Hi!

CS 744: NEXUS

Shivaram Venkataraman
Fall 2021
Course Project Proposals
- Due Oct 25!
- 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

Training

Inference

Training Data

Learn

Model

$\hat{y}$

Query

Prediction

Application

Feedback

PyTorch

DDP

Pipe dream
EXAMPLE APPLICATION

Video analysis service
- Thousands of streams, thousands of tenants
- Each stream is processed by a DNN-based “query”
- Latency SLOs (10s to 100s of ms)

Traffic cameras → Stream data to the cloud → Each tenant is running a diff model

Object detection model
find bounding boxes
and classify
“Car” is at

Diagram:
- Video stream → Frame sampling → Lightweight analysis → DNN-based analysis → Aggregation → Video analysis outputs
- Only take 1fps
- Remove frames with no data
GOAL: HIGH GPU UTILIZATION

Placement

$\Rightarrow$ Overhead to moving a model to GPU

$\Rightarrow$ Decide which model(s) go to which GPU

Batching

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

10 frames per sec | $\beta \rightarrow$ fixed overhead

$\Rightarrow$ wait until you can form this batch

$\Rightarrow$ Cost effective when you use an accelerator like a GPU

$\Rightarrow$ Computationally efficient

$\Rightarrow$ Larger batch size
### Scheduling Batched Execution

#### Diagram

- **Model A**
  - Previous batch: 75 ms
  - Current batch: 75 ms
  - Duty cycle: 125 ms

- **Model B**
  - Batch 4: 200 ms

- **Model C**
  - Batch 4: 60 ms

#### Table

<table>
<thead>
<tr>
<th>Model A</th>
<th>Model B</th>
<th>Model C</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Batch</strong></td>
<td><strong>Lat</strong></td>
<td><strong>Req/s</strong></td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>75</td>
<td>107</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>160</td>
</tr>
</tbody>
</table>

#### Residual Workload

- Consider A and B being colocated.
  - A for 75 ms, B for 50 ms, A for 75 ms...
  - Latency (A) + Latency (B) will both meet SLAs.

- A and C cannot be colocated because that will lead to SLA violation.
**BATCH-AWARE SCHEDULING**

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

**Approach:** Allocate “full” GPUs based on load. Handle residuals

If reg. rate = 1100 req/s,

1 GPU at bs = 16, 125 req/s → \[ \frac{1100}{125} = 8 \]

\[ \Rightarrow 100 \text{ reqs} \]

“Residual”

**Greedy Approximation**

- Start by giving every residual workload its own GPU
- Merge workloads together → 2 GPUs
- → 1 GPU

Workloads are “squishy”
HANDLING COMPLEX QUERIES

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 $L$

minimize $\sum_v R_v l_v(b_v)/b_v$
subject to $\sum_u l_u(b_u) \leq L \quad \forall v \in \text{leaf}$

Dynamic Programming
"Pipe dream"
ADAPTIVE BATCHING

Clipper: Adapt the batch size based on the oldest request

$t_8 = \text{oldest request needs to be processed in 5ms}$

Lower my batch size in order to meet this SLO

α is low, β is high, then drop rate is high

Lower batch size, but is falling behind lower

$\alpha$ is low, $\beta$ is high

fixed cost per req.

α (ms) 1.0 1.2 1.4 1.6 1.8

0% 10% 20% 30% 40%

Bad rate

uniform

poisson
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

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

If you drop this frame, the next frame might have similar contents.
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 throughput changes

Complex queries → Greedy algorithm

Throughput is related to video streams

Cluster Manager

Global Scheduler

Every epoch

Monitor if workload changes

Epoch scheduling

Update latency split from query latency SLO

Determine whether models can be prefix batched

Perform squishy bin packing

Workload stats

User requests

Application Container

Nexus Library

Frontend

Backend

prefix model
	suffix1
	suffix2

common prefix

GPU scheduler

GPU

Model Database

Data flow ↔ Control flow
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
<table>
<thead>
<tr>
<th><strong>Pytorch Distributed</strong></th>
<th><strong>PipeDream</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>DataParallel Training API</td>
<td>Generalize parallelism: Pipeline parallel</td>
</tr>
<tr>
<td>Overlap compute, communication</td>
<td>Reduce network, maintain consistency</td>
</tr>
</tbody>
</table>

**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?

When data switches from 1 car to 250 cars

"Profile" → Only saw batches with few cars

"Profile" estimate batch size based on this

you might miss SLO if using large batch

→ Can Nexus be more "dynamic"?

→ Can we take a "worst case" view? under utilization
Scales well with workload during change; there are some dropped requests. Elasticity.
Next class: SQL