CS 744: NEXUS

Shivaram Venkataraman
Fall 2022
ADMINISTRIVIA

Course Project Proposals
- Due Oct 26! → check Canvas
- See Piazza for template

Midterm details → Prev year Questions on Piazza
- Oct 27th: Includes papers from Datacenter as a Computer to Nexus
- Open book, open notes
- Held in class time 9:30–10:45am Central Time
  1 pm – 2:15 pm
MACHINE LEARNING: INFERENCE

Training

Training Data

Learn

$x \xrightarrow{\text{forward}} \hat{y}$

Model

Inference

Query

Classify

Image

Prediction

Cat

Application

Feedback

Adapt
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)

Inference pipeline

Lots of data → arriving continuously
SLO → service-level objectives

Admin/User level decision
input to the system.
SCHEDULING GOAL: HIGH GPU UTILIZATION

Placement
- Which GPUs has which ML model.
- Moving ML model to GPU is expensive.
- GPU memory is limited. Can't store all models.

Batching
- batch_lat(b) = \alpha b + \beta,
  \alpha = 1, \beta = 5
  b = 1 \text{ latency } = 5
  b = 10 \text{ latency } = 15

- Matrix Computations
- Fixed cost
  - Overhead of launching a GPU kernel
  - Cost that grows with batch size

- GPUs are expensive
- 100s of GB
- PCIe
  - 8 GB
  - 16 GB
- GPU

- GPU memory
- 100s GB
Batch size trade-off

\[ b = 1 \rightarrow \text{how latency} \rightarrow \text{lower utilization} \]

\[ b = 128 \rightarrow \text{high utilization} \]

\[ \text{latency SLO miss} \]

\[ \text{to make a batch takes a while} \]

\[ \text{inference of large batch takes longer} \]
SCHEDULING BATCHED EXECUTION

None of the models can saturate CPU on their own when run at the same GPU:
- Model A: \( b=8, \text{lat}=75 \text{ ms} \)
- Model B: \( b=4, \text{lat}=50 \text{ ms} \) → FITS
- Model C: \( b=4, \text{lat}=60 \text{ ms} \) Does not FIT

Target tputs: A: 64, B: 32, C: 32 req/sec. SLO: 250ms

<table>
<thead>
<tr>
<th>Batch</th>
<th>Lat</th>
<th>Req/s</th>
</tr>
</thead>
<tbody>
<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>

<table>
<thead>
<tr>
<th>Batch</th>
<th>Lat</th>
<th>Req/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>50</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>90</td>
<td>89</td>
</tr>
<tr>
<td>16</td>
<td>125</td>
<td>128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Batch</th>
<th>Lat</th>
<th>Req/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>60</td>
<td>66.7</td>
</tr>
<tr>
<td>8</td>
<td>95</td>
<td>84</td>
</tr>
<tr>
<td>16</td>
<td>125</td>
<td>128</td>
</tr>
</tbody>
</table>
Batch-Aware Scheduling

For each model:

- All information from prev slide

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

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

Model A: Rate = 1100 reg/s → 8 full GPUs
Best throughput: b = 16, 125 reg/s

Greedy Approximation

- Find a packing assignment of k residuals
- Merge models to minimize num GPUs
- Merge models to minimize num GPUs

k GPUs to start

Min of two duty cycles

Reduce batch size
Challenge:

How do we set latency SLOs for complex queries?

SSD = 100ms

and

car / face = 150ms

Best assignment depends on model properties
SCHEDULING COMPLEX QUERIES

Query Analysis to determine latency SLO splits
Inputs: Models with request rate $R_i$ latency SLO $L$

\[
\text{minimize } \sum_{\nu} R_{\nu} l_{\nu}(b_{\nu})/b_{\nu} \rightarrow \text{find batch sizes that minimize num GPUs used.}
\]

subject to $\sum_{u: M_{\text{root}} \sim M_{\nu}} l_u(b_u) \leq L \quad \forall \nu \in \text{leaf}$
ADAPTIVE BATCHING

Clipper: Adapt the batch size based on the oldest request in the queue

\[ \text{Queue of request} \rightarrow \text{dropping request that will miss SLO is a good idea} \]

\( t_{\text{arrival}} + \text{SLO} \) get time at which this should be executed

\[ \text{batch latent}(b) = \alpha b + \beta, \]

[Graph showing the relationship between \( \alpha \) and bad rate with 'uniform' and 'poisson' distributions]
Early-dropping scheme
1. Scans queue using sliding window of batch size
2. Stop at the first request with that can execute entire window

Check batch \([1, 2, 3, 4]\) meet SLO
batch \([2, 3, 4, 5]\) meet SLO

Batch size is given

\[b = 4\]

Reduces num of requests that miss SLO

Graph:
- Throughput (req/s)
- \(\alpha (ms)\)
- lazy drop
- early drop
- optimal
OTHER FEATURES

Prefix Batching

GPU Multiplexing

Overlapping CPU and GPU computation
NEXUS ARCHITECTURE

**Control Plane**
- Run periodically (minute)
- Finds num GPUs for each model
- Batch size at each GPU

**Data Plane**
- Req arrives
- Fwded to backend

**Cluster Manager**
- Monitor if workload changes

**Global Scheduler**
- Every epoch
- Update latency split from query latency SLO
- Determine whether models can be prefix batched
- Perform squishy bin packing

**Workload stats**

**Application Container**
- Nexus Library

**Frontend**
- Routing table

**Backend**
- Prefix model
  - suffix1
  - suffix2
  - common prefix
- GPU scheduler

**Model ingest**
- Model Database

**Throughput**

**Data flow**
- Created batches
- Dispatched
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
DISCUSSION

https://forms.gle/PtEaiF4casfZm2jY6
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?

Latency profiling can be hard to do
- Use worst case (~100 cars)
  - lead to underutilization

Profiles
- Dynamic fashion → could pay off if you have patterns
- Linear formula? \( \text{number of cars} \)
  - Pipeline needs a new model to count cars.
At transitions, the bad rate is high.
Next class: SQL
Coming soon
  Project Introductions
  Midterm I