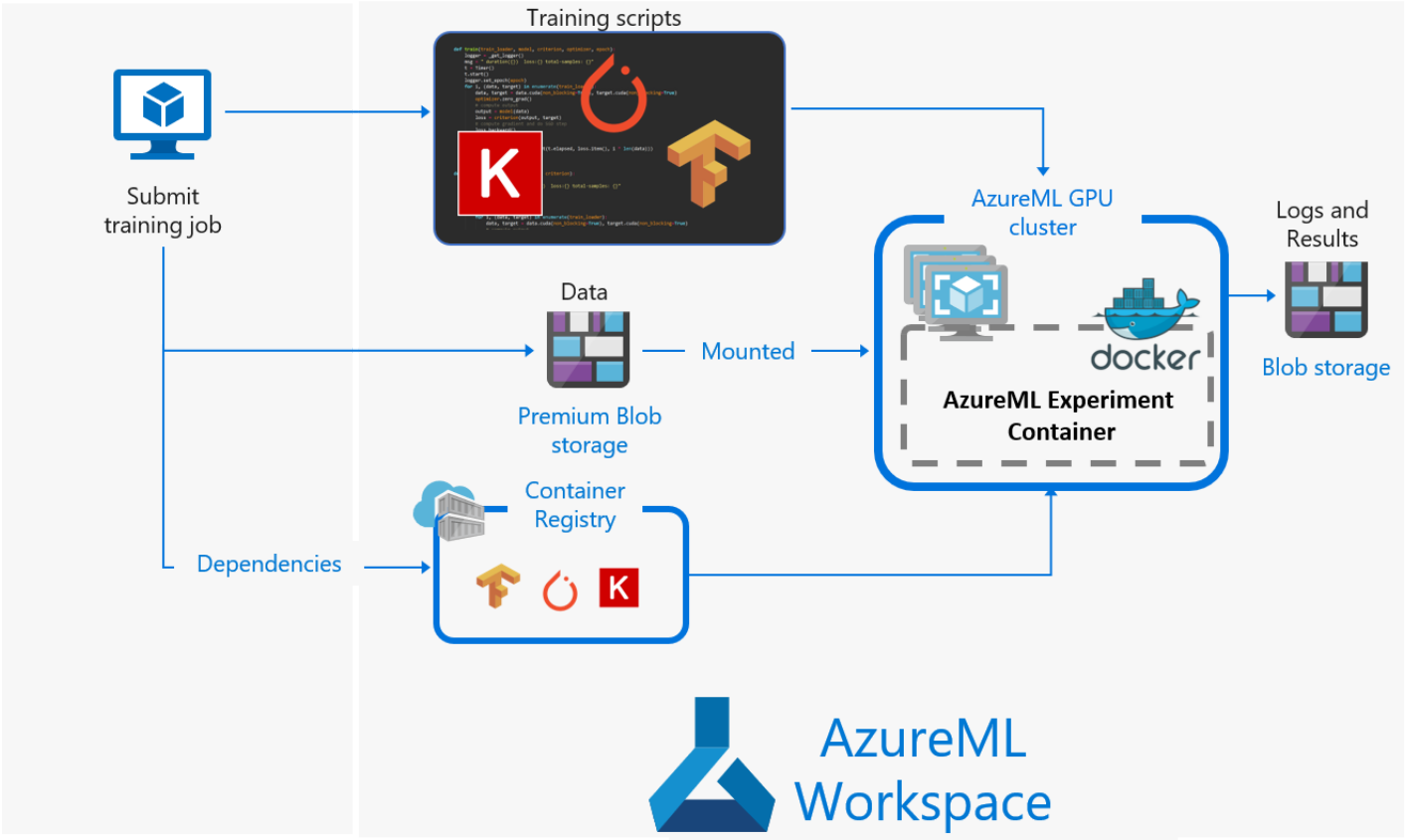
Distributed training of deep learning models on Azure

06/05/2019 • 7 minutes to read • Contributors

In this article

This reference architecture shows how to conduct distributed training of deep learning models across clusters of GPU-enabled VMs. The scenario is image classification, but the solution can be generalized for other deep learning scenarios such as segmentation and object detection.

A reference implementation for this architecture is available on GitHub.

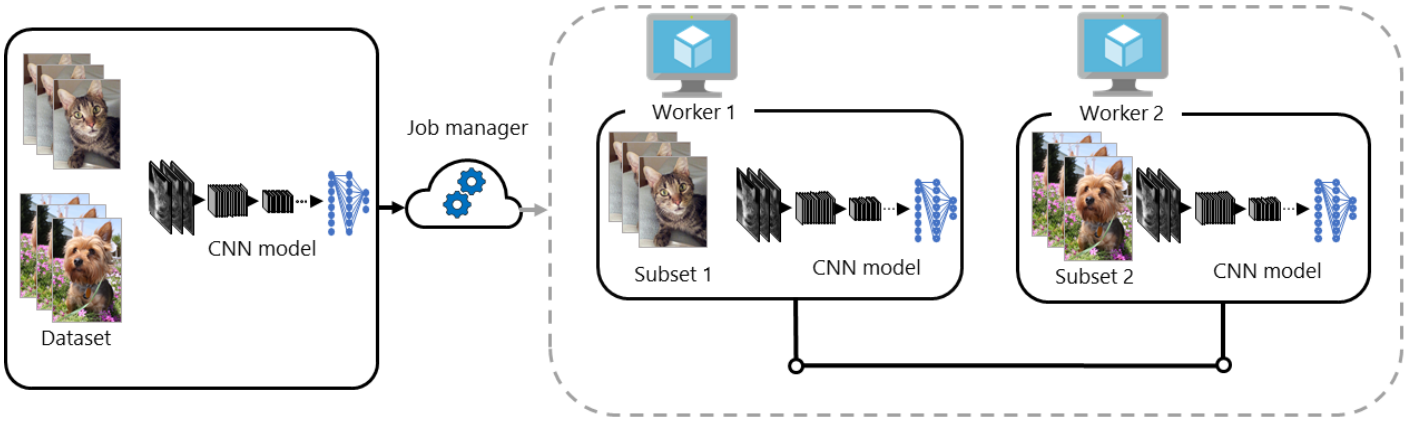


Scenario: Image classification is a widely applied technique in computer vision, often tackled by training a convolutional neural network (CNN). For particularly large models with large datasets, the training process can take weeks or months on a single GPU. In some situations, the models are so large that it's not possible to fit reasonable batch sizes onto the GPU. Using distributed training in these situations can shorten the training time.

In this specific scenario, a ResNet50 CNN model is trained using Horovod on the Imagenet dataset and on synthetic data. The reference implementation shows how to accomplish this task using TensorFlow.

There are several ways to train a deep learning model in a distributed fashion, including data-parallel and model-parallel approaches based on synchronous or asynchronous updates. Currently the most common scenario is data parallel with synchronous updates. This approach is the easiest to implement and is sufficient for most use cases.

In data-parallel distributed training with synchronous updates, the model is replicated across *n* hardware devices. A mini-batch of training samples is divided into *n* micro-batches. Each device performs the forward and backward passes for a micro-batch. When a device finishes the process, it shares the updates with the other devices. These values are used to calculate the updated weights of the entire mini-batch, and the weights are synchronized across the models. This scenario is covered in the GitHub repository.



This architecture can also be used for model-parallel and asynchronous updates. In model-parallel distributed training, the model is divided across *n* hardware devices, with each device holding a part of the model. In the simplest implementation, each device may hold a layer of the network, and information is passed between devices during the forward and backwards pass. Larger neural networks can be trained this way, but at the cost of performance, since devices are constantly waiting for each other to complete either the forward or backwards pass. Some advanced techniques try to partially alleviate this issue by using synthetic gradients.

The steps for training are:

1. Create scripts that will run on the cluster and train your model, then transfer them to file storage.
2. Write the data to Premium Blob Storage.
3. Create an Azure Machine Learning workspace. This will also create an Azure Container Registry to host your Docker Images.
4. Create an Azure Machine Learning GPU Cluster.
5. Submit jobs. For each job with unique dependencies, a new Docker image is built and pushed to your container registry. During execution, the appropriate Docker image runs and executes your script.
6. All the results and logs will be written to Blob storage.

Architecture

This architecture consists of the following components.

Azure Machine Learning Compute plays the central role in this architecture by scaling resources up and down according to need. AzureML Compute is a service that helps provision and manage clusters of VMs, schedule jobs, gather results, scale resources,and handle failures. It supports GPU-enabled VMs for deep learning workloads.

Blob storage is used to store the logs and results.

Premium Blob storage is used to store the training data. The premium blob storage is mounted in the nodes using blob fuse. Premium blob offers better performance than standard blob and is recommended for distributed training scenarios. When mounted blob fuse uses caching so that the data is saved locally. This means that after the first epoch subsequent epochs will read from the local storage which is the most performant option.

Container Registry is used to store the Docker image that Machine Learning Compute uses to run the training.

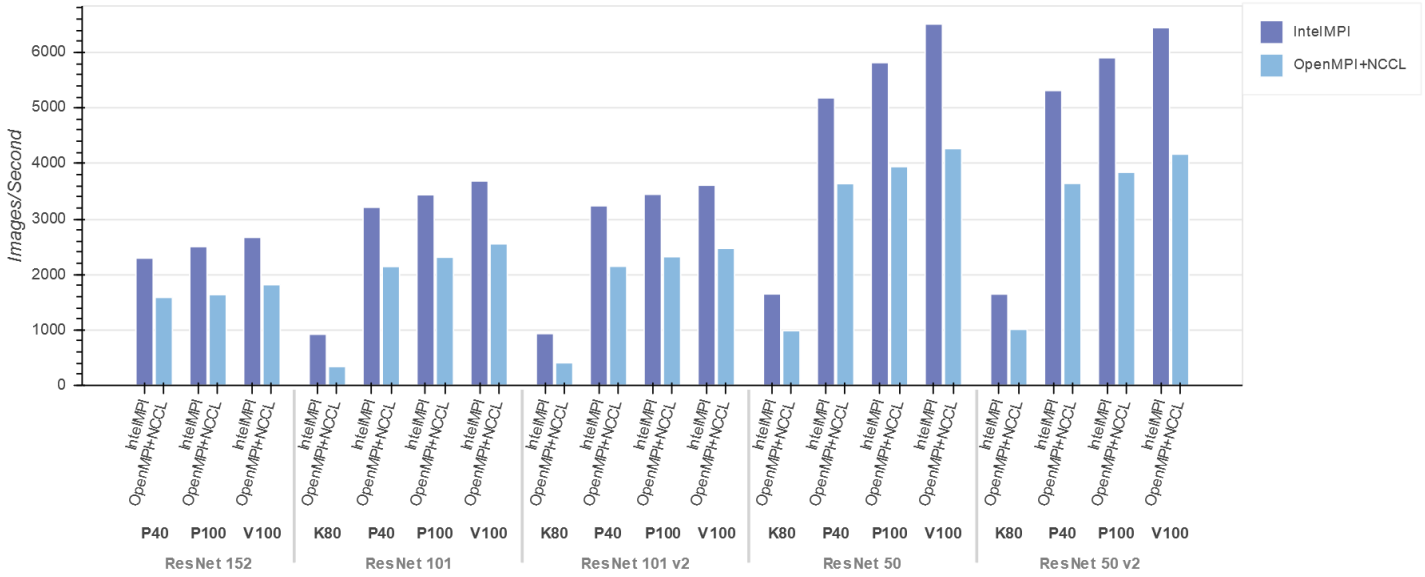
Performance considerations

Azure provides four GPU-enabled VM types suitable for training deep learning models. They range in price and speed from low to high as follows:

Azure VM series	NVIDIA GPU
NC	K80
ND	P40
NCv2	P100
NCv3	V100

We recommended scaling up your training before scaling out. For example, try a single V100 before trying a cluster of K80s.

The following graph shows the performance differences for different GPU types based on benchmarking tests carried out using TensorFlow and Horovod. The graph shows throughput of 32 GPU clusters across various models, on different GPU types and MPI versions. Models were implemented in TensorFlow 1.9



Each VM series shown in the previous table includes a configuration with InfiniBand. Use the InfiniBand configurations when you run distributed training, for faster communication between nodes. InfiniBand also increases the scaling efficiency of the training for the frameworks that can take advantage of it. For details, see the Infiniband benchmark comparison.

Although Machine Learning Compute can mount Blob storage using the blobfuse adapter, we don't recommend using Blob Storage this way for distributed training, because the performance isn't good enough for the majority of cases to handle the necessary throughput. Use Premium Blob for the data as shown in the architecture diagram.

Scalability considerations

The scaling efficiency of distributed training is always less than 100 percent due to network overhead — syncing the entire model between devices becomes a bottleneck. Therefore, distributed training is most suited for large models that cannot be trained using a reasonable batch size on a single GPU, or for problems that cannot be addressed by distributing the model in a simple, parallel way.

Distributed training is not recommended for running hyperparameter searches. The scaling efficiency affects performance and makes a distributed approach less efficient than training multiple model configurations separately.

One way to increase scaling efficiency is to increase the batch size. That must be done carefully, however, because increasing the batch size without adjusting the other parameters can hurt the model's final performance.

Storage considerations

When training deep learning models, an often-overlooked aspect is where the data is stored. If the storage is too slow to keep up with the demands of the GPUs, training performance can degrade.

Machine Learning Compute supports many storage options. For best performance it is advisable that you download the data locally to each node. However, this can be cumbersome, because all the nodes must download the data from Blob Storage, and with the ImageNet dataset, this can take a considerable amount of time. By default AzureML mounts storage such that it caches the data locally. This means in practice that after the first epoch the data is read from local storage. This combined with Premium Blob Storage offers a good compromise between ease of use and performance.

Data format

With large datasets it is often advisable to use data formats such as TFRecords and parquet which provide better IO performance than multiple small image files.

Security considerations

Encrypt data at rest and in motion

In scenarios that use sensitive data, encrypt the data at rest — that is, the data in storage. Each time data moves from one location to the next, use SSL to secure the data transfer. For more information, see the Azure Storage security guide.

Secure data in a virtual network

For production deployments, consider deploying the Azure Machine Learning cluster into a subnet of a virtual network that you specify. This allows the compute nodes in the cluster to communicate securely with other virtual machines or with an on-premises network. You can also use service endpoints with blob storage to grant access from a virtual network.

Monitoring considerations

While running your job, it's important to monitor the progress and make sure that things are working as expected. However, it can be a challenge to monitor across a cluster of active nodes.

Azure Machine Learning offers many ways to instrument your experiments. The stdout/stderr from your scripts are automatically logged. These logs are automatically synced to your workspace Blob storage. You can either view these files through the Azure portal, or download or stream them using the Python SDK or Azure Machine Learning CLI. If you log your experiments using Tensorboard, these logs are automatically synced and you can access them directly or use the Azure Machine Learning SDK to stream them to a Tensorboard session.

Deployment

The reference implementation of this architecture is available on GitHub. Follow the steps described there to conduct distributed training of deep learning models across clusters of GPU-enabled VMs.

Next steps

The output from this architecture is a trained model that is saved to blob storage. You can operationalize this model for either real-time scoring or batch scoring. For more information, see the following reference architectures:

- Real-time scoring of Python scikit-learn and deep learning models on Azure
- Batch scoring on Azure for deep learning models