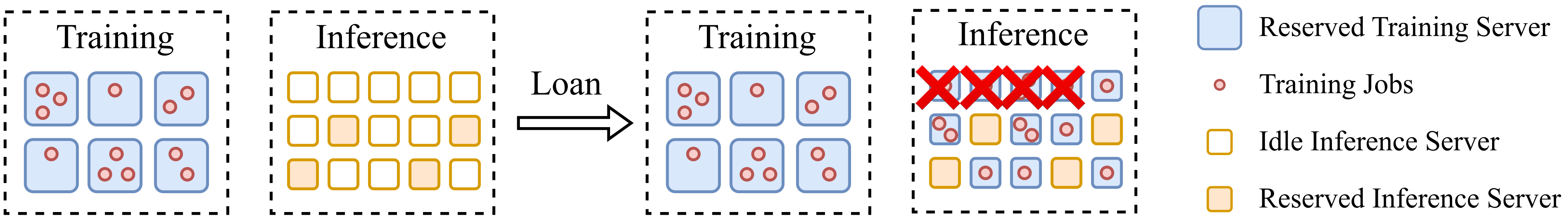
Key Questions

- Capacity Loaning: improve cluster efficiency
  - Loaning is simple. How about Reclaiming?
  - Which on-loan servers should be returned to minimise preemptions?



Key Questions

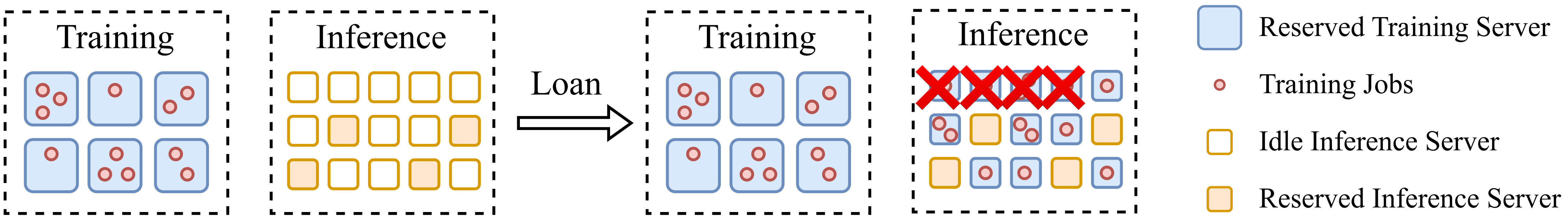
- Capacity Loaning: improve cluster efficiency
  - Loaning is simple. How about Reclaiming?
  - Which on-loan servers should be returned to minimise preemptions?



- Preempted jobs are put back to the queue and wait for available resources.
  - Jobs without checkpointing loses the existing progress.

Key Questions

- Capacity Loaning: improve cluster efficiency
  - Loaning is simple. How about Reclaiming?
  - Which on-loan servers should be returned to minimise preemptions?



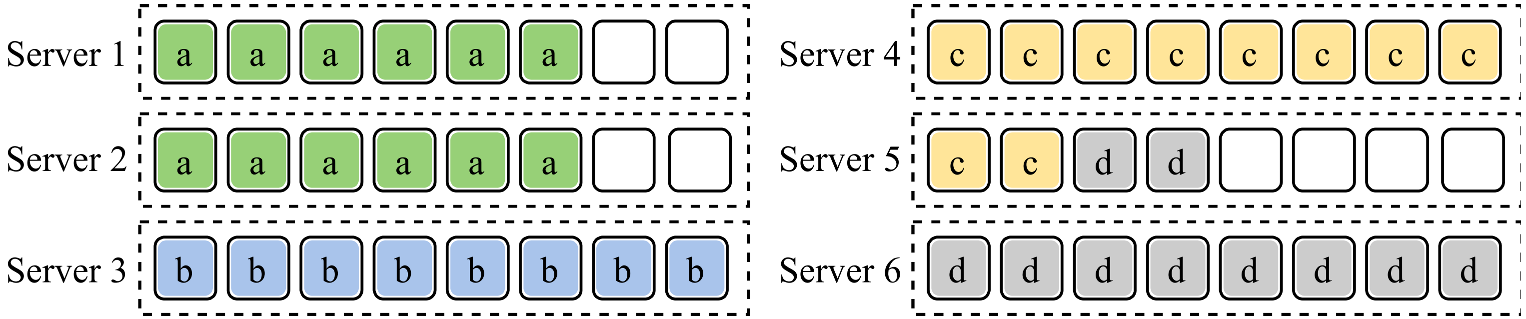
- Preempted jobs are put back to the queue and wait for available resources.
  - Jobs without checkpointing loses the existing progress.

Preemptions are inevitable and hurt job completion time.

Capacity Loaning - Challenges in Server Reclaiming

Objective: minimise the number of preemptions

- Challenges
  - Naively selecting servers leads to unnecessary preemption.
  - Existing job placement might be messy.



Consider a Reclaim request

2 servers

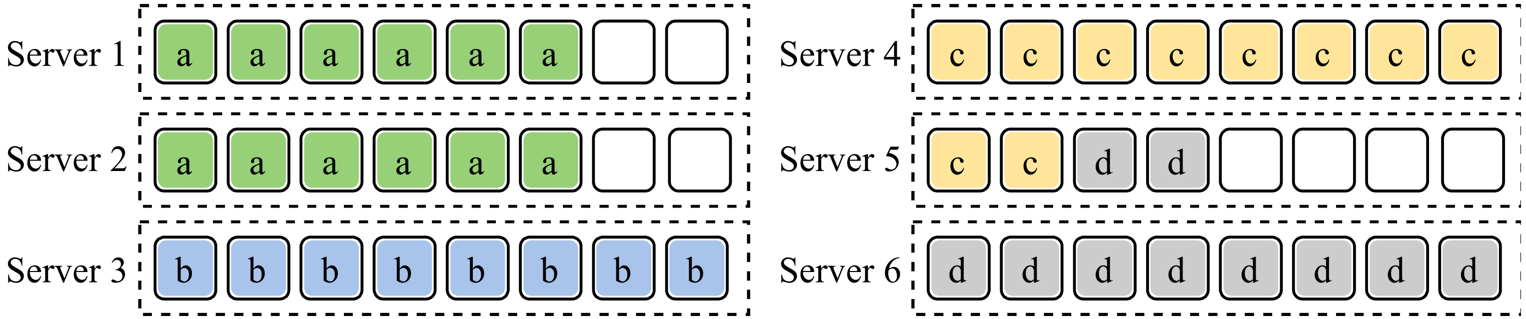
1, 2 or 2, 3 or 4,5

Servers have inter-dependency when co-hosting a job.

Capacity Loaning - Challenges in Server Reclaiming

Objective: minimise the number of preemptions

- Challenges
  - Naively selecting servers leads to unnecessary preemption.
  - Existing job placement might be messy.



Consider a Reclaim request

2 servers

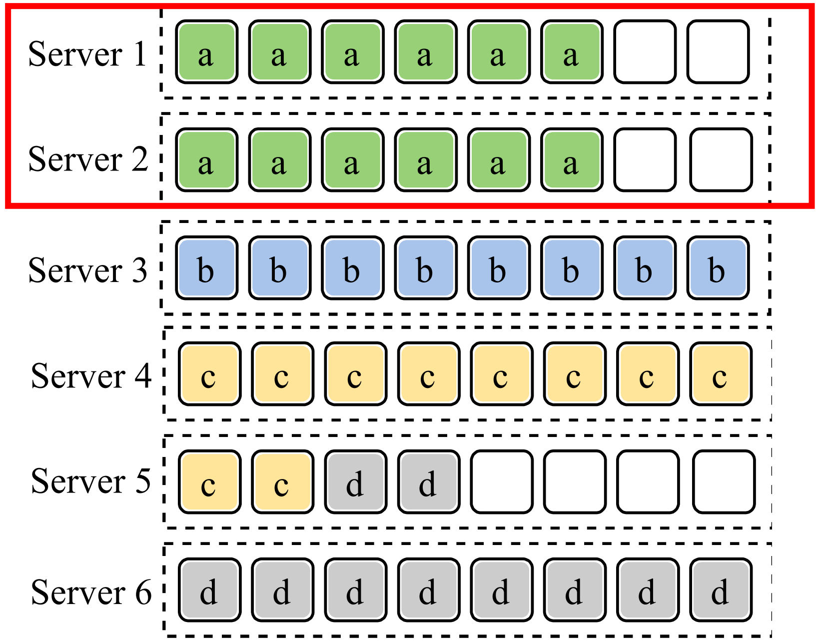
1, 2 or 2, 3 or 4,5

Servers have inter-dependency when co-hosting a job.

Capacity Loaning - Problem Formulation

Objective: minimise the number of preemptions

Given a Reclaim request of N servers and job allocation status {S1: Ja, S2: Ja…}, which servers to select to minimise the number of job preemptions?



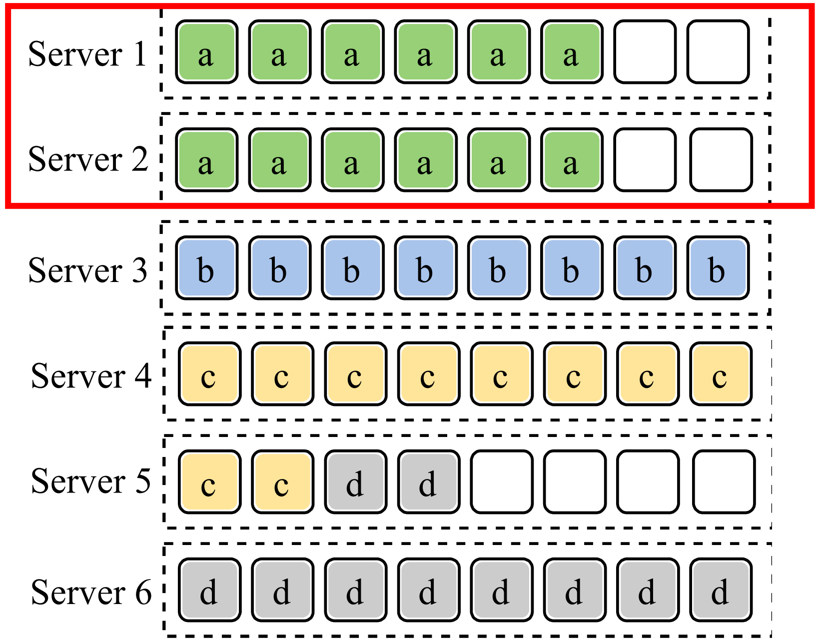
Reclaim Server 1 empties Server 2

Capacity Loaning - Problem Formulation

Objective: minimise the number of preemptions

Given a Reclaim request of N servers and job allocation status {S1: Ja, S2: Ja…}, which servers to select to minimise the number of job preemptions?

Answer: Model as a Knapsack problem



Reclaim Server 1 empties Server 2

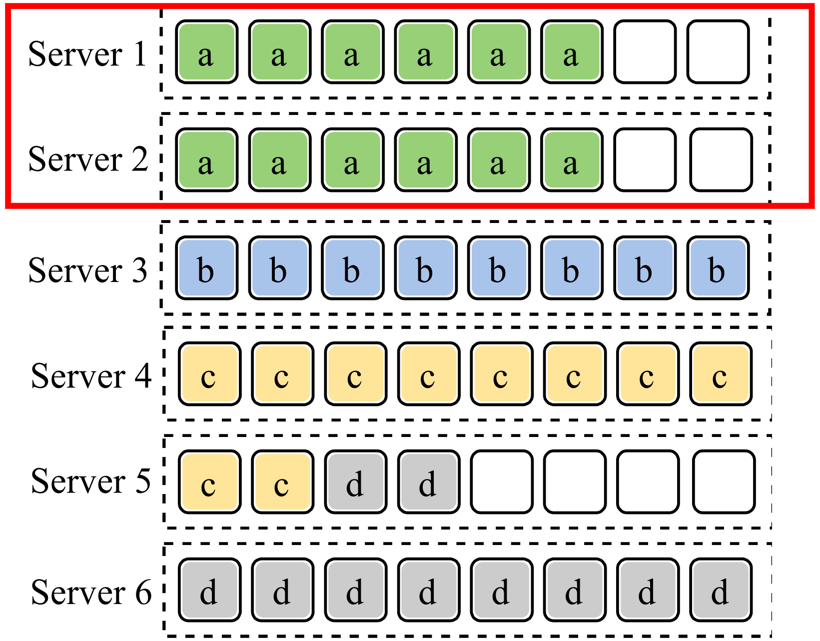
Capacity Loaning - Problem Formulation

Objective: minimise the number of preemptions

Given a Reclaim request of N servers and job allocation status {S1: Ja, S2: Ja…}, which servers to select to minimise the number of job preemptions?

Answer: Model as a Knapsack problem

- Knapsack Size – Reclaim request
- Item – Server
- Item value – # of job preemptions



Reclaim Server 1 empties Server 2

Capacity Loaning - Problem Formulation

Objective: minimise the number of preemptions

Given a Reclaim request of N servers and job allocation status {S1: Ja, S2: Ja…}, which servers to select to minimise the number of job preemptions?

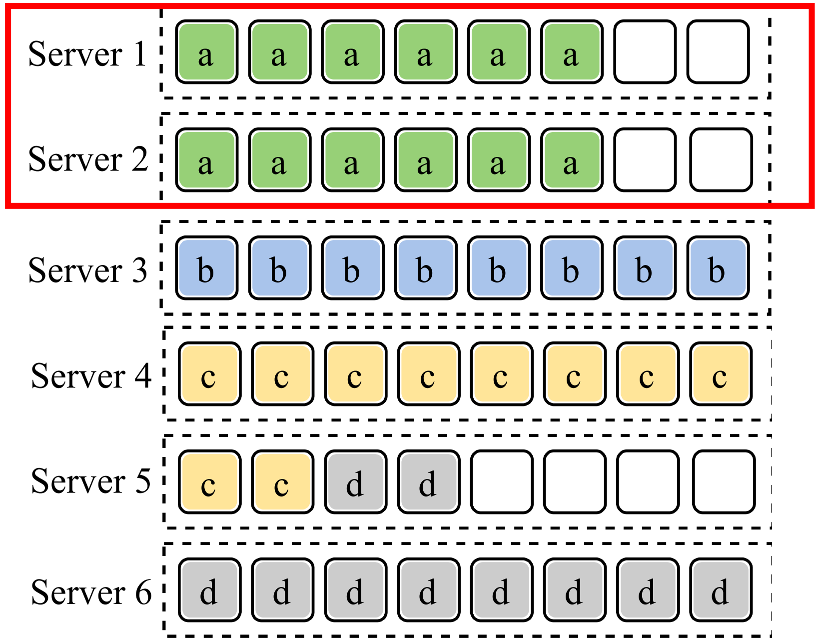
Answer: Model as a Knapsack problem

Knapsack Size – Reclaim request

Item – Server

Item value – # of job preemptions

How about inter-dependency?



Reclaim Server 1 empties Server 2

Capacity Loaning - Problem Formulation

Objective: minimise the number of preemptions

Given a Reclaim request of N servers and job allocation status {S1: Ja, S2: Ja…}, which servers to select to minimise the number of job preemptions?

Answer: Model as a Knapsack problem

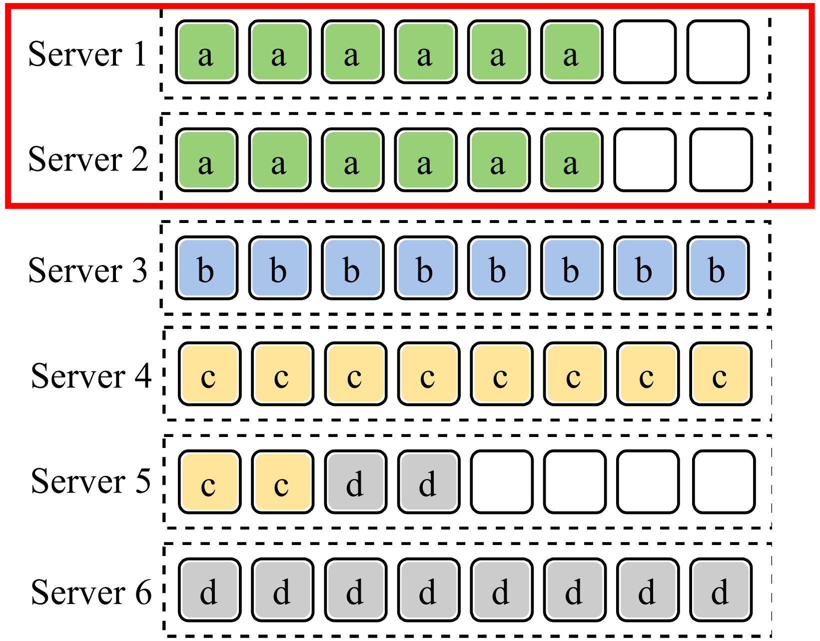
Knapsack Size – Reclaim request

Item – Server

Item value – # of job preemptions

How about inter-dependency?

Knapsack with dependent item values!



Reclaim Server 1 empties Server 2

Capacity Loaning - Problem Formulation

Objective: minimise the number of preemptions

Given a Reclaim request of N servers and job allocation status {S1: Ja, S2: Ja…}, which servers to select to minimise the number of job preemptions?

Answer: Model as a Knapsack problem

Knapsack Size – Reclaim request

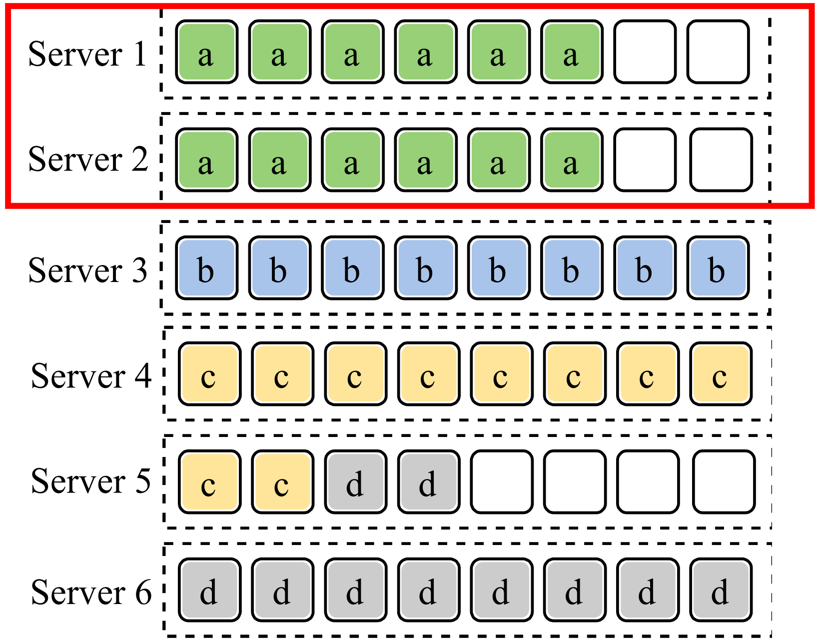
Item – Server

Item value – # of job preemptions

How about inter-dependency?

Knapsack with dependent item values!

NP-hard problem…



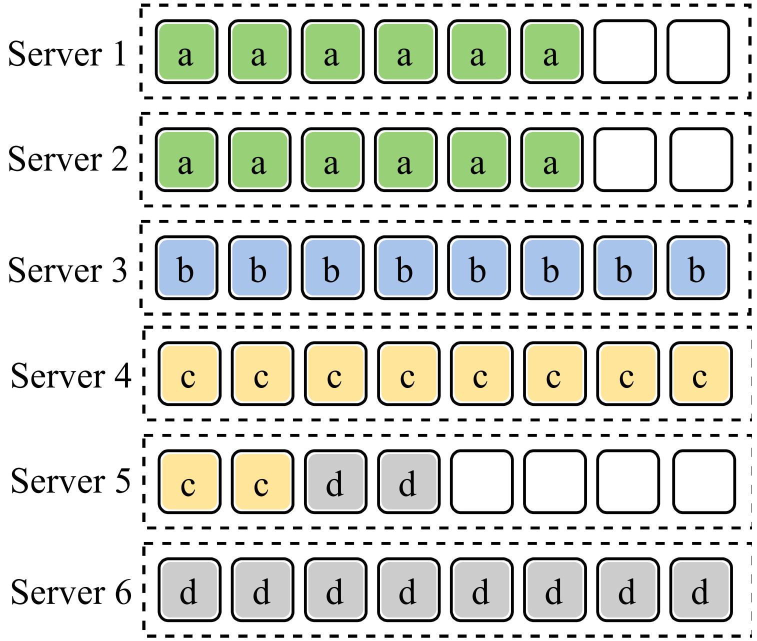
Reclaim Server 1 empties Server 2

Capacity Loaning - Low Overhead Greedy Heuristic

Objective: minimise the number of preemptions

We define a new item value by considering servers cohosting each job

Job	Co-hosted by # of servers	Per Server Value
a	2	0.5
b	1	1
c	2	0.5
d	2	0.5

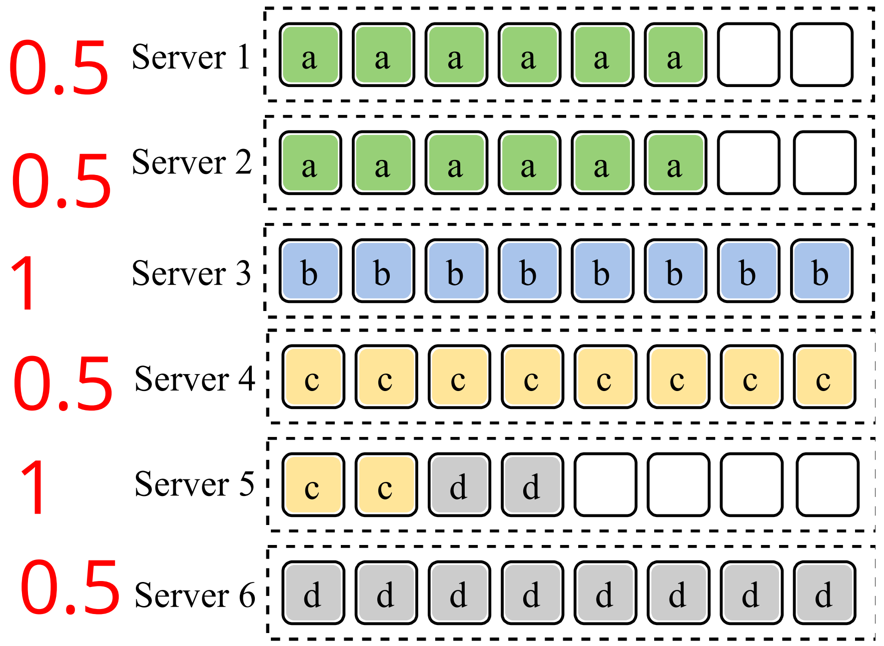


Capacity Loaning - Low Overhead Greedy Heuristic

Objective: minimise the number of preemptions

We define a new item value by considering servers cohosting each job

Job	Co-hosted by # of servers	Per Server Value
a	2	0.5
b	1	1
c	2	0.5
d	2	0.5



Value: Sum of job's server fraction

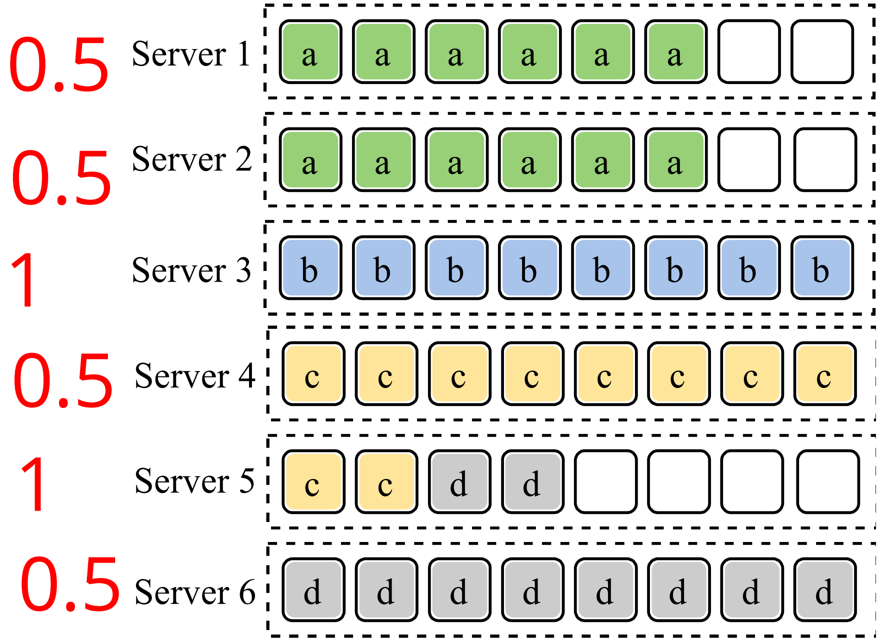
Capacity Loaning - Low Overhead Greedy Heuristic

Objective: minimise the number of preemptions

We define a new item value by considering servers cohosting each job

Job	Co-hosted by # of servers	Per Server Value
a	2	0.5
b	1	1
c	2	0.5
d	2	0.5

1. Greedily select lowest-value server
2. Preempt jobs
3. Update server costs



Value: Sum of job's server fraction

Key Questions

- Capacity Loaning:
  - Objective: Minimise inevitable preemptions
- Elastic Scaling:
  - Objective: Minimise average JCT and assist Capacity Loaning
  - How to determine resource allocation of the elastic training jobs?
  - How to place the jobs in a changing resource pool?

Job running time changes along with allocated resources.

Job Scheduling with Elastic Scaling Workloads

Objective: minimise average Job Completion Time (JCT)

Problem setup

- Elastic job J: running time *RT*
  - Minimum Demand : w^{min}
  - Maximum Demand: w^{max}

Limited Elasticity

  - Training throughput scales linearly within $[w^{min}, w^{max}]$.
- Given *G* GPUs and a set of *N* elastic jobs $\{J_1, J_2 \ldots J_N\}$, decide resource allocation R_i ($g_i^{min} \leq R_i \leq g_i^{max}$) of each job J_i to minimise average JCT.

Resource Allocation of Elastic Scaling Workloads

Objective: minimise average Job Completion Time (JCT)

Cluster Capacity: 8 GPU

Job	**w^{min}**	**w^{max}**	**Min. running time**
A	2	**6**	50
B	2	6	20

Sol.	**Initial Allocation**		**JCT**		**Average JCT**
	A	B	A	B	
1	6	2	50	53.33	51.67
2	2	6	63.33	20	41.67
3	4	4	60	30	45

Resource Allocation of Elastic Scaling Workloads

Objective: minimise average Job Completion Time (JCT)

Cluster Capacity: 8 GPU

Job	w^{min}	w^{max}	Min. running time
A	2	**6**	50
B	2	6	20

Sol.	Initial Allocation		JCT		Average JCT
	A	B	A	B	
1	6	2	50	53.33	51.67
2	2	$\boxed{6}$	63.33	20	$\boxed{41.67}$
3	4	4	60	30	45

Short job

Resource Allocation of Elastic Scaling Workloads

Objective: minimise average Job Completion Time (JCT)

Cluster Capacity: 8 GPU

Job	w_{min}	w_{max}	Min. running time
A	2	**6**	50
B	2	6	20

If the w^{max} of Job A is 3:

Sol.	Initial Allocation		JCT		Average JCT
	A	B	A	B	
1	6	2	50	53.33	51.67
2	2	$\boxed{6}$	63.33	20	$\boxed{41.67}$
3	4	4	60	30	45

Short job

Resource Allocation of Elastic Scaling Workloads

Objective: minimise average Job Completion Time (JCT)

Cluster Capacity: 8 GPU

Job	w_{min}	w_{max}	Min. running time
A	2	**6**	50
B	2	6	20

Sol.	Initial Allocation		JCT		Average JCT
	A	B	A	B	
1	6	2	50	53.33	51.67
2	2	$\boxed{6}$	63.33	20	$\boxed{41.67}$
3	4	4	60	30	45

Short job

If the w^{max} of Job A is 3:

Job	w_{min}	w_{max}	Min. running time
A	2	**3**	100
B	2	6	20

Sol.	Initial Allocation		JCT		Average JCT
	A	B	A	B	
1	3	5	100	24	62
2	2	6	106.67	20	63.33

Resource Allocation of Elastic Scaling Workloads

Objective: minimise average Job Completion Time (JCT)

Cluster Capacity: 8 GPU

Job	wmin	wmax	Min. running time
A	2	**6**	50
B	2	6	20

If the w^{max} of Job A is 3:

Job	wmin	wmax	Min. running time
A	2	**3**	100
B	2	6	20

Sol.	Initial Allocation		JCT		Average JCT	
	A	B	A	B		
1	6	2	50	53.33	51.67	
2	2	6	63.33	20	41.67	Short job
3	4	4	60	30	45	

Sol.	Initial Allocation		JCT		Average JCT	
	A	B	A	B		
1	3	5	100	24	62	Long job
2	2	6	106.67	20	63.33	

Resource Allocation of Elastic Scaling Workloads

Objective: minimise average Job Completion Time (JCT)

Cluster Capacity: 8 GPU

Job	w_{min}	w_{max}	Min. running time
A	2	**6**	50
B	2	6	20

Sol.	Initial Allocation		JCT		Average JCT
	A	B	A	B	
1	6	2	50	53.33	51.67
2	2	6	63.33	20	41.67
3	4	4	60	30	45

Short job

If the w_{max} of Job A is 3:

Job	w_{min}	w_{max}	Min. running time
A	2	**3**	100
B	2	6	20

Sol.	Initial Allocation		JCT		Average JCT
	A	B	A	B	
1	3	5	100	24	62
2	2	6	106.67	20	63.33

Long job

Shortest-job-first does not always work for elastic training jobs.

Lyra - Job Scheduling Heuristic

Objective: minimise average Job Completion Time (JCT)

Job = Base (min.) Demand + Flexible (max. - min.) Demand

Lyra - Job Scheduling Heuristic

Objective: minimise average Job Completion Time (JCT)

Job = Base (min.) Demand + Flexible (max. - min.) Demand

- First-class citizen
- Gang Scheduling
- Incurs queuing delay

Lyra - Job Scheduling Heuristic

Objective: minimise average Job Completion Time (JCT)

Job = Base (min.) Demand + Flexible (max. - min.) Demand

Base (min.) Demand:
- First-class citizen
- Gang Scheduling
- Incurs queuing delay

Flexible (max. - min.) Demand:
- Non-binary allocation
- Impact running time
- 0 for inelastic jobs

Lyra - Job Scheduling Heuristic

Objective: minimise average Job Completion Time (JCT)

Job = Base (min.) Demand + Flexible (max. - min.) Demand

Base (min.) Demand	Flexible (max. - min.) Demand
• First-class citizen	• Non-binary allocation
• Gang Scheduling	• Impact running time
• Incurs queuing delay	• 0 for inelastic jobs

Two-phase resource allocation:

1. Prioritise Base Demand using Shortest-Job-First (SJF) to minimise queuing time
2. Allocate the remaining resources to fulfill the Flexible Demand
   1. Minimise job running time

Lyra - Resource Allocation of Flexible Demand

Objective: minimise average Job Completion Time (JCT)

Phase 2: Allocate the remaining resources to fulfill Flexible Demand

Job	w_{min}	w_{max}	Min. running time
A	2	**3**	100
B	2	6	20

Job B accepts an extra [1, 4] workers

Job A: 2 GPU/worker, Job B: 1 GPU/worker

Multiple-choice Knapsack problem (select at most one item from each group)

Lyra - Resource Allocation of Flexible Demand

Objective: minimise average Job Completion Time (JCT)

Phase 2: Allocate the remaining resources to fulfill Flexible Demand

Job	w^{min}	w^{max}	Min. running time
A	2	**3**	100
B	2	6	20

Job B accepts an extra [1, 4] workers

Job A: 2 GPU/worker, Job B: 1 GPU/worker

Multiple-choice Knapsack problem (select at most one item from each group)

Knapsack Size – available GPUs

Group – job

Item - possible allocation of the job

Item weight – extra GPUs required

Value – running time reduction

Lyra - Resource Allocation of Flexible Demand

Objective: minimise average Job Completion Time (JCT)

Phase 2: Allocate the remaining resources to fulfill Flexible Demand

Job	w_{min}	w_{max}	Min. running time
A	2	**3**	100
B	2	6	20

Job B accepts an extra [1, 4] workers

Job A: 2 GPU/worker, Job B: 1 GPU/worker

Multiple-choice Knapsack problem (select at most one item from each group)

Knapsack Size – available GPUs

Group – job

Item - possible allocation of the job

Item weight – extra GPUs required

Value – running time reduction

Group	Item	Weight	Value
A	1	2	50
B	1	1	20
	2	2	30
	3	3	36
	4	4	40

2 worker -> 150
3 worker -> 100

Lyra - Job Placement

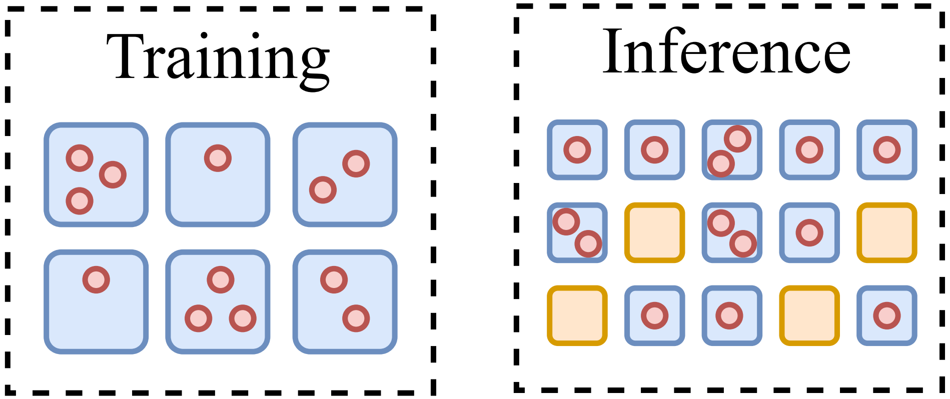
Objective: minimise preemption and average Job Completion Time (JCT)

We follow the bin packing with best-fit decreasing job placement strategy.

How does elastic training jobs help to minimise preemption?

Different servers are prioritised for jobs during placement.

- Base Demand
- Non-fungible jobs



- Flexible Demand
- Fungible & Heterogeneous jobs

Lyra - Job Placement

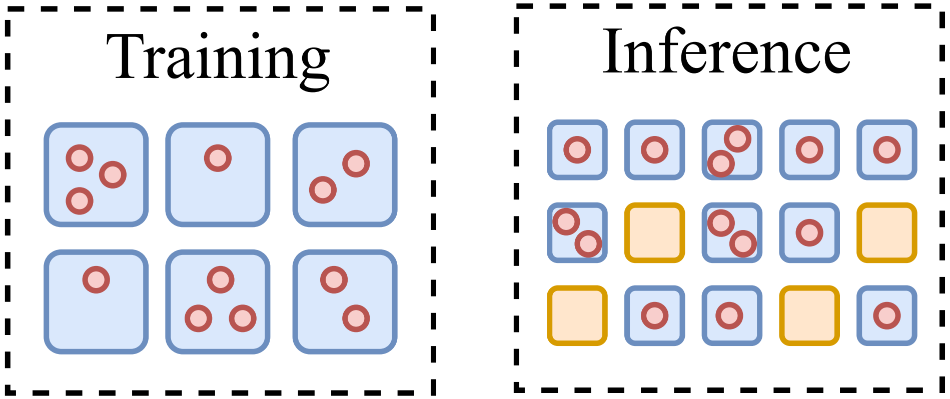
Objective: minimise preemption and average Job Completion Time (JCT)

We follow the bin packing with best-fit decreasing job placement strategy.

How does elastic training jobs help to minimise preemption?

Different servers are prioritised for jobs during placement.

- Base Demand
- Non-fungible jobs



- Flexible Demand
- Fungible & Heterogeneous jobs

- Place base demand and flexible demand on separate groups
- Vacate servers by scaling in elastic jobs to meet reclaim request first

Evaluation – Experiment Setup

- 2V100 => 1V100 + 3 T4 => 1B + 3 * 1/3B
- LBBSP
- Trace: 15-day job trace, 50390 training job + 3544 V100 GPUs for training
- Baseline: No capacity loaning or elastic scaling
- Scenarios
    1. Basic: ~5% of large jobs support elastic training workload + 21% of jobs are fungible workload
    2. Advanced: Basic + 10% jobs are heterogenous workload (non-ideal performance)
    3. Ideal: all jobs are elastic, fungible and heterogeneous workload

Evaluation – Testbed Results

A 8-hour sampled trace: 180 training jobs, 10 elastic jobs

Cluster: 32 x V100 GPUs, 32 x T4 GPUs

Scenario	Scheme	Queuing Time (s)			JCT (s)			Preemption
		Mean	Median	95%ile	Mean	Median	95%ile	Ratio
Overall	Baseline	1532	772	1003	4078	2183	3096	0
	Lyra	1109	503	738	3335	1747	2731	18%

6 loaning, 8 reclaiming involving 10 servers, 73 scaling operations

Evaluation – Simulation Results

Avg. Queuing time: 1.52x -> 1.67x -> 2.66x

Avg. JCT: 1.48x -> 1.59x -> 1.87x

#	Scenario	Scheme	Queuing Time (s)			JCT (s)			GPU Usage		Preemption
			Mean	Median	95%ile	Mean	Median	95%ile	Training	Overall [1]	Ratio [2]
1	—	Baseline [3]	3072	55	8357	16610	791	82933	0.72	0.52	0
2	Basic	Lyra	2010	26	3358	11236	568	56477	0.86	0.65	12.24%
3	Advanced	Lyra	1835	24	3238	10434	525	56553	0.86	0.68	7.35%
5	Ideal	Lyra	1157	22	3204	8891	422	41146	0.93	0.72	5.72%

Preemption ratio drops by 2.14x in Ideal scenario.

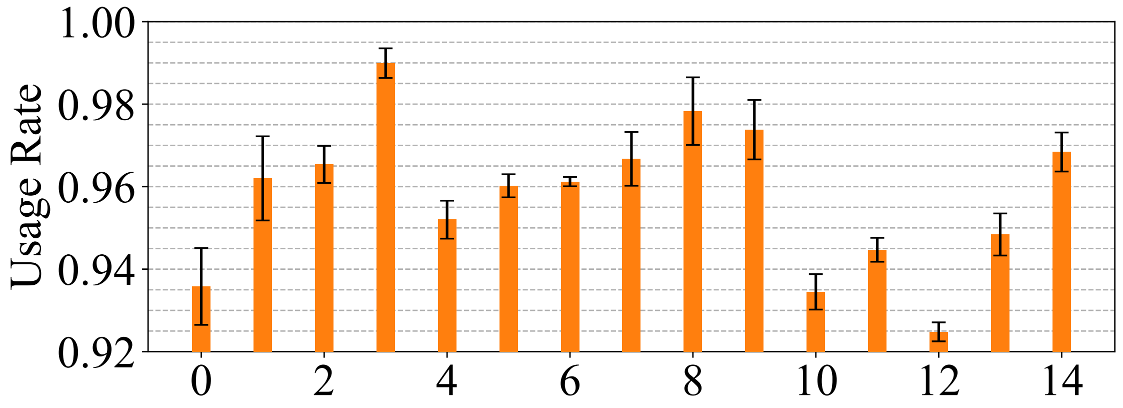
Resources on On-loan Servers

Queuing time and JCT of jobs running on on-loan servers.

	Queuing Time			JCT		
	Mean	Median	95%tile	Mean	Median	95%tile
Baseline	4573	1283	23351	11547	2122	60170
Lyra	1029 (4.44x)	272 (4.71x)	7249 (3.22x)	6832 (1.69x)	1256 (1.69x)	35604 (1.69x)

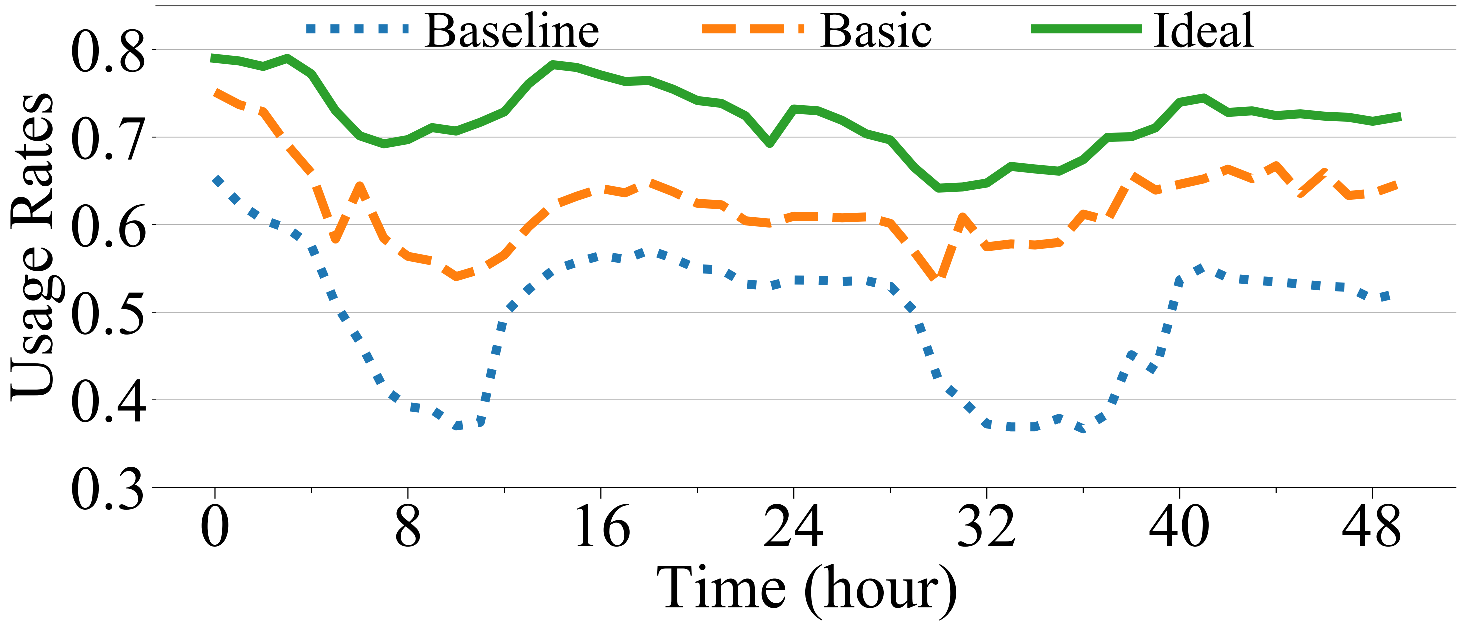
Resource usage of on-loan servers

Above ~93%



GPU Utilisation in different scenarios

48-hour overall GPU utilisation in Basic and Ideal scenario



Check out our paper for more evaluation results.

Summary of contributions

- An elastic GPU cluster scheduler for deep learning.
- Exploits cluster-level elasticity by capacity loaning and job-level elasticity by scheduling elastic scaling jobs.
- Proposes efficient heuristics for capacity loaning and elastic job scheduling.

Thanks!