Hi!

CS 744: NEXUS

Shivaram Venkataraman

Fall 2021

ADMINISTRIVIA

Course Project Proposals

- Due Oct 25! → Monday
- See Piazza for template

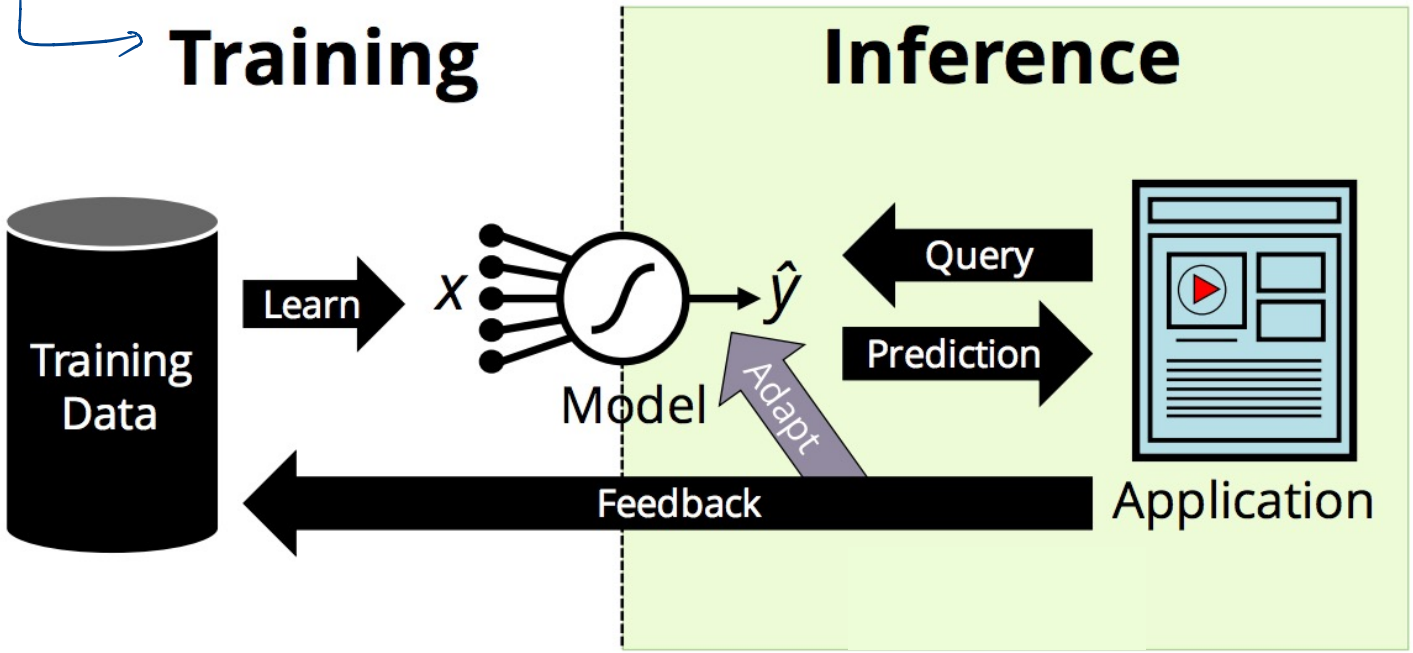
Midterm details

- Oct 28th: Includes papers from "Datacenter as a Computer" to "Nexus"
- Open book, open notes
- Held in class time 9.30-10.45am Central Time

MACHINE LEARNING: INFERENCE

Pytorch DDP

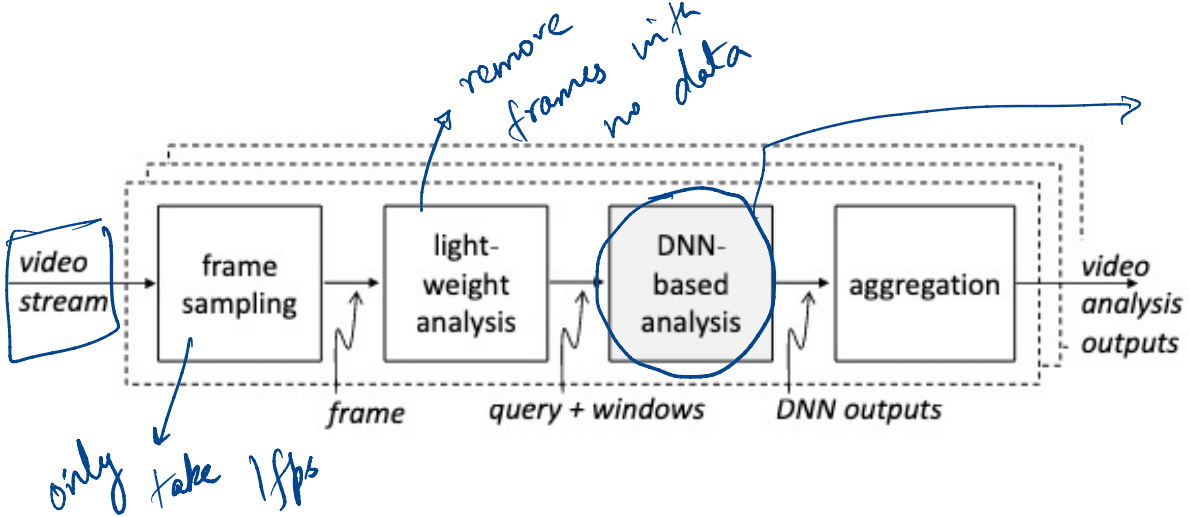
Pipedream



EXAMPLE APPLICATION

Video analysis service → Traffic cameras → stream data to the cloud

- Thousands of streams, thousands of tenants → each tenant is running a diff model
- Each stream is processed by a DNN-based "query"
- Latency SLOs (10s to 100s of ms)



Scheduler to determine placement and batching

GOAL: HIGH GPU UTILIZATION

Placement

↳ Overhead to moving a model to GPU

↳ Decide which model(s) go to which GPU

↳ Cost-effective when you use an accelerator like a GPU

Batching

$$\text{batch\_lat}(b) = \alpha b + \beta,$$

10 frames per sec

β → fixed overhead

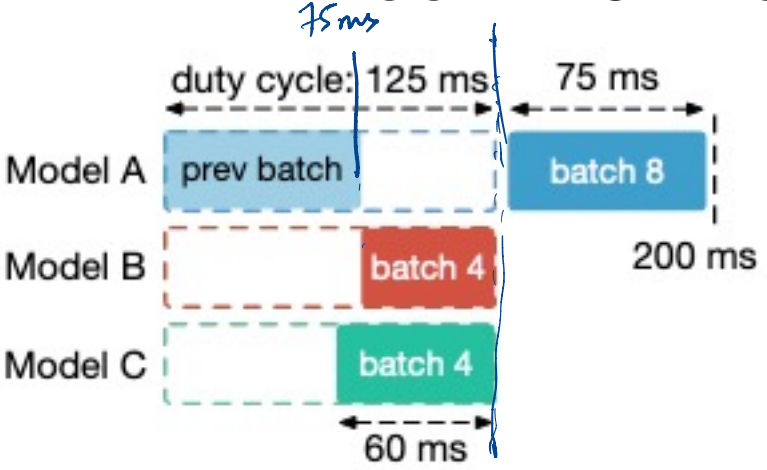
α → per example cost in this batch

Larger batch size

↳ computationally efficient

↳ wait until you can form this batch

SCHEDULING BATCHED EXECUTION



Model A			Model B			Model C		
Batch	Lat	Req/s	Batch	Lat	Req/s	Batch	Lat	Req/s
4	50	80	4	50	80	4	60	66.7
8	75	107	8	90	89	8	95	84
16	100	160	16	125	128	16	125	128

(b) Residual workload

Consider A, B being colocated

A for 75 ms, B for 50 ms, A for 75ms ...

Latency (A) + Latency (B) will both meet SLOs

A & C cannot be colocated because that will lead to SLO violation

BATCH-AWARE SCHEDULING

Input need to satisfy ↑

every request latency ↑ ≤ SLO / if not drop

→ latency at various batch sizes

Inputs: Request rate, SLO for each model, Profiles at batch size

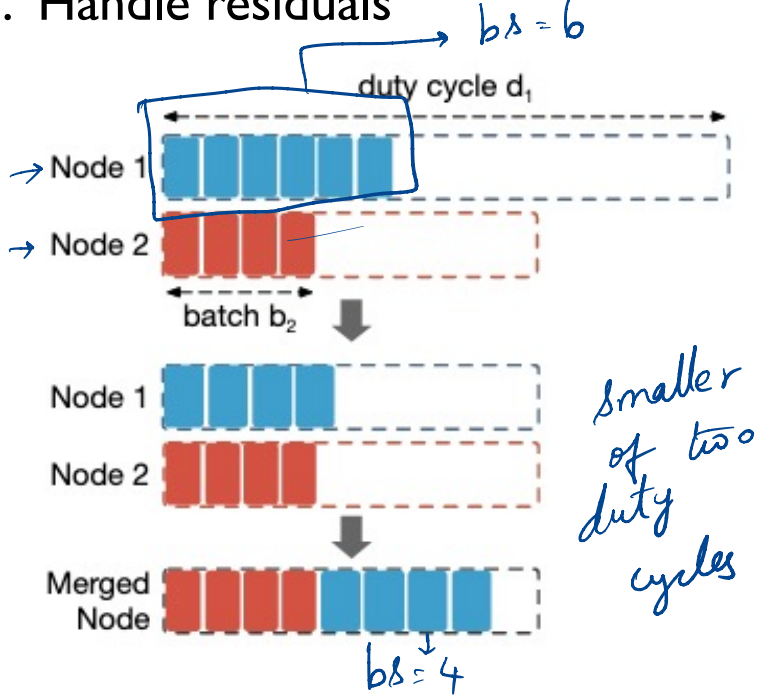
Approach: Allocate "full" GPUs based on load. Handle residuals

If req. rate = 1100 req/s.
1 GPU at bs = 16, 125 req/s ⇒ $\lfloor \frac{1100}{125} \rfloor = 8$
⇒ 100 req/s "residual"

Greedy Approximation

Decreasing occupancy

- start by giving every residual workload its own GPU
- Merge workloads together ⇒ 2 GPUs → 1 GPU
- Workloads are "squishy"



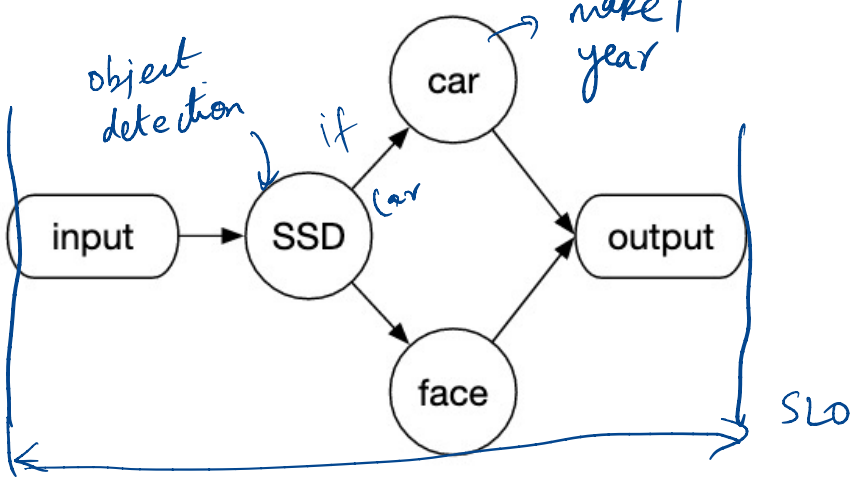
smaller of two duty cycles

HANDLING COMPLEX QUERIES

1 query = 1 DNN model → Simple query

Challenge:

How do we set latency SLOs for complex queries?



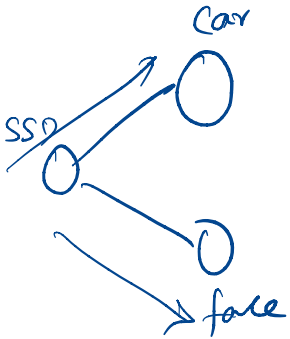
Additional challenge

we don't know what fraction calls "car" and what fraction calls "face"

SCHEDULING COMPLEX QUERIES

Query Analysis to determine latency SLO splits

Inputs: Models with request rate R_i, latency SLO L



$$\underset{\{b_v\}}{\text{minimize}} \quad \sum_v R_v l_v(b_v)/b_v$$

best batch size
minimize resources used

$$\text{subject to} \quad \sum_{u:M_{\text{root}} \rightsquigarrow M_v} l_u(b_u) \le L \qquad \forall v \in \text{leaf}$$

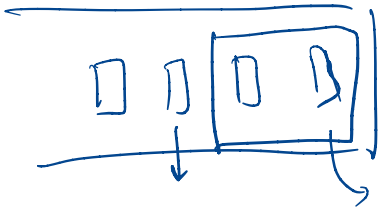
Dynamic Programming

"Pipe dream"

ADAPTIVE BATCHING

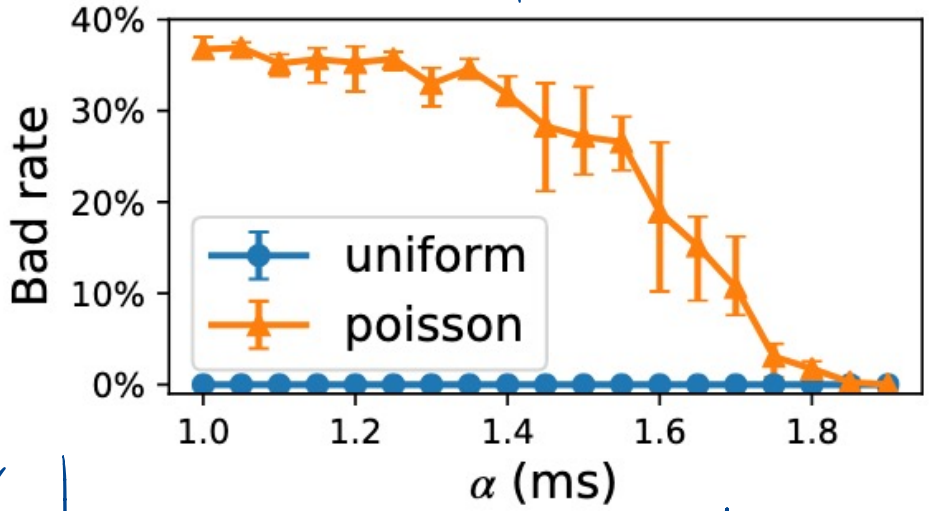
fixed cost per req.
↑
α is low, β is high
then drop rate is high

Clipper: Adapt the batch size based on the oldest request



ts = oldest request needs to be processed in 5ms

lower my batch size in order to meet this SLO



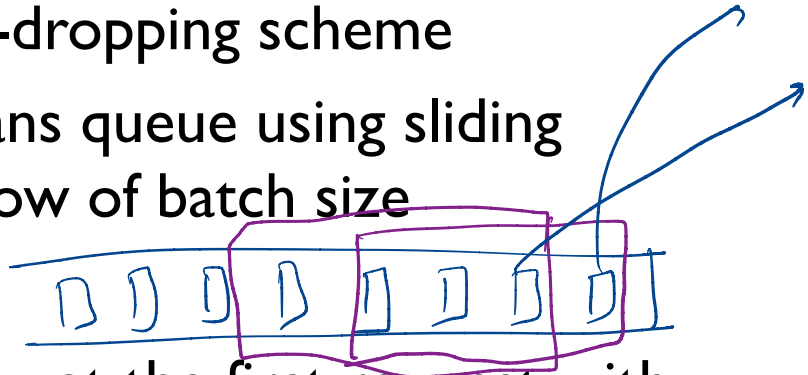
lower batch size, tput is lower

"falling behind"

BATCH-AWARE DISPATCH

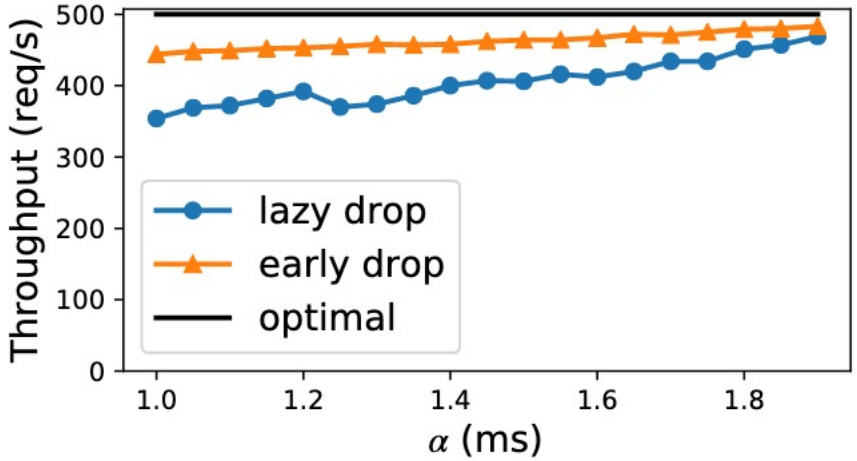
Early-dropping scheme

1. Scans queue using sliding window of batch size



2. Stop at the first request with that can execute *entire window*

if you drop this frame next frame might have similar contents



Drop requests at the head of the queue which cannot meet SLO

OTHER FEATURES

Prefix Batching → Transfer learning → Many models have same "prefix" layers but differ in last layer

Do this by hashing the models

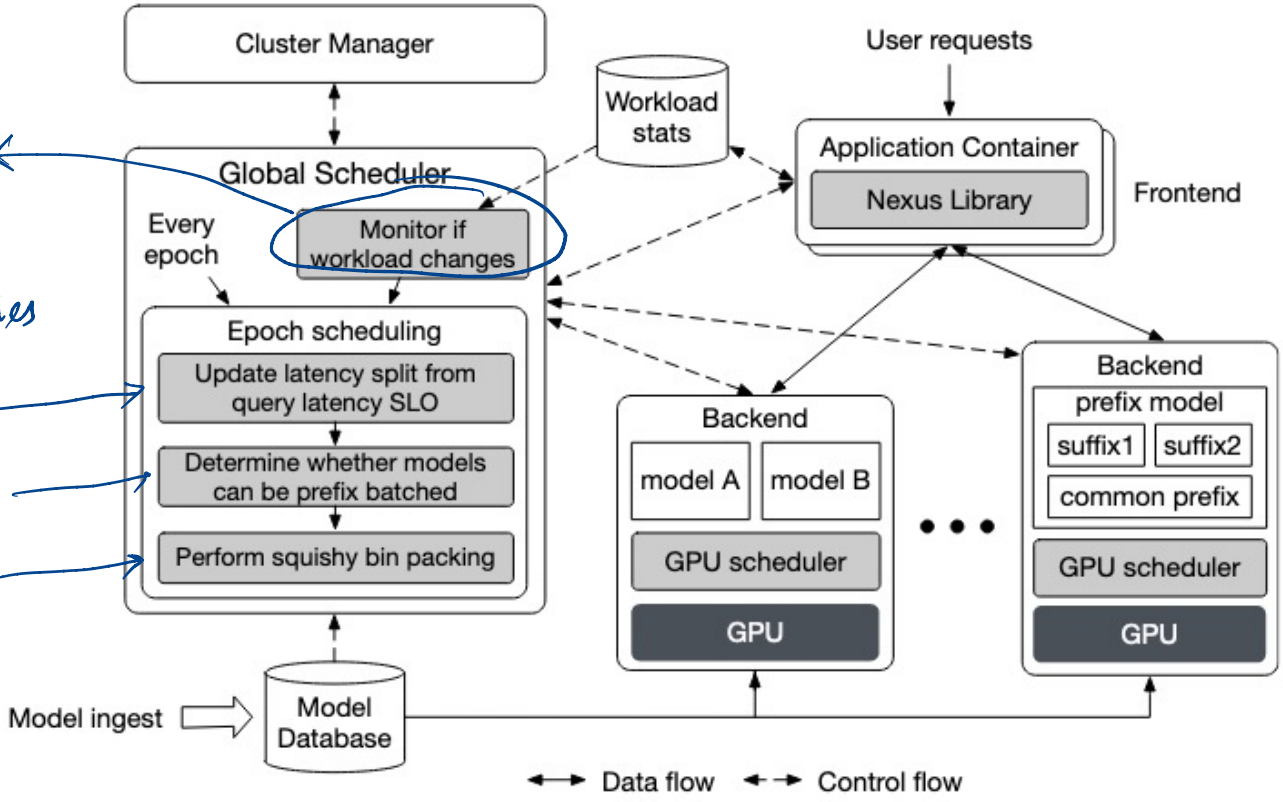
GPU Multiplexing

↳ round robin fashion

Overlapping CPU and GPU computation

↳ Pre-processing & post-processing

NEXUS ARCHITECTURE



Adapting to input changes

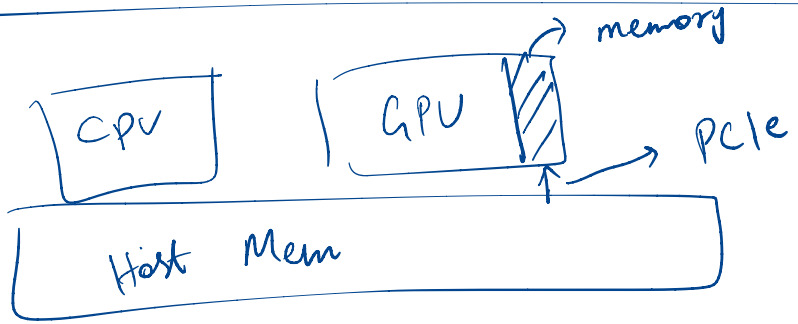
Complex queries

Greedy algorithm

Throughput is related to video streams

SUMMARY

- ML Inference goals: latency SLO, GPU utilization
- Nexus: Handle multiple tenants, multiple DNNs
- Schedule using squishy bin packing
- Breakdown SLO for complex queries, adaptive batching



Pytorch Distributed

DataParallel Training API
Overlap compute, communication

PipeDream

Generalize parallelism: Pipeline parallel
Reduce network, maintain consistency

Ray

Reinforcement learning applications
Actors and tasks, Local and global scheduler

Pollux

Scheduler ML training jobs in a cluster
Co-adaptive scheduling to set batch size, LR

Nexus

System for ML Inference, scheduling
Meet latency SLOs while ensuring high utilization

DISCUSSION

https://forms.gle/XQ4CfNzTTFsSrVv7A

Consider a scenario where you have a model that takes variable amount of time depending on the input. For example if a frame contains 100 cars it takes 250ms to process but if the frame has 1 car then it finishes in 10ms. What could be one shortcoming in using Nexus to schedule this model? SLO = 250ms

When data switches from 1 car to 250 cars

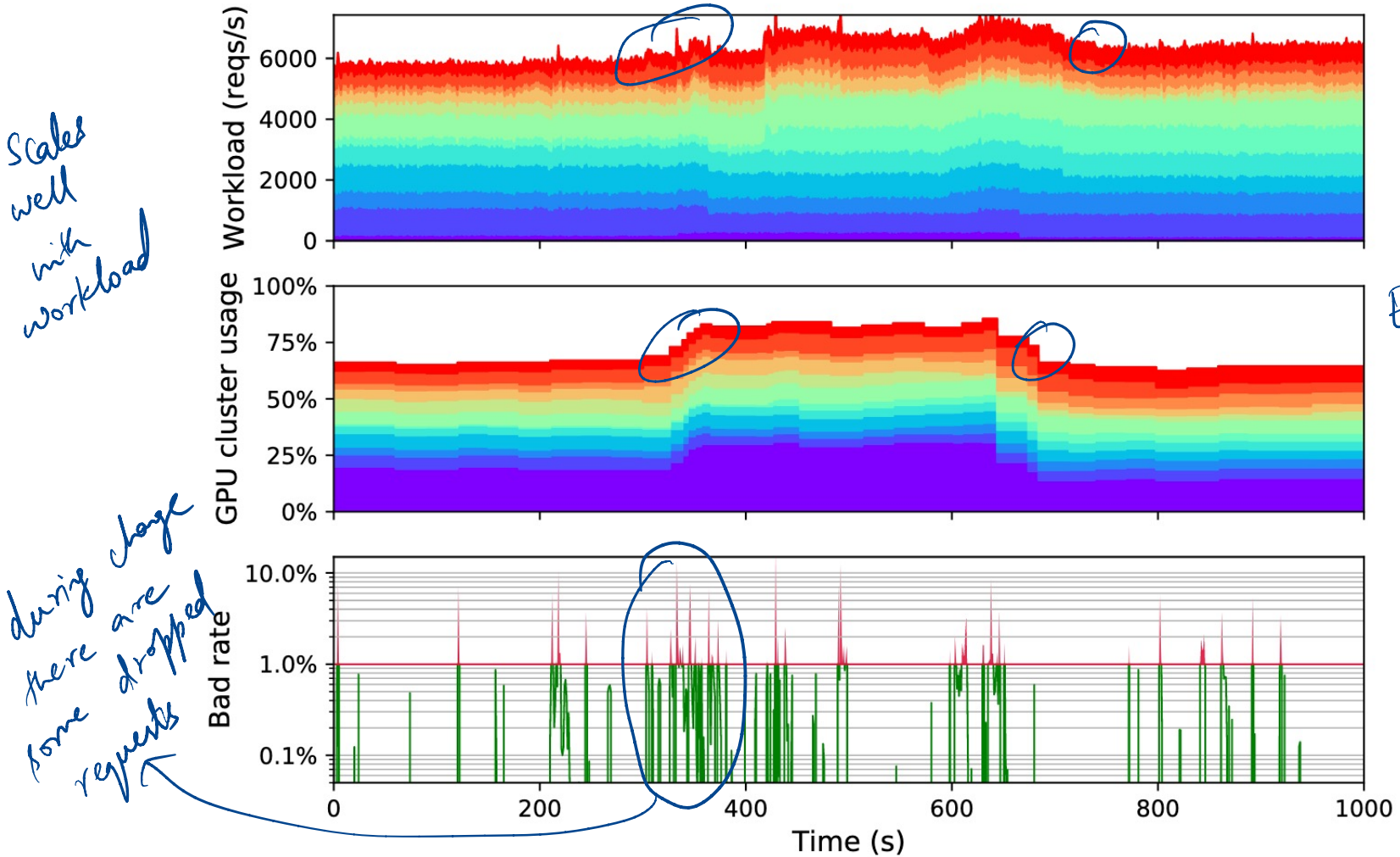
"Profile" → Only saw batches with few cars

estimate batch size based on this

you might miss SLO if using large batch

→ Can Nexus be more "dynamic"?

→ If we take a "worst" case view → under utilization



Scales well with workload

Elasticity

during change there are some dropped requests

Next class: SQL