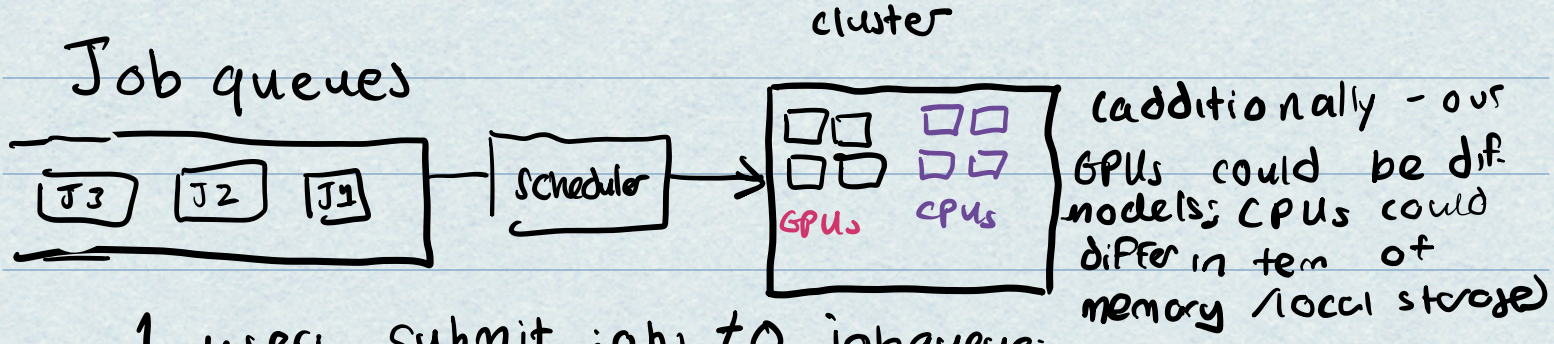
Agenda:

- cluster scheduling fundamentals
- Looking at a real ML cluster & tradeoffs
- Hetero-geneity aware clusterscheduling
- Alpaserve: model parallelism w/ statistical multiplexing

*for the rest of the class, we won't be talking about Attention specifically, but rather, more general training/inference workloads

Part 1: cluster scheduling fundamentals

- setting: we have a cluster of resources, queue of jobs

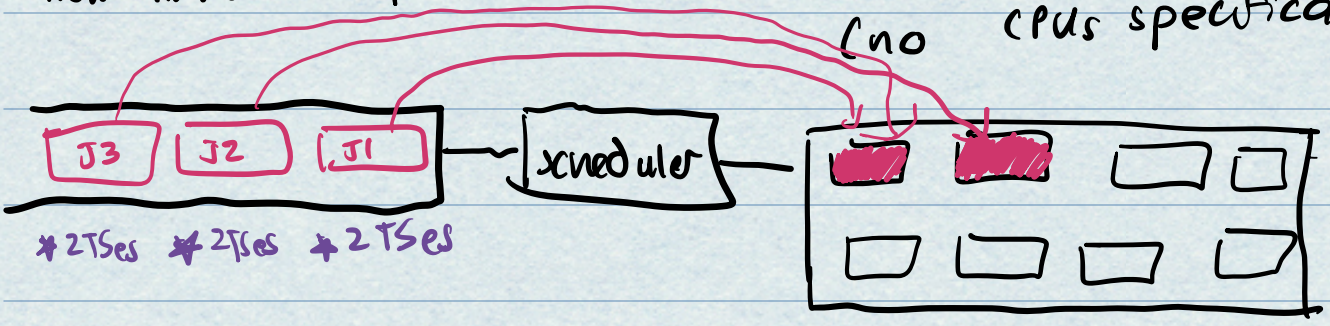


1. users submit jobs to jobqueue:
   - inference OR training (or-any CPU workload - cluster scheduling NOT specific to ML)
   - each job specifies resources it needs (# CPUs, # GPUs)
   - scheduler allocates entire set of resources (no sharing :) to job, job run on cluste

* Discussion: what is a real life GPU cluster you all have interacted with? SLURM!

↳ provides many "management" features/observability features in addition to just workload scheduling

* how this works: suppose we have a simple-8-GPU cluster (no CPUs specifically)



- at iteration, we try to schedule job "greedily" and also see if any existing job is finished
  ↳ depending on setting, scheduler may or may not know length of time each job takes

1. TS 0: J1 arrives, assigned to GPU 0.
2. TS 1: J2 arrives, assigned to GPU 1
3. TS 2: J1 completed, GPU 0 freed
4. Also at TS 2: J3 arrives: now assigned to GPU 0 as GPU 0 is free

* How does cluster scheduler decide which resources to give and even which jobs to schedule?
  ↳ could have multiple jobs arriving at once/a choice of which jobs to run (not all may be able to fit)

- scheduler will use a POLICY that satisfies a desired OBJECTIVE such as:

1) Throughput: execute as many jobs in unit time

2) Cost: if cluster machines have some associated cost, minimize total cost to run all jobs

3) Fairness: give every job job a EQUAL SHARE of resources

- what are SPECIFIC METRICS TO OPTIMIZE?
  - Job completion time (JCT): what is TOTAL TIME job took, including any time spent waiting a queue
  - makespan: Total time to execute a batch of jobs.

*now lets go through a few policies specifically designed to optimize some of these objectives

- Suppose we have the following jobs: (durations known ahead of time), 2 GPUs

*all jobs require 1 GPU

Job	Duration	Arrival Time
0	30	0
1	10	0
2	25	1/4
3	40	

- Activity: with 2-3 people around you - discuss the SIMPLEST POSSIBLE scheduling algs (the

one we went over in basic example).

↳ this policy is called FCFS! or "first-come-first serve"

- might help to fill in followins table.

Execution time so far:

Job ↓ TS→	0	1	4	10	30	35	70	JCT
0 (30)	0	1	4	10	30	—	—	30
1 (10)	0	1	4	10	—	—	—	10
2 (25)	—	0	0	0	20	25	—	35 - 1 = 34
3 (40)	—	—	0	0	0	5	40	70 - 4 = 66

* time 0: jobs 0 / 1 scheduled
* time 1: job 2 arrives, but GPUs are occupied
* time 4: Job 3 arrives, queued behind Job 2, and GPUs still utilized
* time 10: Job 1 finishes - (can calculate JCT) - now schedule Job 2, was next in q
* time 30: Job 0 finishes - now schedule Job 3

- Discuss: what was makespan?
  - ↳ 70 (time for all jobs to finish)
  - avg JCT is = 35 (note that avg JCT doesn't take into accnt sizes of diff Jobs

* main issue w/ FCFS:

- jobs will suffer from LONG QUEUING delay (e.g. think abt short job queued behind long jobs)

*idea: we can PREEMPT previously scheduled jobs, replace them with jobs in queue

* next scheduling alg - SJF - shortest job first

- schedule jobs in order of shortest duration (preempt longer jobs if no speed)

*Let's consider some scenario as before - what will JCT and makespan be?

Job	Duration	Arrival Time
0	30	0
1	10	0
2	25	1
3	40	4

Job \ TS→	0	1	4	10	26	39	66	JCT
0 (30)	0	1	1	1	17	30	-	39
1 (10)	0	1	4	10	-	-	-	10
2 (25)	-	0	3	9	25	-	-	25
3 (40)	-	-	0	0	0	13	40	62

*avg JCT: 34

*makespan: 64

} LOWER THAN FCFS

Timestamp 0: 0 and 1 scheduled

Timestamp 1: 2 arrives and PREEMPTS 0

Timestamp 2: 3 arrives, longer than 0, 1, 2, so stays in queue

Timestamp 10: 1 finishes, 0 scheduled back

Timestamp 26: Job 2 completes, 3 finally scheduled

Timestamp 39: Job 0 finishes

Timestamp 66: Job 3 finishes

□ One more: priority (each job comes w/ an assigned priority)

↳ Jobs w/ higher priority always preempt lower

priority ↑ is higher priority

Job	Duration	Arrival Time	Priority
0	30	0	3
1	10	0	1
2	25	1	2
3	40	4	4

Job ↓ TS →	0	1	4	30	44	52	53	JCT
0 (30)	0	1	4	30	—	—	—	30
1 (10)	0	1	1	1	1	9	10	53
2 (25)	—	0	3	3	17	25	—	51
3 (40)	—	—	0	26	40	—	—	40

* at TS 1, 2 arrives and preempts 1

* at TS 4, 3 arrives and preempts 2
* Ts 30: 1 finishes, 2 back on
* Ts 44: 3 finishes, 1 back on
    * makespan: 53
    * average JCT: 43.5

- In summary, on this cluster/workload:

Policy	Average JCT	makespan
FIFO (FCFS)	35	70
SJF	34	66
PRIORITY	43.5	53

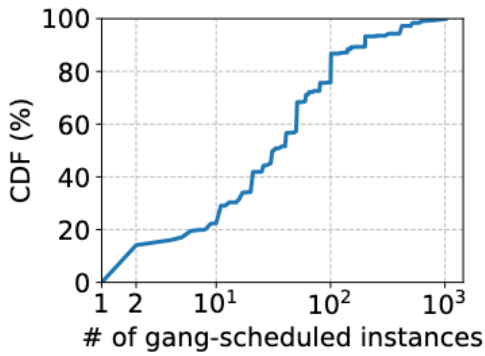
* on this workload, priority-based increases avg JCT, but reduces makespan
- optimizing for makespan is kind of like optimizing for tput

* Challenges faced by a REAL ML cluster:

- all graphs are from this NSDI paper: https://www.usenix.org/system/files/nsdi22-paper-weng.pdf
- from real ML cluster in Alibaba:
    - collected in 2020.
    - 1.2 million training and inference job
    - 6700+ GPUs, on 1800 nodes (each server either has 2 or 8 GPUs)
    - 85% of jobs are gang-scheduled:

Table 1: Machine specs of GPU clusters in the existing trace analysis works. GPUs with † are equipped with NVLink [12]. The Philly trace does not reveal CPU specs and GPU types.

System	#CPUs	Mem (GiB)	#GPUs	GPU type	#Nodes
PAI	64	512	2	P100	798
	96	512	2	T4	497
	96	512	8	Misc.	280
	96	384	8	V100M32†	135
	96	512/384	8	V100†	104
	96	512	0	N/A	83



(b) CDF of gang-scheduled task instances.

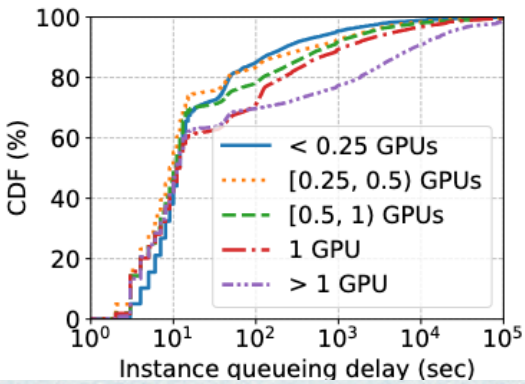
* note each machine has dif. CPU types / memory amount (this is 1 of many clusters)

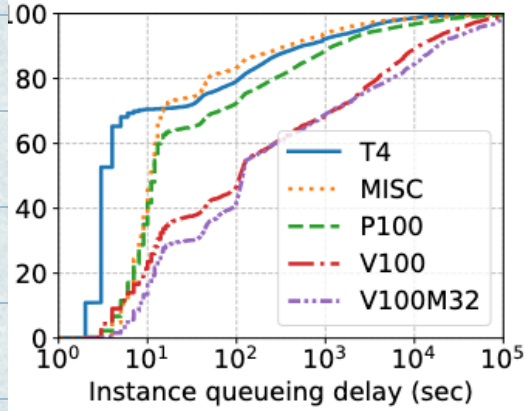
- for when multiple tasks need to run at same time

* observation 1: queuing delay increases when...

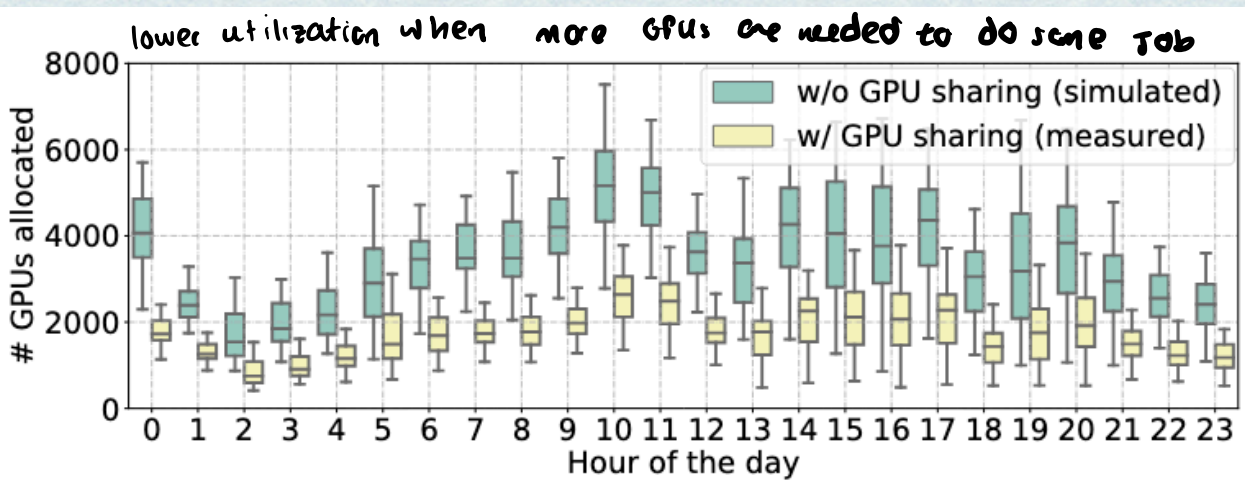
resource requirements for jobs increase

more powerful ML accelerators are requested

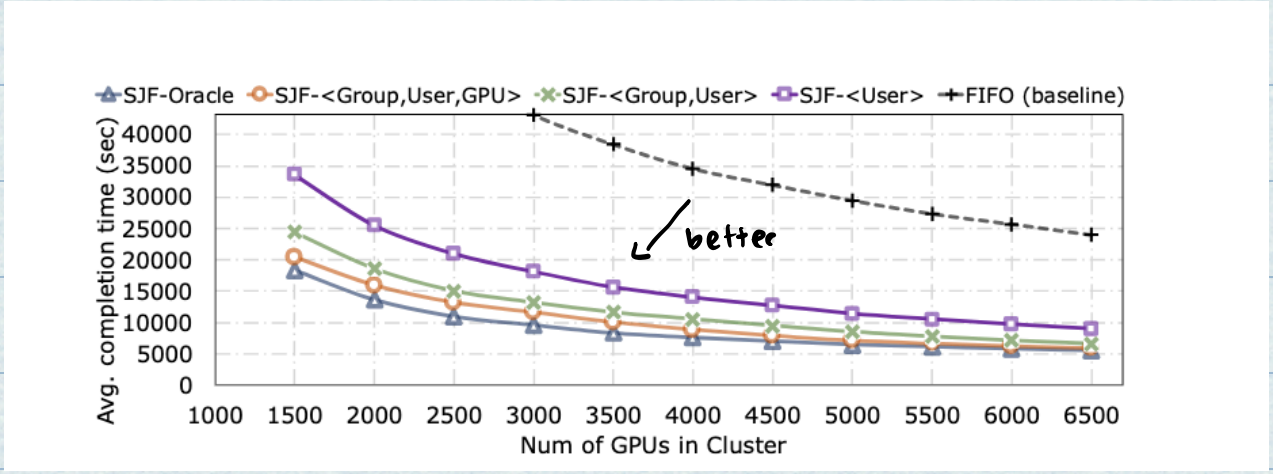




* observation 2: packing multiple jobs on same GPU (space sharing) will improve utilization



* observation 3: SJF policies are better than FIFO



- Additional General challenges for ML Training jobs:

1) Performance heterogeneity:

- diverse set of models (LLMs, vision, etc)
- diverse set of HW platforms (dif. GPU generations, TPUs, etc)

2) Distributed jobs:

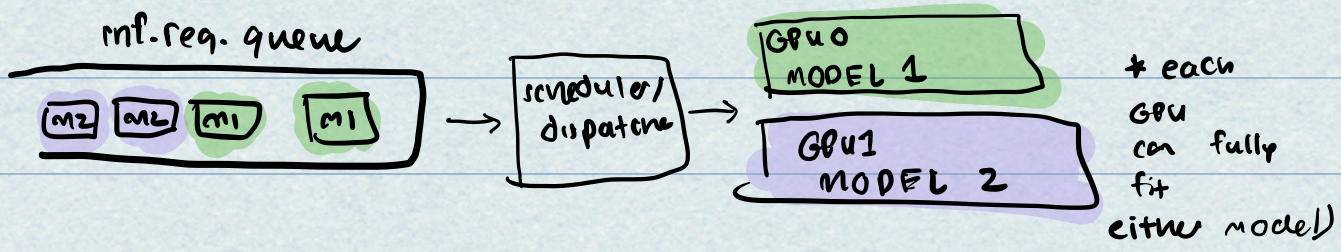
- generally better for multi-GPU jobs to be on same server, will maximize inter-node BW

3) Large state:

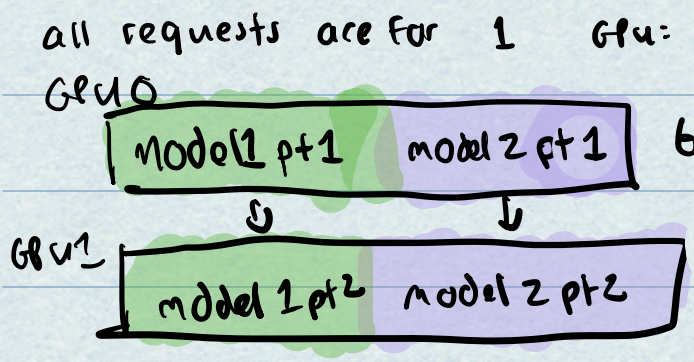
- if a training job is preempted, job will have to CHECKPOINT weights, later load this back

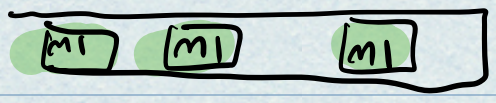
* PART 3: model parallelism while serving (alpaserve)

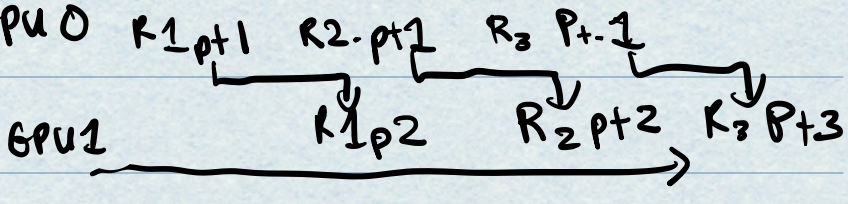
- original Alpaserve paper: https://www.usenix.org/conference/osdi23/presentation/li-zhouhan
- scenario: we have 2 models, 2 GPUs, our cluster serves inference requests for both at SAME TIME
- what would be strawman way to serve this?



- we would place 1 model per GPU
- no model parallelism needed (and model parallelism could cause overhead!
  - ↳ due to either pipeline bubbles (pipeline parallelism)
  - ↳ or frequent communication/synchronization (tensor model parallel)
- why could this scheme be bad?
  - what if all our requests in trace at some point in time are for a SINGLE model (a burst!) - then we're effectively wasting half of the GPUs we have available to us
- idea: let's use model parallelism to use both GPUs even if all requests are for 1 GPU:







Time → w/o model parallelism - one of the GPUs would be wasted

- the paper more generally reasons about:
  - when model parallelism > replication (above,

we cannot replicate, but you could imagine choice btwn following:

Replication (imagine 2 of each model fit on GPU)

Model parallelism:

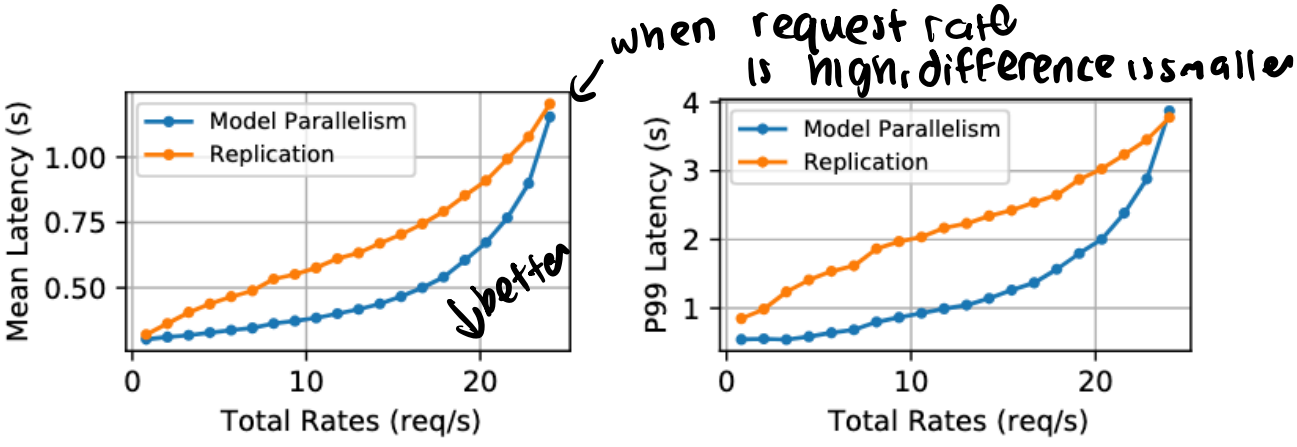


Figure 5: Serving performance with changing arrival rates. Model parallelism is beneficial for smaller rates.

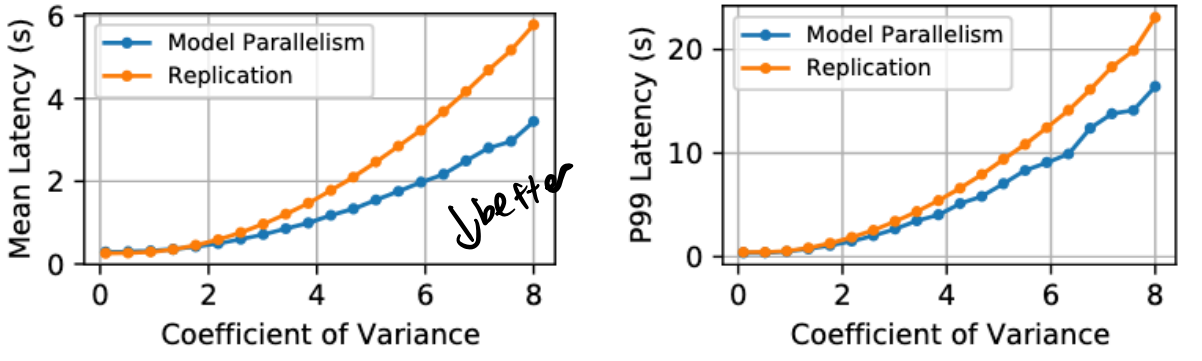


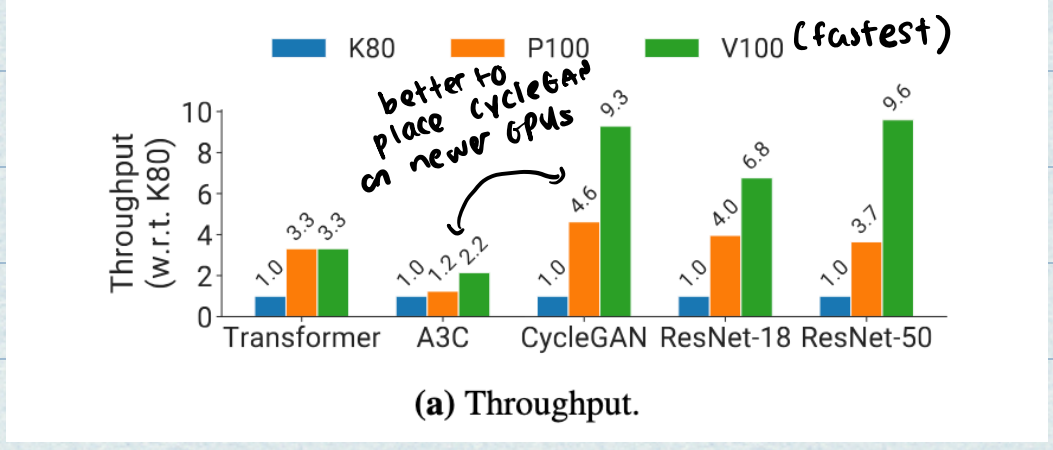
Figure 6: Serving performance with changing CVs. Model parallelism is beneficial for larger CVs. → measure of burstiness in which models requested

* paper has strategies to automatically decide placement when:

- many more than 2 models or 2 GPUs
- choice btwn INTER-OPERATOR PARALLELISM (divide at layer boundaries, like for pipeline parallel) and INTRA-OPERATOR PARALLELISM (divide within ops like in tensor model parallelism)

PART 4: ACCOUNTING FOR HETEROGENEITY IN CLUSTER SCHEDULERS

→ more info in OSDI 20 paper on Gavel: https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak

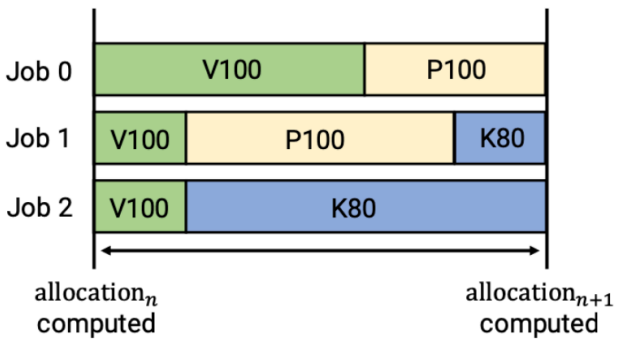


(a) Throughput.

key problem:
- not every job benefits at same proportional rate from better GPUs

- gavel paper lets us reason about resource allocation via optimization problems:

$$X^{\text{example}} = \begin{array}{c} \begin{matrix} V100 & P100 & K80 \end{matrix} \\ \begin{pmatrix} 0.6 & 0.4 & 0.0 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0.0 & 0.8 \end{pmatrix} \end{array} \begin{matrix} \text{job 0} \\ \text{job 1} \\ \text{job 2} \end{matrix}$$



- we have 3 GPU types, 3 Jobs

- allocation matrix: X_{ij} is fraction of time Job i spends on GPU j. → e.g. job 0

- assume for each job and GPU type, we can estimate (from offline mmt):

T_{ij} → throughput of job i on GPU j. (iterations/s)

- given matrix X (allocations) and matrix T (measured tputs) we can calculate:

effective tput: sum of tput achieved on each resource type, weighted by time spent on that type:

$$\text{tput}(i, X) = \sum_{j \in \text{accelerators}} T_{ij} \cdot X_{ij}$$

(job i)

(throughput normalized by allocation)

- now, we have a goal: find some allocation X which satisfies some global perf. objective

$$\text{maximize}(X) \sum_{i \in \text{jobs}} \text{tput}(i, X)$$

(can be represented)

this is a linear program that can be solved by existing libraries

s.t.

$0 \leq X_{ij} \leq 1 \quad \forall (i,j)$ → each individual alloc is valid fraction

$\sum_j X_{ij} \leq 1 \quad \forall i$ → total allocation for job can't exceed 100%

- one more constraint abt total resources allocated not exceeding what is available

- we can now express our previous scheduling policies as objectives:

$$\text{Minimize}_X \min_i \frac{\text{num\_steps}_i}{\text{tput}(i, X)} \rightarrow \text{minimize runtime of shortest job (SJF)}$$

$$\text{Minimize}_X \max_i \frac{\text{num\_steps}_i}{\text{tput}(i, X)} \rightarrow \text{minimize runtime of longest job (minimizing makespan).}$$

Further resources

- This lecture is mostly from cs229s lecture:

https://docs.google.com/presentation/d/12BUjwe8hr1ofHozLWO41D_aNG_JyC8zeQclfrwKrxE0/edit#slide=id.p

- Alibaba cluster analysis paper (Weng et. al):

https://www.usenix.org/system/files/nsdi22-paper-weng.pdf

- Alpaserve paper (Li et. al):

https://www.usenix.org/conference/osdi23/presentation/li-zhouhan

- Gavel paper (Narayanan et. al):

https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak