Summary of “Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis”

Shi Pu (ivanpu), Jianbin Zhang (jianbinz), Wenqi Zhu (zwq)

Problem and Motivation

The authors of this paper believe that existing DNN serving systems(e.g., TF serving and Clipper) have the following problems and downsides:

- They do not utilize resources well for DNN applications.
- They are single application solutions that cannot coordinate resource allocations across multiple DNN applications or perform any cross-DNNs optimizations.
- They limit the granularity of batched execution to the whole model level.

In order to serve multiple DNN applications efficiently and solve the problems stated above, the authors propose **Nexus, a cluster-scale resource management system to effectively distribute the large incoming workload onto the clusters of accelerators.** The system aims at high utilization and high throughput while still meeting latency SLO.

Hypothesis

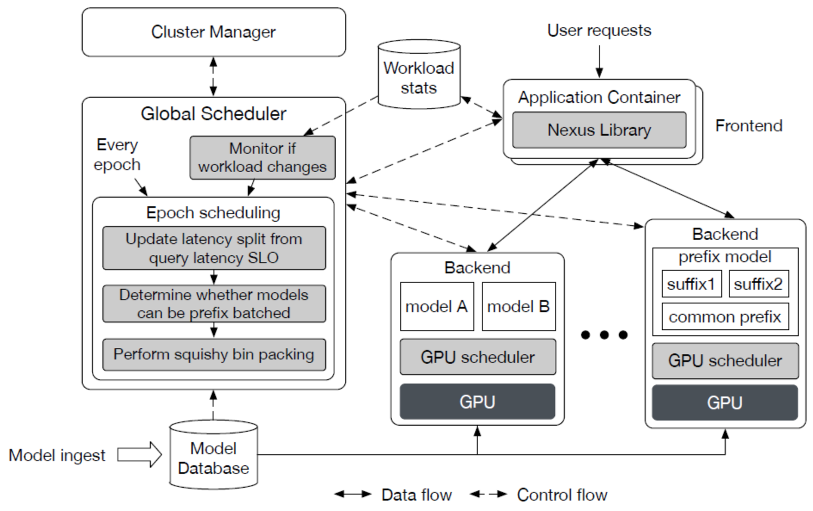
- Most modern DNN models are small enough to coexist within the memory of a single GPU.
- Some applications run a series of DNNs in sequence. Therefore, the scheduler needs to identify the networks within the same application so the user can set a latency requirement on the entire application.
- Transfer learning creates models with a lot of common layers. However, the benefit of batching is lost when training/running these networks because conventional batching only applies to the same models. There exist optimization opportunities since these networks are basically the same except for the last few layers.

Solution Overview

Nexus Architecture

Overview: Nexus consists of three planes: the management plane, the control plane, and the data plane.

- Management plane: In the management plane, the models, once ingested into the system, are stored in a model database. Then, at injection time, Nexus will measure the latency and memory usage of each modal, to derive a batching profile.



- Control plane: The control plane uses the model profile to assign frontends and backends, and to invoke the epoch scheduler to determine the batch size and which models to run.

- Data plane: The data plane consists of the Nexus runtime, which includes the Nexus library and the backend modules. The data plane is responsible for dispatching and executing the actual user requests.

Three main techniques

Nexus contributes three main techniques for optimization used by the global scheduler and the node dispatcher.

- **Cluster-level: A profiling-based batch-aware resource allocator.** The allocator schedules residual loads based on a greedy scheduling algorithm, which is named *squishy bin packing*.
  It first allocates one GPU for each model session S_i, and computes the largest batch size b_i to maximize GPU efficiency while still meeting latency SLO.
  Then it merges multiple sessions within one GPU's duty cycle for higher utilization. To choose a node to merge, it sorts all nodes by its occupancy in decreasing order, and picks the node with least occupancy. It always maintains two invariants during merging: the duty cycles shouldn't increase, and occupancy of combined sessions should not be larger than one GPU's ability. It first uses the minimum duty cycle of two GPU nodes as the new duty cycle and adjusts the batch size. Then it validates the merge by checking if occupancy of merged GPU node is no more than one.

- **Application-level: Scheduling complex queries.** This technique allows users to specify latency SLO at the whole-query level instead of per DNN.
  Nexus performs split latency adaptively over time considering the real time workload. Because latency is determined by batch size, scheduling queries is essentially an optimization problem to find the best batch sizes that minimizes the GPU count while

meeting the latency SLO along the execution path.
Nexus uses dynamic programming to solve the optimization mentioned above in case of fork-join dependency graphs.

- **Model-level: Batch-aware dispatch.** The Nexus backend overlaps the CPU computation, such as pre- and post-processing for the models, within GPU computation, in order to maximize GPU utilization.
  Existing DNN frameworks lack support for concurrent execution of multiple models on a single GPU machine. Nexus fixes this issue by managing the execution of all models on a GPU, allowing it to select batch sizes and execute schedules for all models in a round-robin fashion on the same GPU machine.
  Nexus batches common prefix layers across models, so that it can avoid recomputing the same prefix layers for different models, significantly reducing the runtime needed for scheduled batched DNN tasks. Nexus achieves this by computing the hash of every sub-tree to detect common sub-trees.
  Nexus skips over requests that would cause sub-optimal batching by adapting the early drop policy, dropping requests if the system detects that there is not enough budget for this task.

Limitations and Possible Improvements

- The squishy bin packing technique relies on the assumption that most DNN models are small so multiple models can fit in one GPU. However, this is not always the case.
- The common prefix batching technique assumes a scenario of transfer learning. It doesn't work for the case where the models are widely different from each other. How to apply batching to completely different models remains an interesting question and we think there is room for improvement.

Summary of Class Discussion

- Q: Are users allowed to specify the batch size for their DNN tasks? For the bin packing process, they need to compute the largest batch size to maximize GPU efficiency with reasonable latency. Then if they allow users to specify batch size, how are they choosing between the user-defined batch size and the batch size they compute?
  A: No, the batch profiles are some helpful metadata, which helps the control plane effectively schedule and batch the model.
- Q: Is it possible that this centralized data plane becomes the bottleneck? Is it possible to move some scheduling to the individual machines so the central scheduler doesn't need to do micromanagement.
  A: If you move scheduling onto back-end that makes more locally optimal decisions, you might violate some global constraints.
- Q: How common is it for two random models to have the same prefix?
  A: The paper didn't give us any specific examples. It did mention transfer learning

where only the last few layers are retrained for the new task. The rest of the layers are pretty similar.

- Q: I think the prefix batching assumes that the models are at least identical for the majority part. Did they provide any solution or discussion to the situation where the models are completely different in the paper?
  A: Prefix batching is only an optimization that we can do. It doesn't mean we can't handle models that are different from each other.

- Q: Is it true prefix batching is only done on the same machine?
  A: In terms of Nexus, it only performs the common prefix on the same GPU.

- Q: Do GPUs run faster running the same code for all inputs?
  A: A GPU only guarantees much higher throughput compared to CPU. I'm not too sure about how fast GPU runs things. Individual GPU cores tend to be weak.

- Q: Are users allowed to specify the batch size for their DNN tasks? For the bin packing process, they need to compute the largest batch size to maximize GPU efficiency with reasonable latency. Then if they allow users to specify batch size, how are they choosing between the user-defined batch size and the batch size they compute?
  A: When you upload the model you also upload a batch profile that contains latency of different batch sizes. We are picking the optimal batch size using that profile data.

- Q: So this squishy bin packing assumes that multiple models can fit in one GPU right? Then how about the large model that might need several GPUs?
  A: Get a bigger GPU :) Maybe do model partition and split the model into different layers and put those onto different machines.

- Q: For DNN execution, it seems like caching is extremely beneficial. Why or why not are people considering doing this?
  A: During research, you want to highlight the new things that you are doing. Adding caching won't be as interesting because other people have already done it. They might have already implemented it but chose not to mention it in the paper.

Summary of “Focus: Querying Large Video Datasets with Low Latency and Low Cost”

Shi Pu (ivanpu), Jianbin Zhang (jianbinz), Wenqi Zhu (zwq)

Problem and Motivation

Querying objects in videos requires running both detector and classifier CNNs, which can be very slow and costly on massive video datasets. Analyzing live videos at ingest time can speed up the query process, but it is expensive. Analyzing videos at query time avoids spending a lot of time at the ingest phase, but since it requires expensive CNN, it incurs high latency. To **enable both low-latency and low-cost object querying** over large historical video datasets, the authors propose a video querying system called Focus. The problems Focus solves include:

- Allows low latency and low cost for object querying without sacrificing recall and precision a lot.
- Allows trade-offs between ingest cost and query latency.

Hypothesis

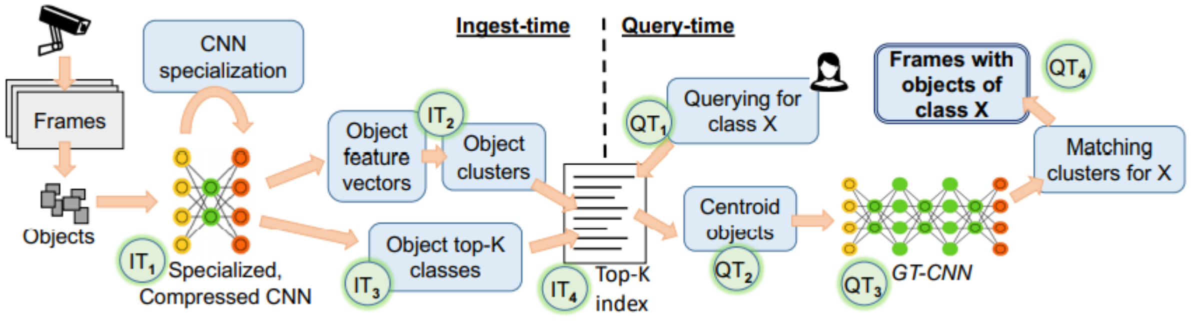
This paper makes a few assumption on real-world video datasets:

- A significant portion of videos is of interest and can be excluded. The author claims that in their video sets, an object class occurs in only 0.16% of the frames on average. Even the most frequent classes only appear in no more than 26% - 78% of the frames. Moreover, the infrequent classes are usually less interesting thus less queried.
- Only a limited set of object classes appear in each video because most videos have different contexts. 2% - 10% of the most frequent object classes cover more than 95% of the objects in all streams.
- Objects within the same class tend to have similar feature vectors. An object often stays in the frame a few seconds, which creates multiple images. It is beneficial to group these similar images together and only classify one of them for a common label.

Solution Overview

Focus aims to process live video streams, and allows the “after-the-fact” queries later to identify frames of a certain class or object. To achieve this objective, there are mainly two parts to the architecture of Focus: ingest-time and query-time.

- At ingest-time, as shown in the graph, Focus utilizes a specialized and compressed “cheap” CNN with fewer convolutional layers (IT_1) to process incoming live video streams. From the video frames, Focus classifies objects and produces feature vectors



from them, which are then used to assign the objects into different clusters (IT_2). For each cluster, Focus produces the top K possible classification classes (IT_3). With the information from the previous steps $IT_2 + IT_3$, Focus then produces top K ingest index, which is a mapping between the classes and the clusters. This index is the final output of the ingest-time process and is then stored for later use in query-time.

- At query-time, when Focus receives a user query request for a specific class X (QT_1), it utilizes the stored mapping in step IT_4 and retrieves the centroid of the matching cluster (QT_2). Focus then runs the expensive GT-CNN as ground truth on these extracted centroids of matching clusters (QT_3). Then, Focus returns all video frames from these clusters whose centroids are classified as object class X back to the user (QT_4).

Limitations and Possible Improvements

- The paper doesn't state how they compress the input GT-CNN to generate ingest-time specialized CNN. We think it would be helpful for the readers to understand Focus better by providing more details in the compression process.

- The paper assumes that the GT-CNN is a high-quality network with high precision and recall. But if the GT-CNN is not that strong, the compressed GT-CNN will perform even worse and generate "wrong" ingest-time output. A possible improvement is that the ingest-time analysis could add a cascading feature, that cascades several cheap CNNs to one. Cascaded cheap CNNs typically have better performance than one cheap CNN.

Summary of Class Discussion

- Q: Do we assume we know what kind of objects that we are looking for?
  A: Focus doesn't make assumptions on what kind of queries that users request.

- Q: Where are the videos stored at? Are they stored in the camera or uploaded to the cloud?
  A: I think they send the videos to some computation places like clouds to process the videos.

- Q: How are the videos processed? Is it based on the individual images or the movement and the changes between the frames?
  A: Individual frames.

- Q: Does ingest time happen continuously or is it more of an offline operation?
  A: I think it does it continuously because the examples that they give include camera on the road, which records continuously.

- Q: What is the use case for Focus? Does it work on live videos?
  A: Often used on videos that you don't necessary need to process live but need fast query later on.

- Q: Is evaluation done on popular objects or both popular and unpopular object?
  A: Both popular and unpopular objects.

- Q: How is k "intelligently" decided by the system?
  A: k is decided at ingest time. Focus runs some different combinations of parameter, compute their precision and recall to decided whether it meets the target.

- Q: Focus's result can't directly compare with other systems because it has a lot of requirement on the data set.
  A: You can't directly compare result from one paper to another paper. It's usually hard to tell which one is better or worse since there could be errors like implementation error. You should always take the numbers in system papers with a grain of salt.