MAXIMIZING UTILIZATION FOR DATA CENTER INference WITH TENSORRT INference SERVER

정소영 상무 (soyoungj@nvidia.com) / 2019년 7월 2일
TENSORRT HYPERSCALE INFEERENCE PLATFORM

WORLD’S MOST ADVANCED SCALE-OUT GPU

INTEGRATED INTO TENSORFLOW & ONNX SUPPORT

TENSORRT INFEERENCE SERVER
TENSORRT INFERENCe SERVER

A Software Application for Deploying AI Models At Scale

- Maximum GPU Utilization
- Mechanisms for Large-Scale Inference Service
- Optimized for Management & Monitoring
- GitHub: [https://github.com/NVIDIA/tensorrt-inference-server](https://github.com/NVIDIA/tensorrt-inference-server)
TENSORRT INFERENCE SERVER
Architected for Maximum Datacenter Utilization

Support a variety of model frameworks
  TensorRT, TensorFlow, Caffe2, custom

Support concurrent model execution, one or multiple models
  Multi-model, multi-GPU and asynchronous HTTP and GRPC request handling

Support many model types: CNN, RNN, “stateless”, “stateful”
  Multiple scheduling and batching algorithms

Enable both “online” and “offline” inference use cases
  Batch 1, batch n, dynamic batching

Enable scalable, reliable deployment
  Prometheus metrics, live/ready endpoints, Kubernetes integration
Extensible backend architecture allows multiple framework and custom support

Extensible scheduler architecture allows support for different model types and different batching strategies

Leverage CUDA to support model concurrency and multi-GPU
# Available Metrics

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Use Case</th>
<th>Granularity</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPU Utilization</strong></td>
<td>Power usage</td>
<td>Proxy for load on the GPU</td>
<td>Per GPU</td>
<td>Per second</td>
</tr>
<tr>
<td></td>
<td>Power limit</td>
<td>Maximum GPU power limit</td>
<td>Per GPU</td>
<td>Per second</td>
</tr>
<tr>
<td></td>
<td>GPU utilization</td>
<td>GPU utilization rate [0.0 - 1.0]</td>
<td>Per GPU</td>
<td>Per second</td>
</tr>
<tr>
<td><strong>GPU Memory</strong></td>
<td>GPU Total Memory</td>
<td>Total GPU memory, in bytes</td>
<td>Per GPU</td>
<td>Per second</td>
</tr>
<tr>
<td></td>
<td>GPU Used Memory</td>
<td>Used GPU memory, in bytes</td>
<td>Per GPU</td>
<td>Per second</td>
</tr>
<tr>
<td><strong>Count</strong></td>
<td>Request count</td>
<td>Number of inference requests</td>
<td>Per model</td>
<td>Per request</td>
</tr>
<tr>
<td><strong>GPU &amp; CPU</strong></td>
<td>Execution count</td>
<td>Number of model inference executions Request count / execution count = avg dynamic request batching</td>
<td>Per model</td>
<td>Per request</td>
</tr>
<tr>
<td></td>
<td>Inference count</td>
<td>Number of inferences performed (one request counts as “batch size” inferences)</td>
<td>Per model</td>
<td>Per request</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>Latency: request time</td>
<td>End-to-end inference request handling time</td>
<td>Per model</td>
<td>Per request</td>
</tr>
<tr>
<td></td>
<td>Latency: compute time</td>
<td>Time a request spends executing the inference model (in the appropriate framework)</td>
<td>Per model</td>
<td>Per request</td>
</tr>
<tr>
<td></td>
<td>Latency: queue time</td>
<td>Time a request spends waiting in the queue before being executed</td>
<td>Per model</td>
<td>Per request</td>
</tr>
</tbody>
</table>
MODEL REPOSITORY

File-system based repository of the models loaded and served by the inference server

Model metadata describes framework, scheduling, batching, concurrency and other aspects of each model

ModelX
  platform: TensorRT
  scheduler: default
  concurrency: ...

ModelY
  platform: TensorRT
  scheduler: dynamic-batcher
  concurrency: ...

ModelZ
  platform: TensorFlow
  scheduler: sequence-batcher
  concurrency: ...
Backend acts as interface between inference requests and a standard or custom framework

Supported standard frameworks: TensorRT, TensorFlow, Caffe2

Providers efficiently communicate inference request inputs and outputs (HTTP or GRPC)

Efficient data movement, no additional copies
CUSTOM FRAMEWORK
Integrate Custom Logic Into Inference Server

Provide implementation of your “framework”/”runtime” as shared library

Implement simple API: Initialize, Finalize, Execute

All inference server features are available: multi-model, multi-GPU, concurrent execution, scheduling and batching algorithms, etc.
MULTIPLE MODELS

ModelZ Backend

- Sequence Batcher
- TensorFlow Runtime

ModelY Backend

- Dynamic Batcher
- TensorRT Runtime

ModelX Backend

- Default Scheduler
- TensorRT Runtime

ModelZ Inference Request

ModelY Inference Request

ModelX Inference Request
MODEL CONCURRENCY
Multiple Models Sharing a GPU

By default each model gets one *instance* on each available GPU (or 1 CPU instance if no GPUs)

Each instance has an *execution context* that encapsulates the state needed by the runtime to execute the model
MODEL CONCURRENCY

Multiple Instances of the Same Model

Model metadata allows multiple instances to be configured for each model

Multiple model instances allow multiple inference requests to be executed simultaneously
MODEL CONCURRENCY
Multiple Instances of Multiple Models

ModelZ Backend
TensorFlow Runtime
Context

ModelY Backend
TensorRT Runtime
Context

ModelX Backend
Default Scheduler
TensorRT Runtime
Context
Context
Context

GPU
CONCURRENT EXECUTION TIMELINE
GPU Activity Over Time

Incoming Inference Requests

ModelX  ModelX  ModelX  ModelY  ModelY  ModelY
CONCURRENT EXECUTION TIMELINE

GPU Activity Over Time

Incoming Inference Requests

Execute ModelX
CONCURRENT EXECUTION TIMELINE
GPU Activity Over Time

Execute ModelX

Incoming Inference Requests

ModelX  ModelX  ModelY  ModelY  ModelY  Time
CONCURRENT EXECUTION TIMELINE

GPU Activity Over Time

Incoming Inference Requests

Execute ModelX

Execute ModelX

Execute ModelX

Execute ModelX

Time
CONCURRENT EXECUTION TIMELINE

GPU Activity Over Time

Incoming Inference Requests

Execute ModelX

Execute ModelX

Execute ModelX

Execute ModelY

Time
CONCURRENT EXECUTION TIMELINE

GPU Activity Over Time

Incoming Inference Requests

Time

Execute ModelX
Execute ModelX
Execute ModelX
Execute ModelY
Execute ModelY
CONCURRENT EXECUTION TIMELINE
GPU Activity Over Time

Incoming Inference Requests

Execute ModelX
Execute ModelX
Execute ModelX
Execute ModelY
Execute ModelY
Execute ModelY

Time
SHARING A GPU
CUDA Enables Multiple Model Execution on a GPU
MULI-GPU
Execution Contexts Can Target Multiple GPUs

ModelY Backend
- Dynamic Batcher

ModelX Backend
- Default Scheduler

TensorRT Runtime
- Context
- Context

CUDA Streams
- CUDA Streams

GPU
Hardware Scheduler
**BATCHING VS NON-BATCHING**

**Batching: Grouping Inference Requests Together**

Batch size = 1
- Run a single inference task on a GPU
- Low-latency, but the GPU is underutilized

Batch size = N
- Group inference instances together
- High throughput and GPU utilization
- Allows employing Tensor Cores in Volta and Turing
SCHEDULER ARCHITECTURE

Scheduler responsible for managing all inference requests to a given model

Distribute requests to the available execution contexts

Each model can configure the type of scheduler appropriate for the model
DEFAULT SCHEDULER
Distribute Individual Requests Across Available Contexts
DEFAULT SCHEDULER
Distribute Individual Requests Across Available Contexts

Incoming requests to ModelX queued in scheduler
DEFAULT SCHEDULER
Distribute Individual Requests Across Available Contexts

Assuming GPU is fully utilized by executing 2 batch-4 inferences at the same time.

Utilization = 3/8 = 37.5%
DEFAULT SCHEDULER
Distribute Individual Requests Across Available Contexts

Assuming GPU is fully utilized by executing 2 batch-4 inferences at the same time.

Utilization = $\frac{2}{8} = 25\%$
DEFAULT SCHEDULER
Distribute Individual Requests Across Available Contexts

Assuming GPU is fully utilized by executing 2 batch-4 inferences at the same time.

Utilization = 4/8 = 50%
DYNAMIC BATCHING SCHEDULER

Group Requests To Form Larger Batches, Increase GPU Utilization

Default scheduler takes advantage of multiple model instances

But GPU utilization dependent on the batch-size of the inference request

Batching is often on of the best ways to increase GPU utilization

Dynamic batch scheduler (aka dynamic batcher) forms larger batches by combining multiple inference request
DYNAMIC BATCHING SCHEDULER
Group Requests To Form Larger Batches, Increase GPU Utilization
DYNAMIC BATCHING SCHEDULER
Group Requests To Form Larger Batches, Increase GPU Utilization
Dynamic batching scheduler
Group Requests To Form Larger Batches, Increase GPU Utilization

Dynamic batcher configuration for ModelY can specify preferred batch-size. Assume 4 gives best utilization.

Dynamic batcher groups requests to give 100% utilization.
Default and dynamic-batching schedulers work with stateless models; each request is scheduled and executed independently.

Some models are stateful, a sequence of inference requests must be routed to the same model instance.

“Online” ASR, TTS, and similar models

Models that use LSTM, GRU, etc. to maintain state across inference requests

Multi-instance and batching required by these models to maximum GPU utilization

Sequence-batching scheduler provides dynamically batching for stateful models.
SEQUENCE BATCHING SCHEDULER
Dynamic Batching for Stateful Models

Sequence: 3 inference requests

Sequence: 5 inference requests
SEQUENCE BATCHING SCHEDULER
Dynamic Batching for Stateful Models

Inference requests arrive in arbitrary order
SEQUENCE BATCHING SCHEDULER
Dynamic Batching for Stateful Models

Sequence batcher allocates context slot to sequence and routes all requests to that slot.

Context has available slots, not used waiting requests due to stateful model requirement.
SEQUENCE BATCHING SCHEDULER
Dynamic Batching for Stateful Models

Sequence batcher allocates context slot to sequence and routes all requests to that slot
SEQUENCE BATCHING SCHEDULER
Dynamic Batching for Stateful Models

ModelZ Backend

Sequence Batcher

Runtime

Context

Context

Sequence batcher allocates context slot to sequence and routes all requests to that slot
SEQUENCE BATCHING SCHEDULER
Dynamic Batching for Stateful Models

On a fully-loaded server, all context slots would be occupied by sequences.

As soon as one sequence ends another is allocated to the slot.
ENSEMBLE MODELS
A way of pipelining models in TRTIS
Recap

**Concurrent Model Execution**
Multiple models (or multiple instances of same model) may execute on GPU simultaneously.

**CPU Model Inference Execution**
Framework native models can execute inference requests on the CPU.

**Metrics**
Utilization, count, and latency.

**Custom Backend**
Custom backend allows the user more flexibility by providing their own implementation of an execution engine through the use of a shared library.

**Stateless / Stateful Inference**
Supports many model types including CNN, RNN, etc.

**Dynamic Batching**
Inference requests can be batched up by the inference server to 1) the model-allowed maximum or 2) the user-defined latency SLA.

**Multiple Model Format Support**
TensorFlow GraphDef/SavedModel
TensorFlow and TensorRT GraphDef
TensorRT Plans
Caffe2 NetDef (ONNX import path)

**Ensemble Model Support**
An Ensemble represents a pipeline of one or more models and the connection of input and output tensors between those models.

**Multi-GPU support**
The server can distribute inferencing across all system GPUs.