Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis

Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong,
Matthai Philipose, Arvind Krishnamurthy, Ravi Sundaram
Analyze video at large scale

Real-time traffic monitoring

Surveillance

Game stream indexing

Intelligent family camera
Video analysis pipeline

Most computation and cost
DNN serving similar to traditional distributed serving

- Auto scaling
- Load balancing
- Latency constraints
DNN serving imposes additional constraints

1. Use accelerators

Diagram:
- Frontend latency: 100ms
- GPU
- GPU
DNN serving imposes additional constraints

1. Use accelerators
2. Pre-load models
DNN serving imposes additional constraints

- Use accelerators
- Pre-load models
- Batch processing
Existing DNN serving systems are single-app solutions

E.g., Tensorflow Serving, Clipper

- Do not coordinate resource allocations across DNN applications
- Rely on external schedulers that cannot perform cross-app optimizations

How to build a serving system that coordinates the serving of multiple DNN applications?
Optimization opportunities

1. Cluster-level: batch-aware, latency-aware resource allocation across models
2. Application-level: handle complex queries
3. Model-level: batch at sub-model granularity
Opportunity 1: cluster-level resource allocation
Opportunity 1: cluster-level resource allocation

Model 1
SLO: 12
Rate: 1/2

worst latency: 12

duty cycle: 8
Opportunity 1: cluster-level resource allocation

Model 1
SLO: 12
Rate: 1/2

worst latency: 12
duty cycle: 8

Latency SLO limits the batching optimization
Opportunity 1: cluster-level resource allocation

Model 1
- SLO: 12
- Rate: 1/2
- Duty cycle: 8

Model 2
- SLO: 18
- Rate: 1/4
- Duty cycle: 12
Opportunity 1: cluster-level resource allocation

Model 1
SLO: 12
Rate: 1/2
duty cycle: 8

Model 2
SLO: 18
Rate: 1/4
duty cycle: 8

50%

50%
Challenge: GPU sharing has to account for SLO and “squishy” load demands across models
Opportunity 2: app-level complex query

Latency SLO 100ms

Detection Recognition

<table>
<thead>
<tr>
<th>Model Latency SLO (ms)</th>
<th>Throughput (reqs/s/GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection</td>
<td>Recognition</td>
</tr>
<tr>
<td>40</td>
<td>60</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 0.1$</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 1$</td>
</tr>
<tr>
<td></td>
<td>$\alpha = 10$</td>
</tr>
</tbody>
</table>
Opportunity 2: app-level complex query

Latency SLO 100ms

α objects

<table>
<thead>
<tr>
<th>Model Latency SLO (ms)</th>
<th>Throughput (reqs/s/GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection</td>
<td>Recognition</td>
</tr>
<tr>
<td>40</td>
<td>60</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>60</td>
<td>40</td>
</tr>
</tbody>
</table>
Opportunity 2: app-level complex query

Latency SLO 100ms

Model Latency SLO (ms) | Throughput (reqs/s/GPU)
---|---
Detection, Recognition | $\alpha = 0.1$, $\alpha = 1$, $\alpha = 10$
40, 60 | Low, Medium, High

Challenge: Latency split impacts efficiency and needs to be adapted to workload.
Opportunity 3: model-level transfer learning

- Fine-tune a model to a different dataset or task
Opportunity 3: model-level transfer learning

- Fine-tune a model to a different dataset or task

Challenge: How to speed up the common part across models?
Nexus: efficient and scalable DNN execution system on GPU cluster

1. Profiling-based batch-aware resource allocator

2. Query analyzer determines latency split given latency SLO

3. Batch common prefix across models
Nexus: efficient and scalable DNN execution system on GPU cluster

1. Profiling-based batch-aware resource allocator

2. Query analyzer determines latency split given latency SLO

3. Batch common prefix across models
Resource allocation problem

- Bin-packing problem: pack model sessions (model, SLO) to GPUs
- Optimization goal: minimize total number of GPUs
- Constraint: requests need to be served within latency SLOs
- More complex than bin packing due to
  - Change the batch size (squishy tasks)
  - Need to meet latency SLO
Squishy bin-packing algorithm

1. Allocate one GPU for each model session, and choose largest batch size $b_i$ such that $d_i + l_i(b_i) \leq L_i$

2. Merge these nodes into fewer nodes
   Maintain two invariants:
   - Duty cycles will never increase
   - Occupancy of combined nodes $\leq 1$

How to merge two nodes?
Which nodes to merge?
How to merge two nodes?

Node 1

Node 2
How to merge two nodes?

1. Use the minimum duty cycle of two nodes as the new duty cycle, and adjust batch size.

Node 1

Node 2

Node 1

Node 2
How to merge two nodes?

1. Use the minimum duty cycle of two nodes as the new duty cycle, and adjust batch size
2. Valid merge if occupancy of merged node is no more than 1
Which nodes to merge?

• Sort all nodes by its occupancy in decreasing order

• For each node
  • Find a merging that yields highest occupancy
  • Otherwise, add this node in the scheduled nodes
Nexus: efficient and scalable DNN execution system on GPU cluster

1. Profiling-based batch-aware resource allocator

2. Query analyzer determines latency split given latency SLO

3. Batch common prefix across models
Query Analysis

**Given**: query latency SLO $L$, request rate for model $u$ as $R_u$, and max throughput of model $u$ with time budget $t$ as $TP_u(t)$

**Goal**: minimize the total number of GPUs
Query Analysis

**Given**: query latency SLO $L$, request rate for model $u$ as $R_u$, and max throughput of model $u$ with time budget $t$ as $TP_u(t)$

**Goal**: minimize the total number of GPUs

1. Extract the dataflow dependency graph between model invocations
Query Analysis

**Given:** query latency SLO $L$, request rate for model $u$ as $R_u$, and max throughput of model $u$ with time budget $t$ as $TP_u(t)$

**Goal:** minimize the total number of GPUs

2. Use the dynamic programming

Define function $f(u, t)$ as the min #GPUs required to run model $u$ and subtree of $u$ within time budget $t$

$$f(u, t) = \min_{t' \leq t} \left( \frac{R_u}{TP_u(t')} + \sum_{v: M_u \to M_v} f(v, t - t') \right)$$

#GPUs for SSD

result is $f(root, L)$

Traffic app

$SSD$

$RS$

$RF$

$RC$

$t'\to t - t'\to car$

$f(S, t)$

(face, car)
Nexus: efficient and scalable DNN execution system on GPU cluster

1. Profiling-based batch-aware resource allocator

2. Query analyzer determines latency split given latency SLO

3. Batch common prefix across models
Prefix batching for transfer learning

• Compute the hash of sub-tree and detect common sub-trees
Prefix batching for transfer learning

• Compute the hash of sub-tree and detect common sub-trees
• Load common prefix once and different suffixes
Prefix batching for transfer learning

- Compute the hash of sub-tree and detect common sub-trees
- Load common prefix once and different suffixes
- Execute common prefix in a batch of mixed requests and execute different suffixes sequentially
Evaluation

• Baseline: Clipper and Tensorflow Serving

• Both lack support for cluster and complex queries
  
  • Batch-oblivious scheduler
    allocates \# GPUs \propto \text{request rate / max throughput under latency SLO on a single GPU}
  
  • Naive query analysis
    splits query latency SLO evenly to each stage
Case study: game analysis

[Image of a game character with a character box and digits box]

Diagram showing:
- Input
- LeNet
- X6
- Res-Net
- Output

Diagrams of different game scenes
Case study: game analysis

- 20 Games with popularity distribution (Zipf-0.9)
- Specialize ResNet-50 by fine-tuning the last layer for each game
- 16 Nvidia GTX 1080Ti with latency SLO 50ms
Case study: game analysis

- 20 Games with popularity distribution (Zipf-0.9)
- Specialize ResNet-50 by fine-tuning the last layer for each game
- 16 Nvidia GTX 1080Ti with latency SLO 50ms
Case study: game analysis

- 20 Games with popularity distribution (Zipf-0.9)
- Specialize ResNet-50 by fine-tuning the last layer for each game
- 16 Nvidia GTX 1080Ti with latency SLO 50ms
Case study: traffic monitoring

![Image of traffic monitoring with annotations]

- **Input**: images of the scene
- **SSD**: detection model
- **Persons**: detected individuals
- **Cars**: detected vehicles

Diagram:
- **Input** → **SSD** → **Persons** → **Output**
- **Input** → **SSD** → **Cars** → **Output**
Case study: traffic monitoring

Latency SLO: 400ms, 16 Nvidia GTX 1080Ti
Large scale evaluation

- Deploy Nexus on 100 Nvidia K80 GPUs
- Run 7 different applications with changing workload

Adapt GPU allocation
Serve requests within latency SLOs
Conclusion

Nexus serves multiple applications at high utilization on a GPU cluster while satisfying latency SLOs

- Uses squishy bin-packing to schedule DNN workloads
- Analyzes complex queries
- Enables prefix batching across models

Code available at https://github.com/uwsampl/nexus