AMPERE® TensorFlow Serving Documentation

v.1.4

AMPERE COMPUTING

Contents

TENSORFLOW SERVING

TensorFlow Serving is a flexible, high-performance serving system for machine learning models, designed for production environments. TensorFlow Serving makes it easy to deploy new algorithms and experiments, while keeping the same server architecture and APIs. TensorFlow Serving provides out-of-the-box integration with TensorFlow models but can be easily extended to serve other types of models and data.

RUNNING TENSORFLOW SERVING ON AIO ACCELERATED DOCKER

TensorFlow-serving works with Ampere Optimized TensorFlow. The first step is to get Optimized TensorFlow docker image. To pull Docker image from Docker Hub repository:

```
# docker pull amperecomputingai/tensorflow:1.7.0
```

Launching the Docker to use TensorFlow Serving

To launch docker image execute the following command:

```
# docker run --privileged=true --cap-add=SYS_PTRACE \
        --security-opt seccomp=unconfined \
        --security-opt apparmor=unconfined \
        -it -v/<location of models to be served>:/models \
        -p 8500:8500 -p 8501:8501 --ipc=host \
        --rm <REPOSITORY:TAG of the docker image>
```

Note: 8500 (gRPS) and 8501 (REST) ports are default ports to access the service. If the user wants different ports, modify these in service launch command line as well as in port mapping while starting docker instance.

TensorFlow Serving is distributed as Debian package and is hosted on AWS. Packages are provided to match Ampere optimized TensorFlow versions.

TensorFlow version	Package location
AIO TensorFlow v*1*.3.0 (TensorFlow v2.7.1)	https://ampereaidevelop.s3.eu-central-1.amazonaws.com/tensorflow-model-server_2.7.1-1.3.0_arm64.deb
AIO TensorFlow v1.4.0 (TensorFlow v2.9.2)	https://ampereaidevelop.s3.eu-central-1.amazonaws.com/tensorflow-model-server_2.9.2-1.4.0_arm64.deb
AIO TensorFlow v1.5.0 (TensorFlow v2.9.2)	https://ampereaidevelop.s3.eu-central-1.amazonaws.com/tensorflow-model-server_2.9.2-1.5.0_arm64.deb
AIO TensorFlow v1.6.0 (TensorFlow v2.11.0)	https://ampereaidevelopus.s3.amazonaws.com/releases/1.6.0/tensorflow-model-server_2.11.0-1.6.0_arm64.deb
AIO TensorFlow v1.7.0 (TensorFlow v2.11.0)	https://ampereaidevelopus.s3.amazonaws.com/releases/1.7.0/tensorflow-model-server_2.11.0-1.7.0_arm64.deb

Command line to get and install v2.7.1

```
# wget https://ampereaidevelop.s3.eu-central-1.amazonaws.com/tensorflow-model-server_2.7.1-1.3.0_arm64.deb
# dpkg -i tensorflow-model-server_2.7.1-1.3.0_arm64.deb
```

To preserve installation use docker commit <container id> (“docker ps” command displays container id).

PREPARE MODELS FOR TENSORFLOW SERVING

To use frozen model with TensorFlow Serving, model must first be converted to “saved model” format. Following link has scripts to convert model for:

ResNet50: resnet50v1_5

InceptionV3: inceptionv3

These scripts can be modified to adapt to other models.

Example for ResNet50 model:

```
# cd /tmp
# wget https://ampereaimodelzoo.s3.eu-central-1.amazonaws.com/resnet_50_v15_tf_fp32.pb
# python model_graph_to_saved_model.py --import_path resnet_50_v15_tf_fp32.pb
# cp -r /tmp/1 <location of models to be served>/resnet50/1
```

TensorFlow Serving can serve multiple models at the same time. To achieve this the models must be converted into “saved model” format and saved in location which docker will access through /model.

LAUNCHING SERVICE

After starting docker instance of accelerated TensorFlow Serving, TensorFlow Serving can be launched using the following command line:

Setup environment:

```
export AIO_PROCESS_MODE=1
export AIO_CPU_BIND=1
export AIO_MEM_BIND=1
export AIO_NUM_THREADS=16
export OMP_NUM_THREADS=16
export AIO_NUMA_CPUS=$(echo {0..15})
```

This environment setting uses 16 CPUs. It can be modified by setting NUM THREADS and NUM and NUMA CPUS to different number(s).

Starting service with a single model

The following command starts service using single model (ResNet50 is used as an example):

```
tensorflow_model_server --rest_api_port=8501 \
                    --model_name="resnet50" \
                    --num_load_threads=16 \
                    --tensorflow_intra_op_parallelism=16 \
                    --tensorflow_inter_op_parallelism=1 \
                    --model_base_path="/models/resnet50/"
```

Starting service with multiple models

To start the service with multiple models, first configuration file for these models must be created. Typical contents of model's configuration file:

```
model_config_list: {
  config: {
    name: "resnet50",
    base_path: "/models/resnet50/",
    model_platform: "tensorflow"
  },
  config: {
    name: "inceptionv3",
    base_path: "/models/InceptionV3/",
    model_platform: "tensorflow"
  },
}
```

Command line to start service:

```
tensorflow_model_server --rest_api_port=8501 \
                    --model_name="resnet50" \
                    --num_load_threads=16 \
                    --tensorflow_intra_op_parallelism=16 \
                    --tensorflow_inter_op_parallelism=1 \
                    --model_config_file=/models/models.config
```

TENSORFLOW SERVING CLIENT

TensorFlow Saving can be accessed using two different types of clients. gRPC and REST API's. This section covers both client applications.

Base architecture of client application

To use TensorFlow Service client application follow these steps:

1. Read image(s) for classification.
2. Convert it so the target model can accept it as an input. Conversion must follow the same steps as done at the time of training of the model.
3. Make batch of image(s). Even in the case of a single image it must be created as batch of one. E.g., if image is of dimension (224,224,3) it should b passed as (1,224,224,3)
4. Send converted images to TensorFlow serving using gRPS or REST (explained in net section).
5. Weight for the response
6. Decode the response as per model's output layer (e.g., SoftMax) to get the results.

REST Client

REST client is simple and doesn’t need to have TensorFlow on client-side system.

Configuration

```
HOST='IP Address of HOST system'
PORT='8501'
SERVER_URL = 'http://'+HOST+':'+PORT+'/v1/resnet50:predict'
```

Request and Response

```
#Read image from disk and preprocess it as per target models
image = pre_process_image_asper_model(image)
#Convert it into NHWC format (multiple images can be added into list)
image=np.stack([image], axis=0)
#Create request
predict_request = json.dumps({'instances': image.tolist()})
response = requests.post(SERVER_URL, data=predict_request)
response.raise_for_status()
response = response.json()['predictions']
response = np.array(response)
```

Parse Response as per models output layer (e.g., SoftMax)

gRPC Client

gRPC client is high performance compared to REST client. It needs TensorFlow package on the client side to create request to server. Along with TensorFlow users need to install Python bindings for TensorFlow serving apis “pip install tensorflow-serving-api”.

Configuration

```
HOST='IP Address of HOST system'
PORT='8501'
```

Request and Response

```
#Read image from disk and preprocess it as per target models
image = pre_process_image_asper_model(image)
#Convert it into NHWC format (multiple images can be added into list)
image=np.stack([image], axis=0)
#Create request
channel = grpc.insecure_channel(HOST+':'+PORT)
stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
request = predict_pb2.PredictRequest()
request.model_spec.name = 'resnet50'
request.model_spec.signature_name = 'serving_default'

request.inputs['input_tensor'].CopyFrom(tf.make_tensor_proto(img, shape=img.shape))
result_future=stub.Predict.future(request, 500.0)

result = result_future.result()
dims   = result.outputs["softmax_tensor"].tensor_shape.dim
shape  = tuple(d.size for d in dims)
```

```
resp   = np.reshape(result.outputs["softmax_tensor"].float_val, shape)
```

Parse Response as per models output layer (e.g., SoftMax).

Complete source code of REST and gRPC client for ResNet50 model can be found at: https://github.com/AmpereComputingAI/tensorflow_serving_client