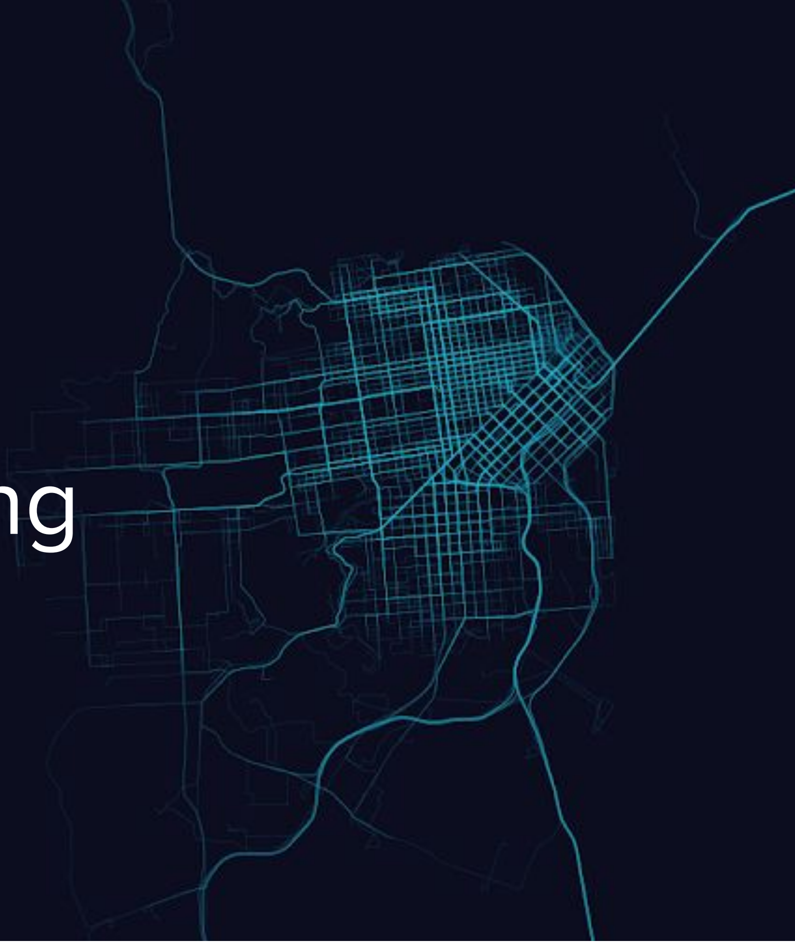
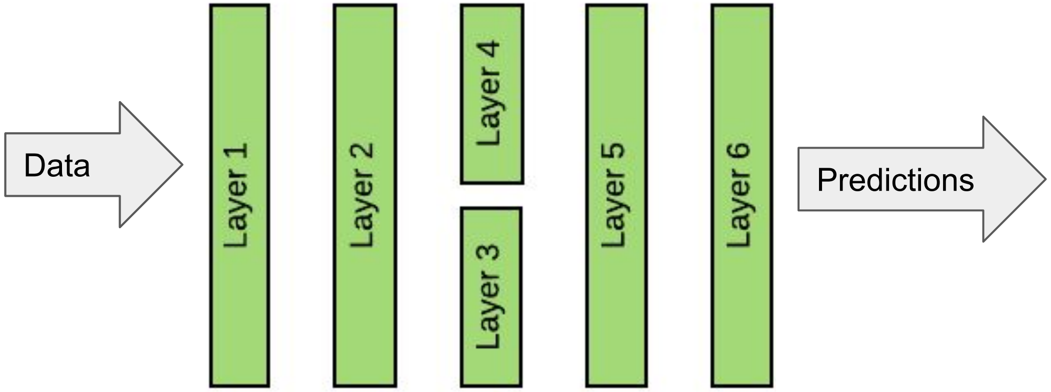
Distributed Deep Learning with Horovod and Spark

Fardin Abdi
Michelangelo Deep Learning Platform
Uber



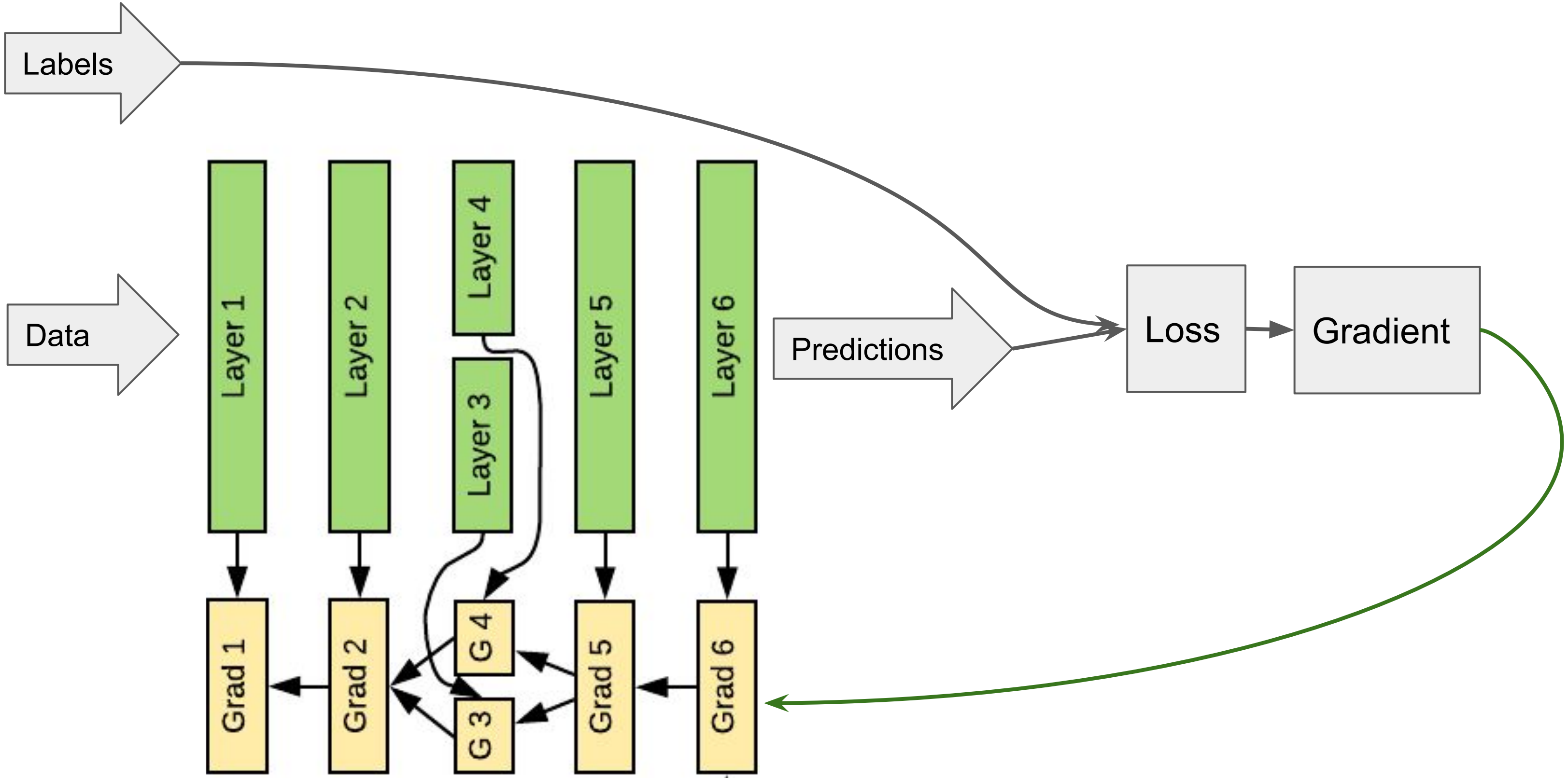
Uber

How does Deep Learning work?



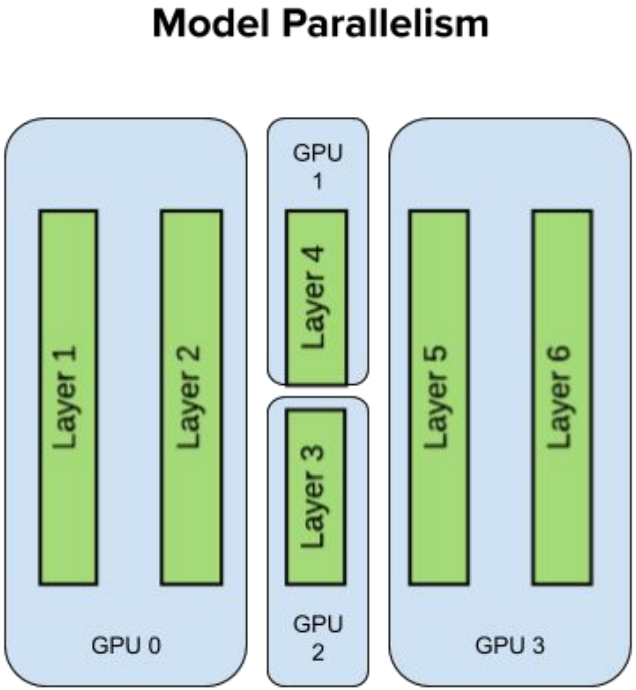
NOTE: Limited Parallelism

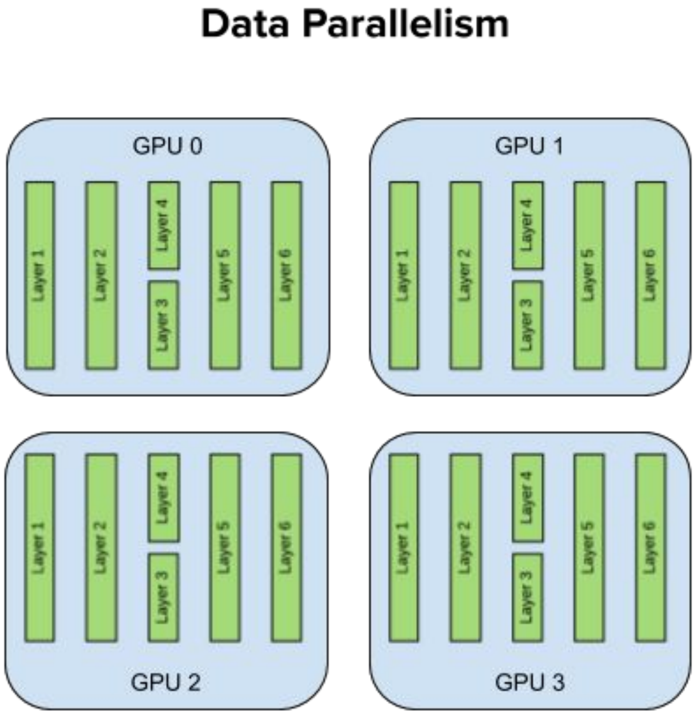
How does Deep Learning training work?



Uber

How does distributed training work?





Uber

Data Parallelism

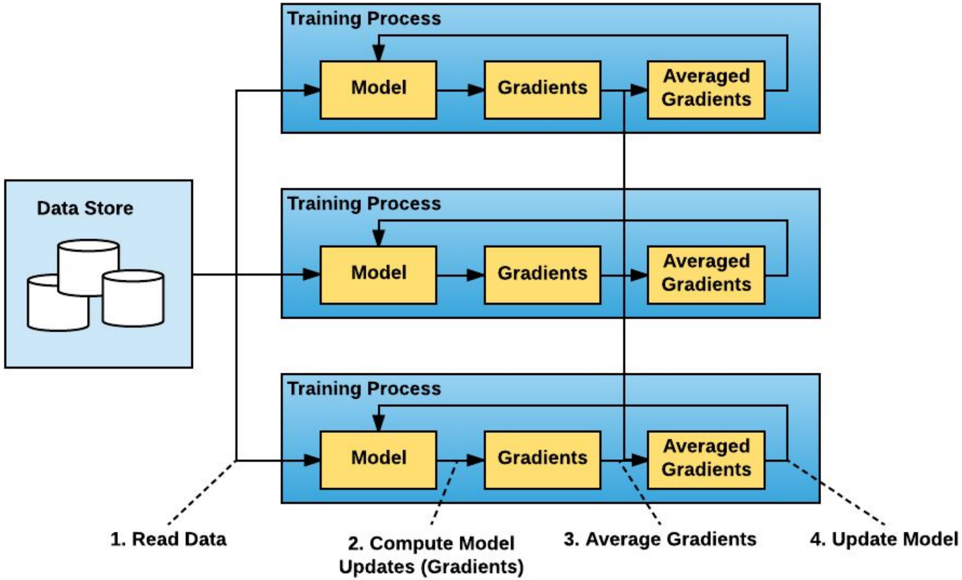
Increasing amount of data
Model complexity does not need to increase

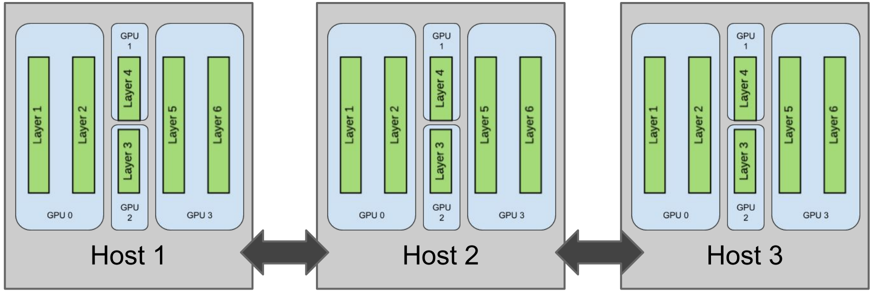
Model must fit in a single GPU:

- NVIDIA® V100 GPUs 32GB

Can mix and match:

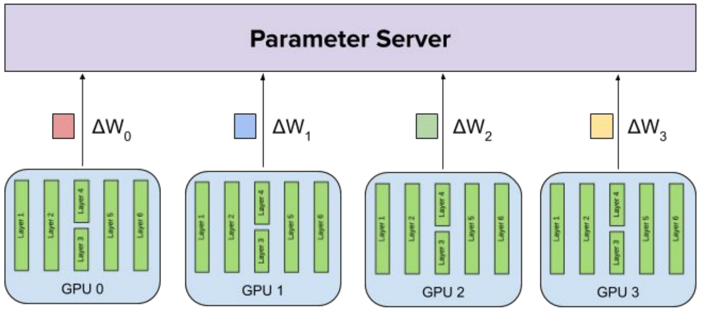
- Data parallel inter-host
- Model parallel intra-host

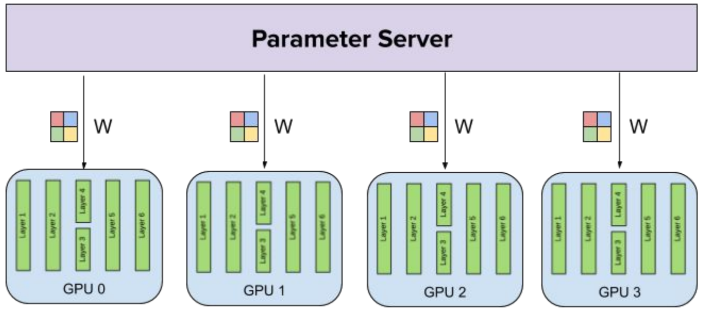




Uber

Distributed Deep Learning (I); Data Parallel Training with Parameter Servers





Pros

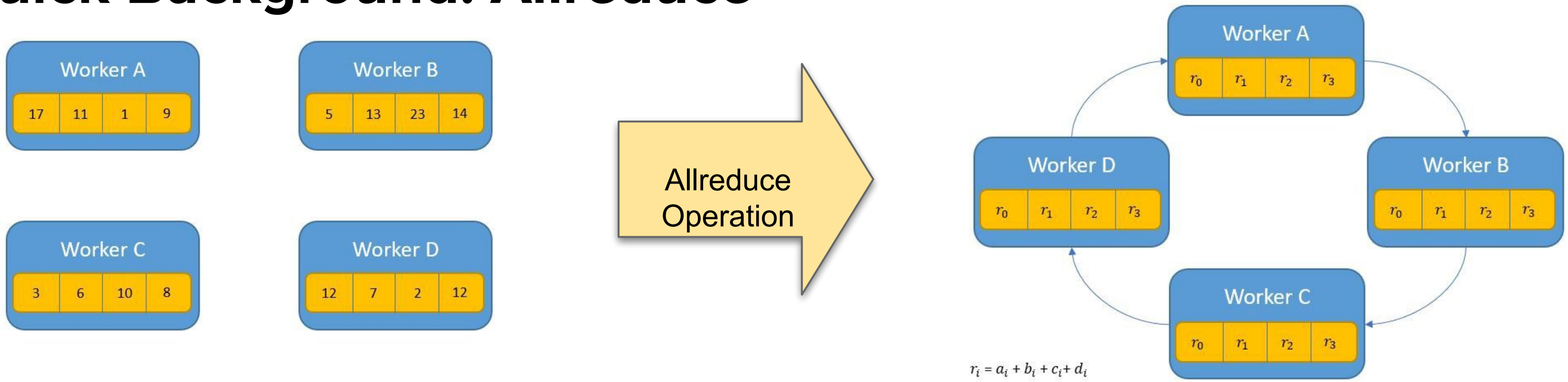
- Fault tolerance
- Supports *asynchronous SGD*

Cons

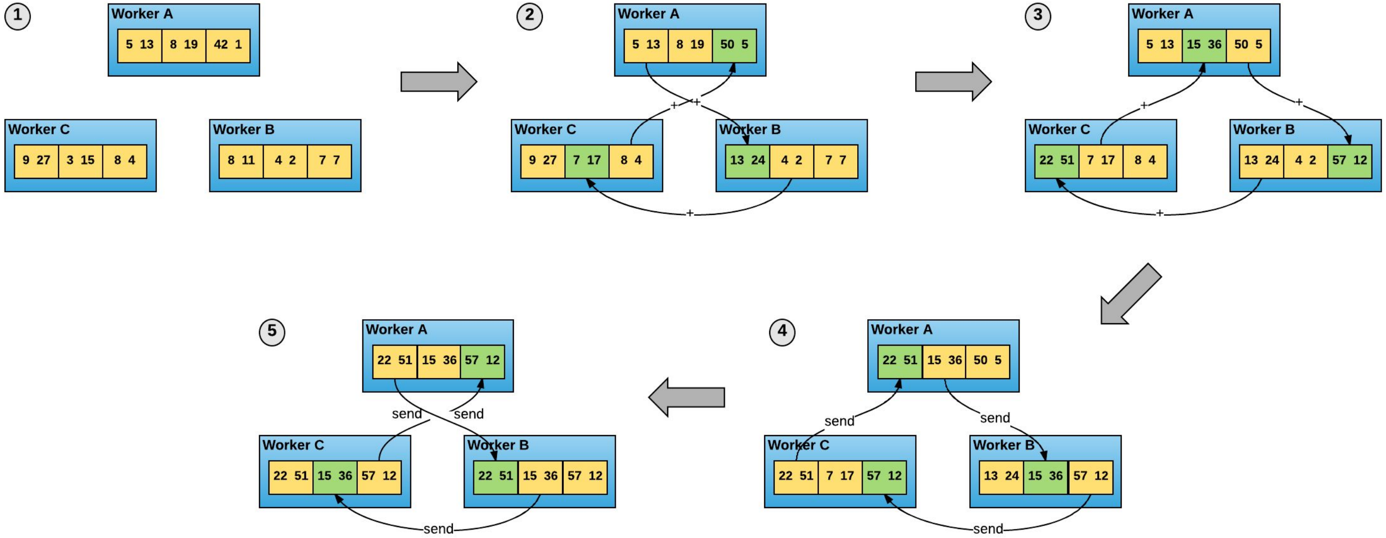
- Usability (tight coupling between *model* and *parameter servers*)
- Scalability (many-to-one)
- Convergence (with async SGD)

Uber

Quick Background: Allreduce



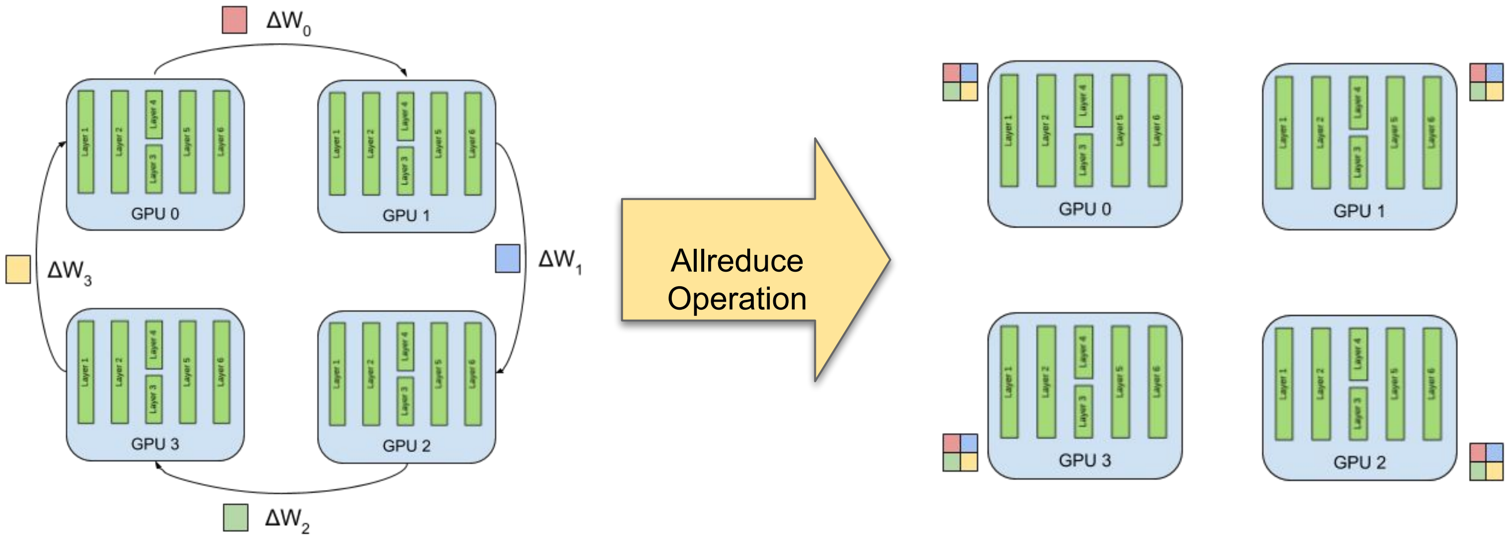
Example Allreduce



Patarasuk, P., & Yuan, X. (2009). Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2), 117-124. doi:10.1016/j.jpdc.2008.09.002

Uber

Distributed Deep Learning (II); Data Parallel Training with Allreduce



Pros

- Bandwidth efficient
- Scalability (many-to-one)
- Convergence (with sync SGD)

Cons

- Fault tolerance

Uber

Horovod; Library for Distributed Deep Learning.

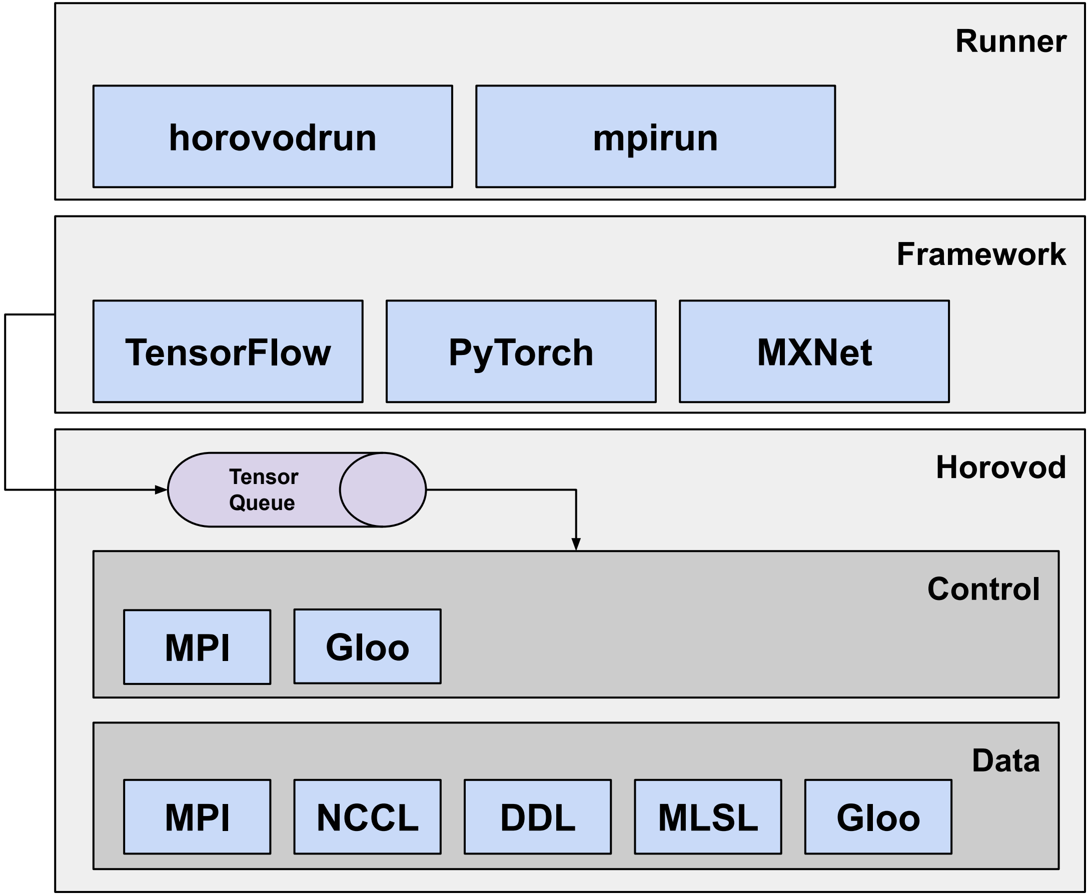
- Uses Allreduce
- Works with stock TensorFlow, Keras, PyTorch, and Apache MXNet
- Only 5-6 lines of code change required
- Installs on top via `pip install horovod`.
- Uses advanced algorithms & leverage features of high-performance networks (RDMA, GPUDirect).
- Separates infrastructure from ML engineers:
    - Infra team provides container & MPI environment
    - ML engineers use DL frameworks that they love
    - Both Infra team and ML engineers have consistent expectations for distributed training across frameworks

HOROVOD

horovod.ai

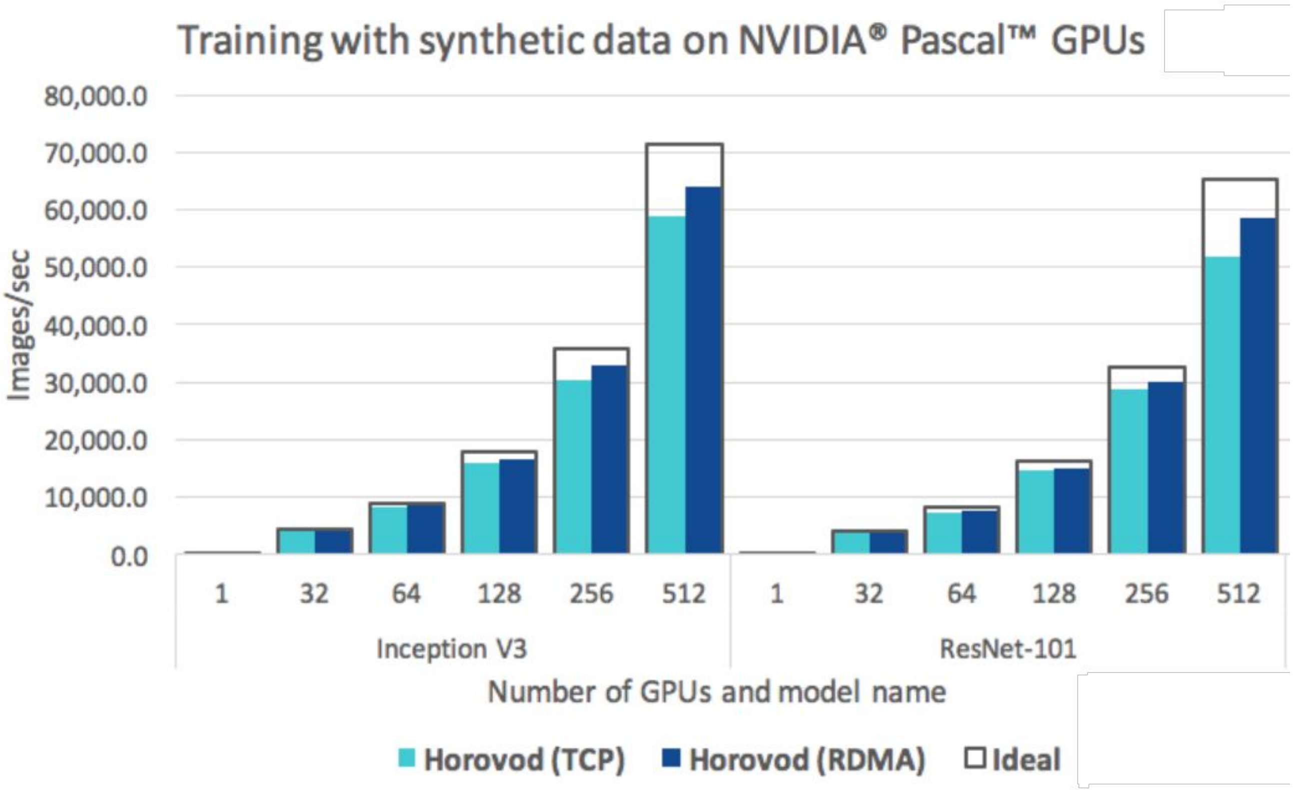
Uber

Horovod Stack



Uber

Horovod Performance



Uber

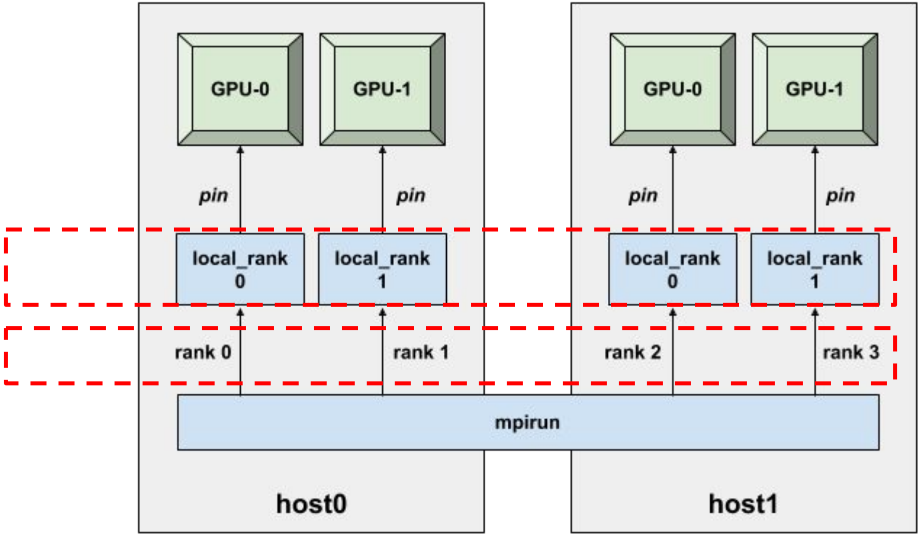
Horovod scales well beyond 128 GPUs. RDMA helps at a large scale.

Using Horovod

Host and Rank

Local rank per host

Global rank of each worker



- One rank per GPU
- Hosts do not have to have same number of ranks
- `hvd.size()` will return 4
- `hvd.local_size()` will return 2

Uber

Horovod+PyTorch; example.py

Import library

Initialize

Pin GPU to local rank.

Adjust learning rate

Wrap optimizer with Horovod Distributed Optimizer

Broadcast parameter.

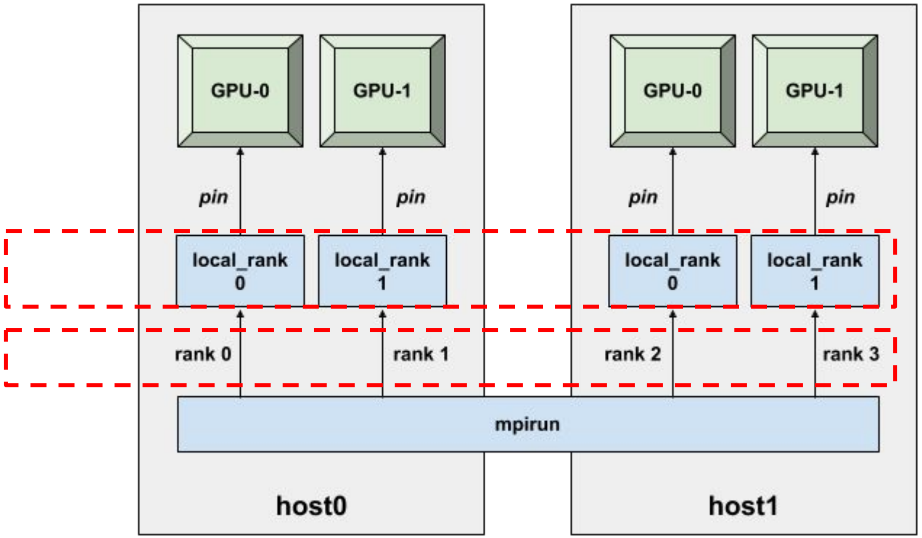
```
import torch
import horovod.torch as hvd

hvd.init()

torch.cuda.set_device(hvd.local_rank())

model = Net(...)
model.cuda()
optimizer = optim.SGD(model.parameters(), lr=0.01 * hvd.size())

optimizer = hvd.DistributedOptimizer(optimizer, named_parameters=model.named_parameters())


hvd.broadcast_parameters(model.state_dict(), root_rank=0)

# Read the data
for epoch in range(100):
  for batch_idx, (data, target) in …:
    optimizer.zero_grad()
    output = model(data)
    loss = F.nll_loss(output, target)
    loss.backward()
    optimizer.step()
```

Uber

Running Horovod

Local rank per host

Global rank of each worker



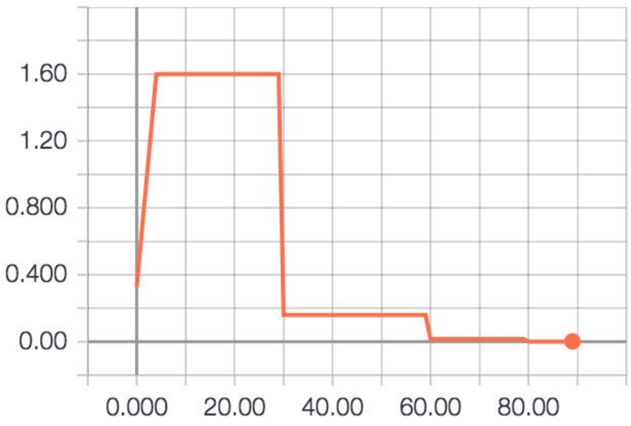
- `example.py` needs to be present on all the hosts
- host1 and host2 need to be able to ssh into each other.
- Start the training:
  - Single-node: `$ horovodrun -np 2 -H localhost:2 python example.py`
  - Multi-node: `$ horovodrun -np 4 -H host1:2,host2:2 python example.py`

Adjusting Learning Rate & Add Distributed Optimizer

```
opt = optim.SGD(model.parameters(), lr=0.01 * hvd.size())

opt = hvd.DistributedOptimizer(opt, named_parameters=model.named_parameters())
```

- Recommend linear scaling of learning rate:
    - $LR_N = LR_1 * N$
    - Smooth warm-up for the first K epochs

- Facebook paper:
    - [Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour](arxiv.org/abs/1706.02677)
        - arxiv.org/abs/1706.02677

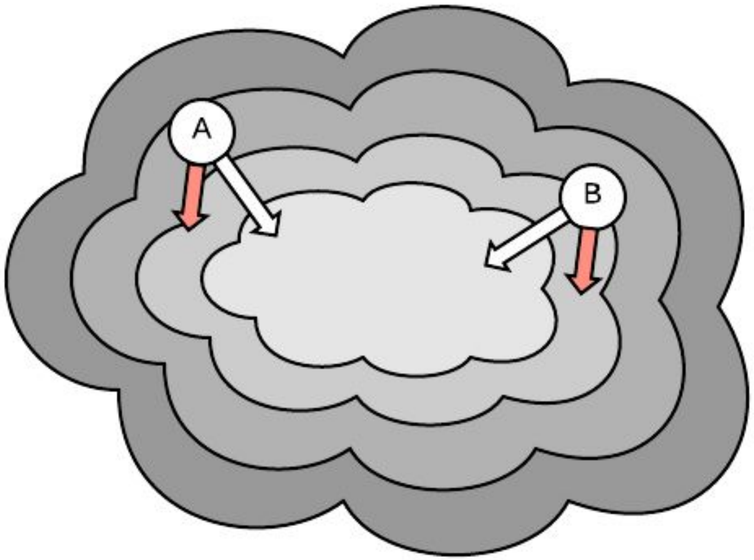


Uber

Synchronizing Initial State

```
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
```

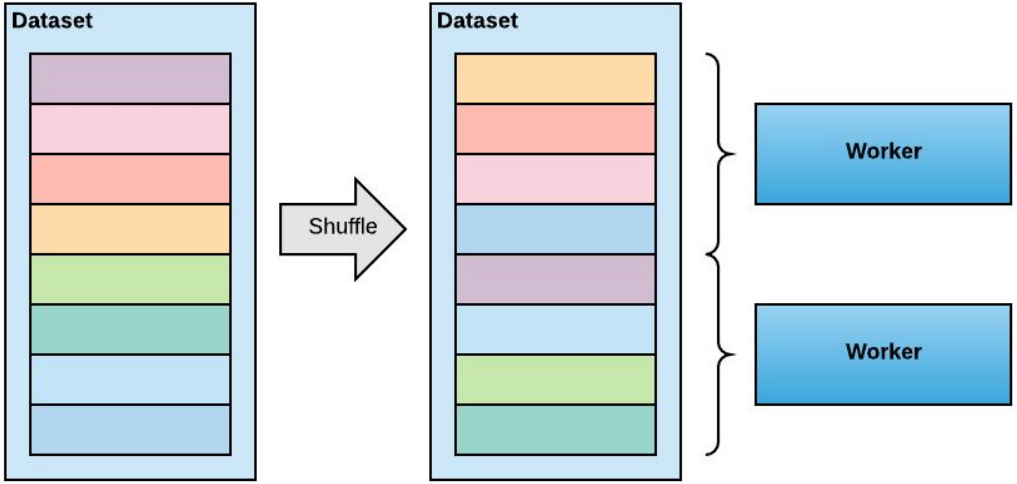
All the workers start from identical model.



Uber

Data: Partitioning

- Shuffle the dataset
- Partition records among workers
- Train by sequentially reading the partition
- After epoch is done, reshuffle and partition again

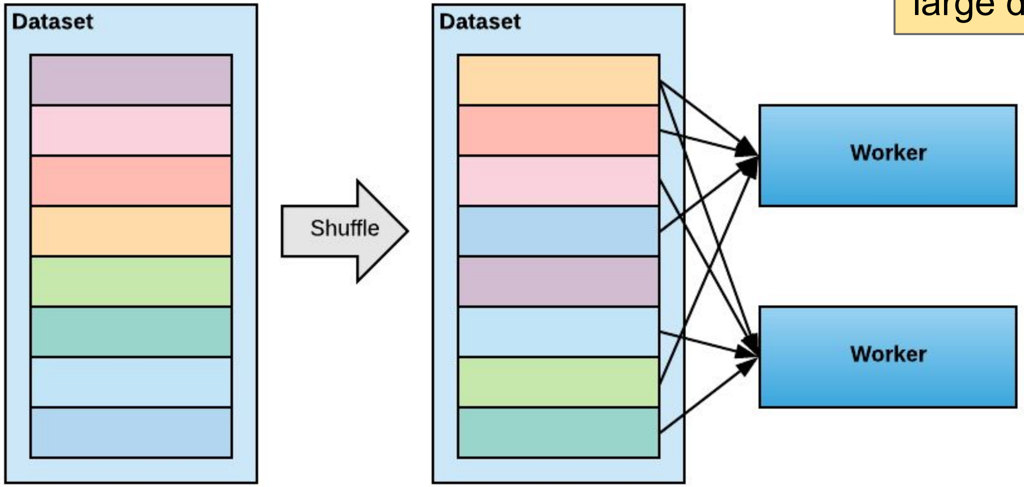


Uber

Data: Random Sampling

- Shuffle the dataset
- Train by randomly reading data from whole dataset
- After epoch is done, reshuffle

NOTE: no guarantee to read samples exactly once (or at all), but converges similarly with large datasets



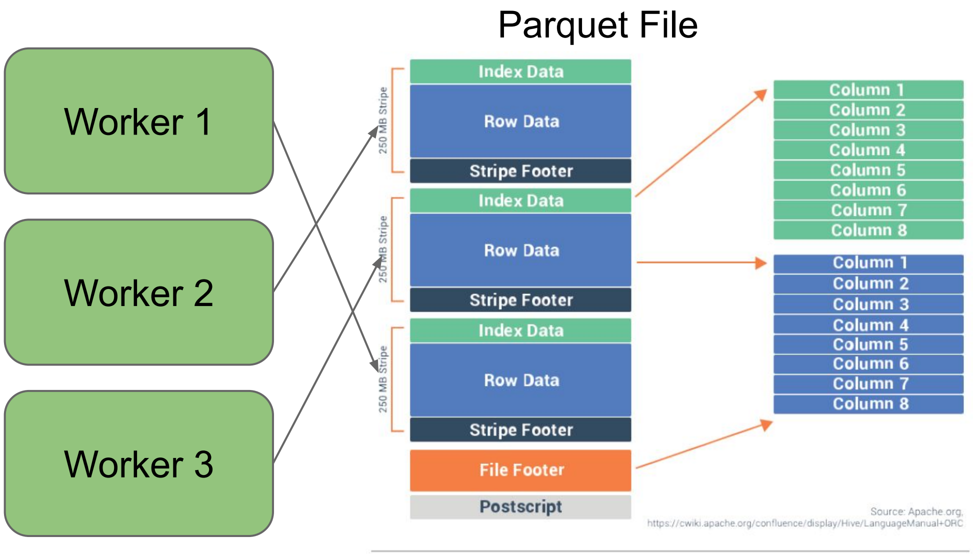
Uber

Data

- Horovod does not handle data access.
- Users can implement their own data distribution and partitioning
- We recommend using Petastorm+Parquet records (Horovod examples)
- Common ways:
    - Copy all the data into local storage of the workers (smaller datasets)
    - Each worker streams the data from HDFS/S3

Data Ingestion with PetaStorm; Recommended Way

- Sharding:
  - Each worker is randomly assigned to a subset of the row-groups
- Shuffling:
  - Workers read the row-groups in a shuffled order
- Optimized IO
  - multi-threaded reading
  - Prefetching



#6. Data Review

- Random sampling may cause some records to be read multiple times in a single epoch, while others not read at all
- In practice, both approaches typically yield same results
- **Conclusion**: use the most convenient option for your case
- **Remember**: validation can also be distributed, but need to make sure to average validation results from all the workers when using learning rate schedules that depend on validation
  - Horovod comes with **`MetricAverageCallback`** for Keras

Best Practices: Learning Rate Adjustment

- Yang You, Igor Gitman, Boris Ginsburg in paper “Large Batch Training of Convolutional Networks” demonstrated scaling to batch of 32K examples (arxiv.org/abs/1708.03888)
  - Use per-layer adaptive learning rate scaling
- Google published a paper “Don't Decay the Learning Rate, Increase the Batch Size” (arxiv.org/abs/1711.00489) arguing that typical learning rate decay can be replaced with an increase of the batch size

Full Example - Keras

```
from tensorflow import keras
import tensorflow.keras.backend as K
import tensorflow as tf
import horovod.tensorflow.keras as hvd

# Initialize Horovod.
hvd.init()

# Pin GPU to be used
config = tf.ConfigProto()
config.gpu_options.visible_device_list =
  str(hvd.local_rank())
K.set_session(tf.Session(config=config))

# Build model…
model = ...
opt = keras.optimizers.Adadelta(
  lr=1.0 * hvd.size())

# Add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)

model.compile(
  loss='categorical_crossentropy',
  optimizer=opt,
  metrics=['accuracy'])

# Broadcast initial variable state.
callbacks =
[hvd.callbacks.BroadcastGlobalVariablesCallback(0)]

model.fit(
  x_train,
  y_train,
  callbacks=callbacks,
  epochs=10,
  validation_data=(x_test, y_test))
```

Uber

TensorFlow

```
import tensorflow as tf
import horovod.tensorflow as hvd


# Initialize Horovod
hvd.init()

# Pin GPU to be used
config = tf.ConfigProto()
config.gpu_options.visible_device_list =
  str(hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.MomentumOptimizer(
  lr=0.01 * hvd.size())

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)
```

```
# Add hook to synchronize initial state
hooks =[hvd.BroadcastGlobalVariablesHook(0)]

# Only checkpoint on rank 0
ckpt_dir = "/tmp/train_logs" \
  if hvd.rank() == 0 else None

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of
# session initialization, restoring from a
# checkpoint, saving to a checkpoint, and
# closing when done or an error occurs.
with
tf.train.MonitoredTrainingSession(checkpoint_dir=ckpt_
dir, config=config, hooks=hooks) as mon_sess:
  while not mon_sess.should_stop():
    # Perform synchronous training.
    mon_sess.run(train_op)
```

Apache MXNet

```
import torch
import horovod.mxnet as hvd

# Initialize Horovod
hvd.init()

# Horovod: pin GPU to local rank.
context = mx.gpu(hvd.local_rank())

# Build model.
net = …
loss = ...
model = mx.mod.Module(symbol=loss, context=context)

# Wrap optimizer with DistributedOptimizer.
opt = hvd.DistributedOptimizer(opt)

# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.get_params(),
root_rank=0)

model.fit(...)
```

Uber

Running Horovod

Single-node:

```
$ horovodrun -np 4 -H localhost:4 python train.py
```

Multi-node: -- Code must be available on all the nodes

```
$ horovodrun -np 16 -H server1:4,server2:4,server3:4,server4:4 python train.py
```

Horovod on Spark

```
In [1]: from pyspark import SparkConf
        from pyspark import SparkContext
        import horovod.spark

In [2]: sc = SparkContext(conf=SparkConf())

In [3]: def train():
            import horovod.tensorflow as hvd
            hvd.init()
            return hvd.rank()

In [4]: print(horovod.spark.run(train, num_proc=4))

        [0, 1, 2, 3]

In [ ]:
```

Uber

Why Spark?

- Allows users to leverage existing Spark infrastructure
  - Including Jupyter and IPython!
- Data preparation & model training in the same environment
- Save to Parquet and use Petastorm for data ingestion
  - Takes care of random shuffling, fault tolerance, etc
  - https://github.com/uber/petastorm

Uber

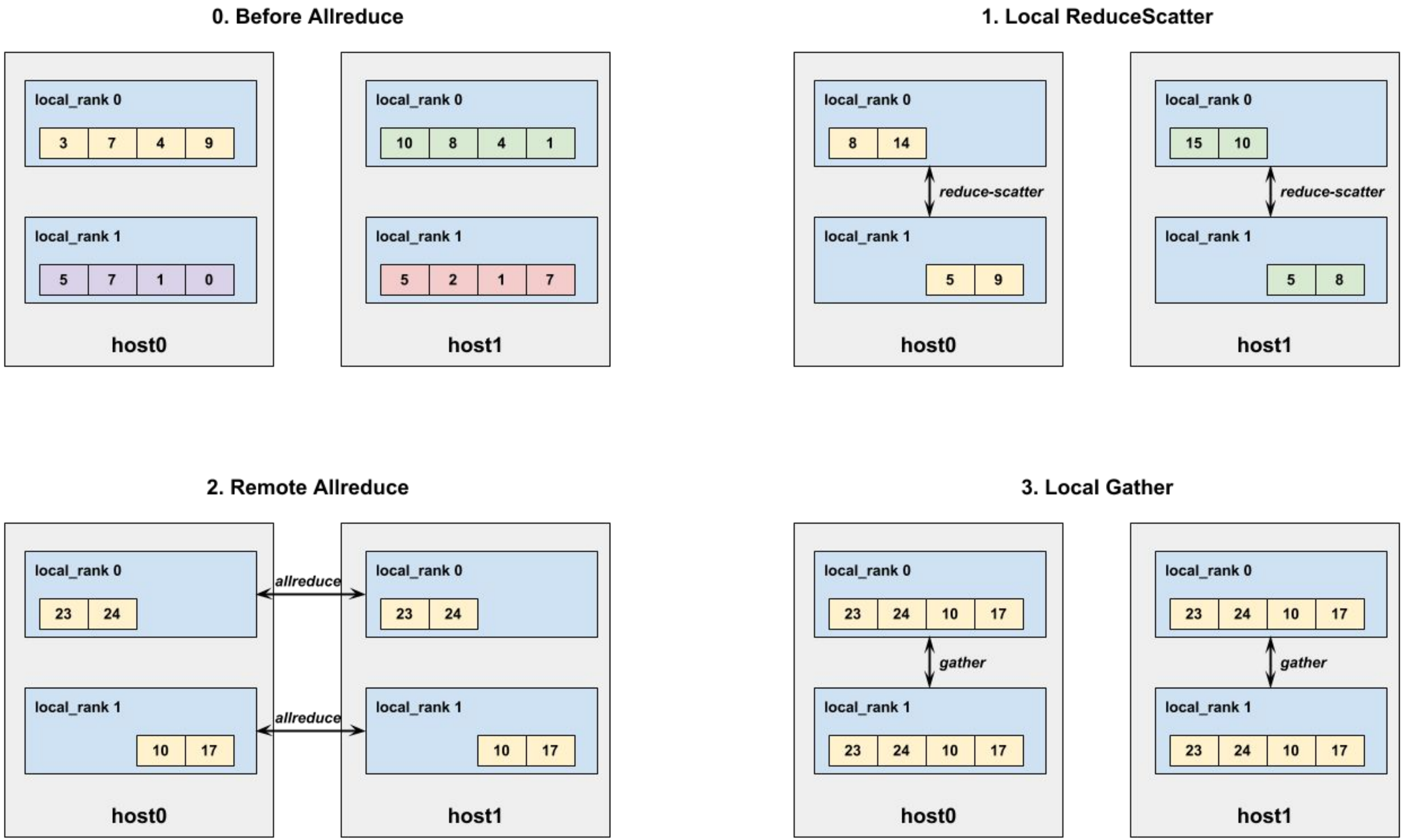
Advanced Features

Horovod Knobs: Hierarchical Algorithms

```
$ HOROVOD_HIERARCHICAL_ALLREDUCE=1 horovodrun ...
$ HOROVOD_HIERARCHICAL_ALLGATHER=1 horovodrun ...
```

- Contributed by NVIDIA & Amazon
- First allreduce locally, then allreduce across nodes in parallel
  - Each worker responsible for a different chunk of the buffer
- Speeds up training for very large cluster setups
  - Homogenous nodes (same # GPUs)
  - Many GPUs per node

Hierarchical Allreduce: Example



Horovod Knobs: Tensor Fusion

```
$ HOROVOD_FUSION_THRESHOLD=67108864 HOROVOD_CYCLE_TIME=5 horovodrun ...
```

- Batch tensors together during allreduce
- **Fusion Threshold:** size of batching buffer (in bytes)
- **Cycle Time:** wait time between sending batches (in milliseconds)

Horovod Knobs: Auto Tuning with Bayesian Optimization

Uber

Use **`HOROVOD_AUTOTUNE=1`** to find the best Horovod parameters

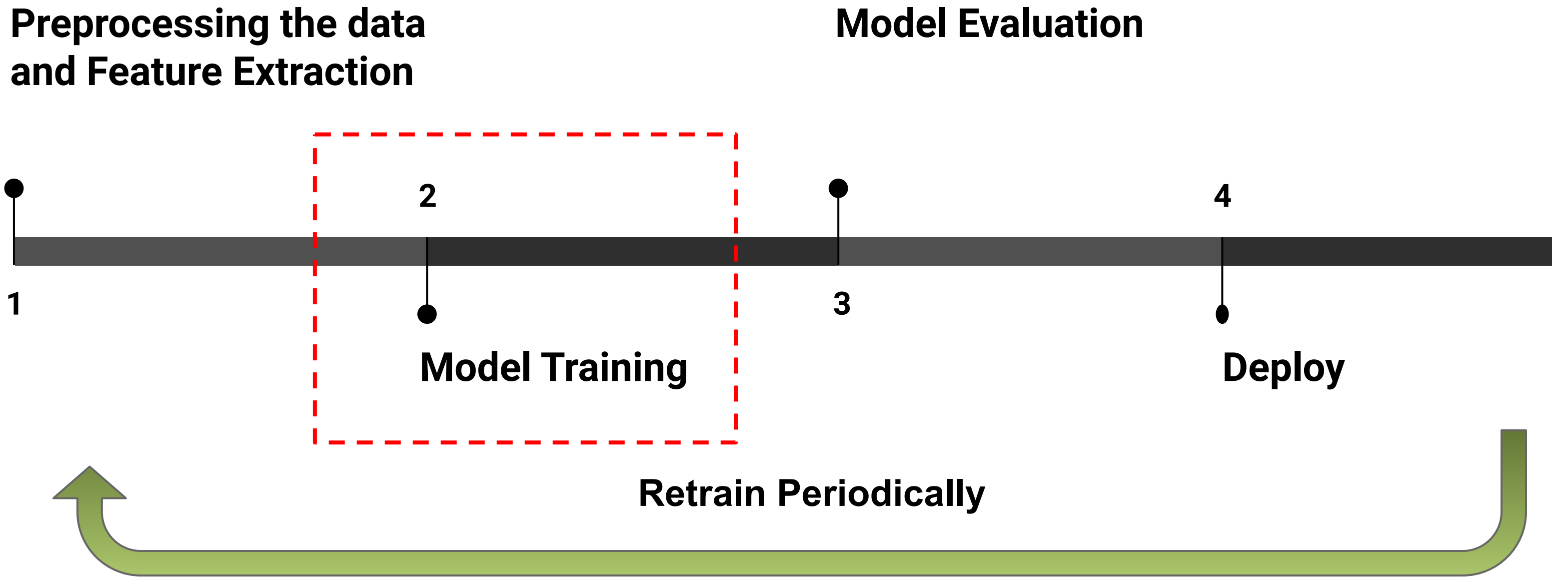
Horovod Knobs: Gradient Compression

- FP16 allreduce
  - `hvd.DistributedOptimizer(..., compression=hvd.Compression.fp16)`
  - Reduces arithmetic computation on GPU
  - Reduces network utilization
- Not selected by Auto Tuning since it may affect model convergence
- More techniques coming - contributions welcome!

Horovod Deep Learning Estimators

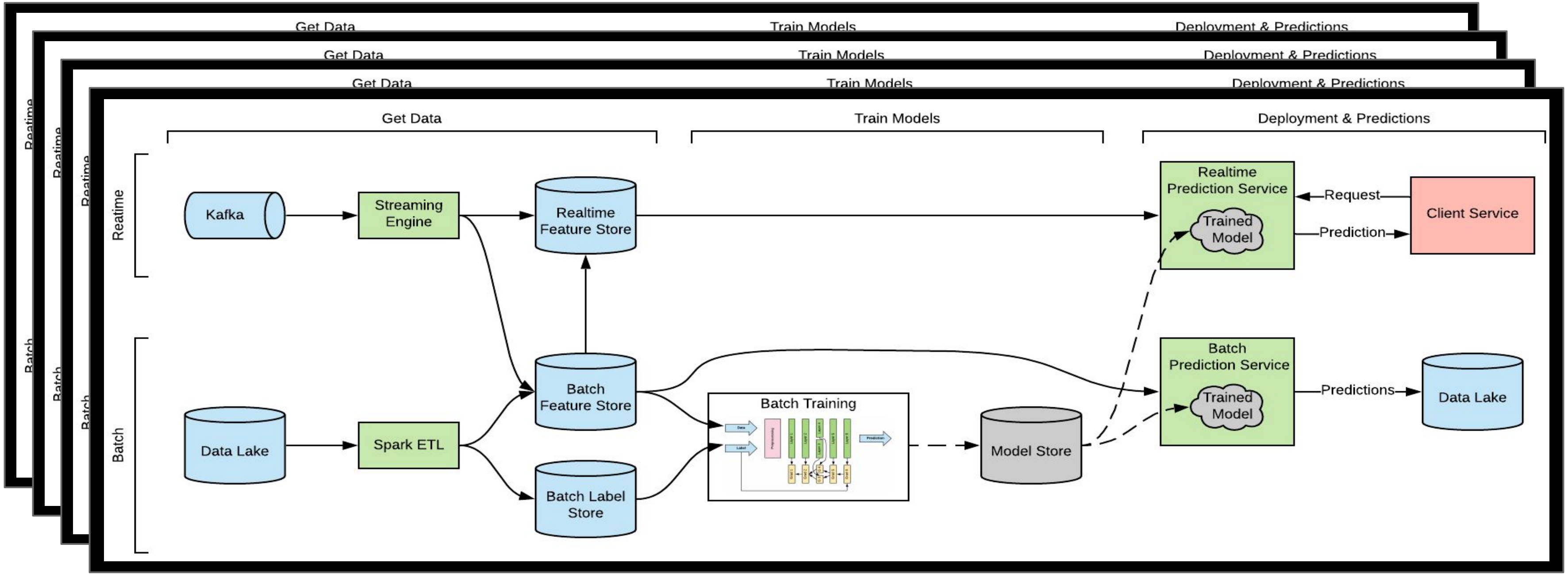
Horovd + Spark:
Deep Learning for Tabular Datasets

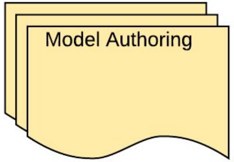
Machine Learning Cycle



Uber

Zoom out further





Uber

What do we need?

- Feature engineering and data prep
- Distributed training
- Evaluation
- Prediction:
  - Batch prediction
  - Online Serving

Horovod's Deep Learning Spark Estimator (Released in Dec 2019)

- Train Deep Learning Models Directly on Spark DataFrames
- Serialize entire training and evaluation pipeline
- Simpler User Experience

Uber

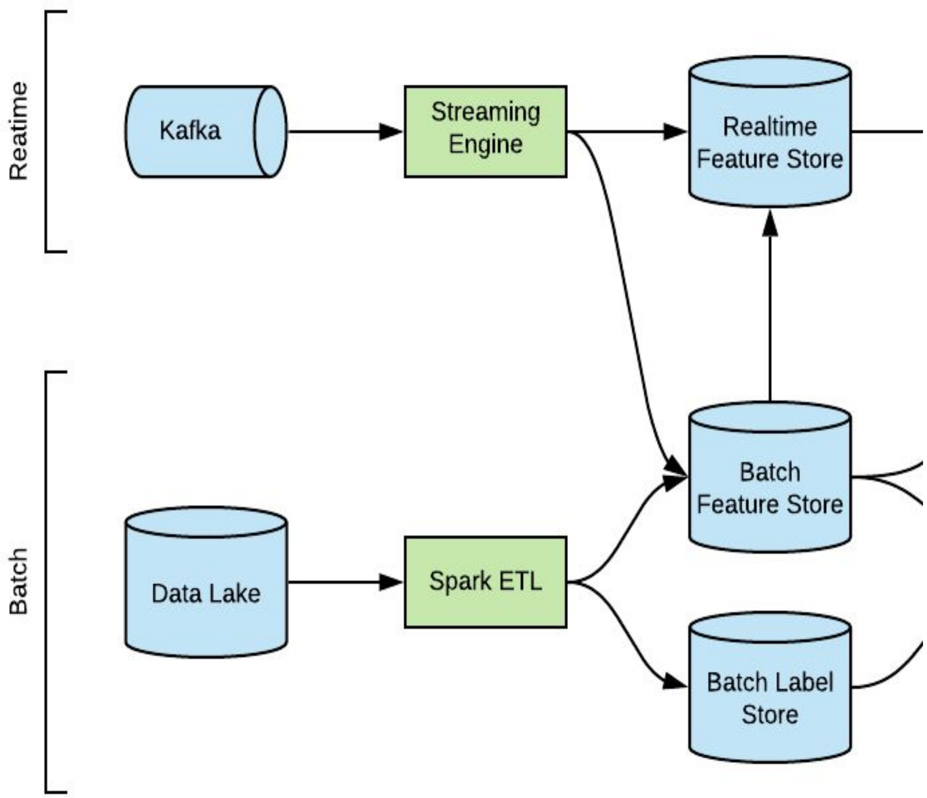
How can you implement such system?

We chose Apache Spark:

- Powerful ETL
- Easy integration with XGBoost
- Existing systems built on top of Spark
- Close collaboration with Spark community

(1) Feature Store

- Data pipelines are bread & butter of Apache Spark
- Good integration of real-time & batch ecosystems
- Overall: not a surprising choice

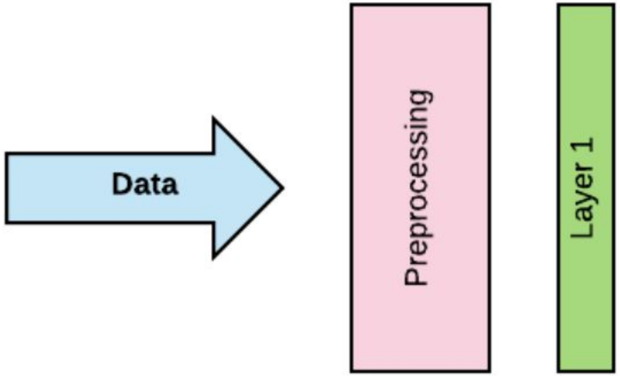


Uber

Preprocessing

Often comes in two different kinds:

1. Example-dependent
   a. Image color adjustments
   b. Image resizing
   c. Normalizing
2. Dataset-dependent
   a. String indexing
   b. Normalization



Solution: Need to fit the preprocessing first, and then apply it.

Petastorm: Data Access for Deep Learning Training

Challenges of Training on Large Datasets:

- Shuffling
- Sharding
- Streaming / Buffering / Caching

Parquet:

- Large continuous reads (HDFS/S3-friendly)
- Fast access to individual columns
- Faster row queries in some cases
- Written and read natively by Apache Spark

Uber

Spark ETL: Extract, Transform, Load

Extract

```
df_customers = spark.read.format("jdbc").options(
     url ="jdbc:mysql://xyz.amazonaws.com:3306/test",
     driver="com.mysql.jdbc.Driver",
     dbtable="customer", user="root", password="password").load()
```

Transform

```
sql='''
SELECT a.first_name, a.last_name, b.order_number, b.total
FROM customers a, orders b WHERE a.customer_number = b.customer_number
'''
output = spark.sql(sql)
```

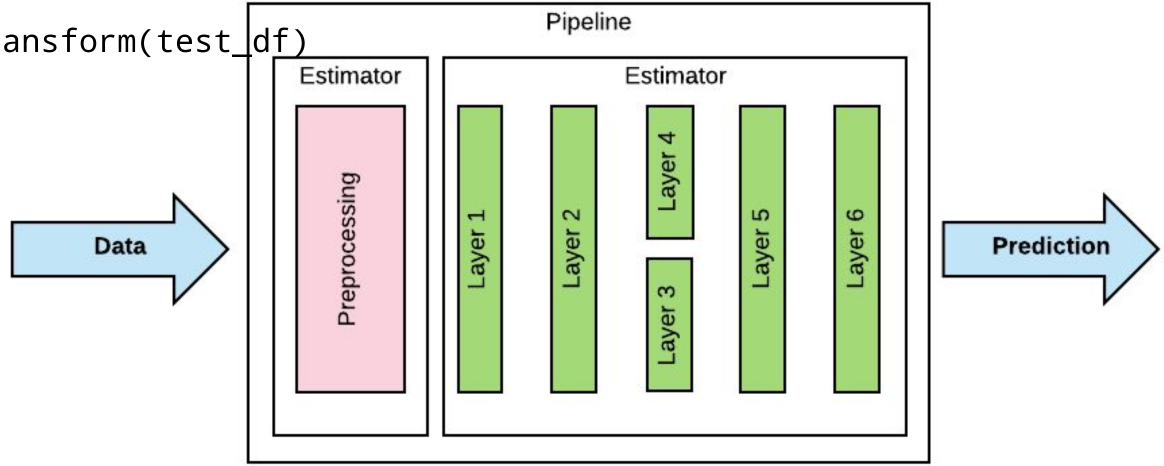
Load

```
output.write.format("orc").save("/tmp/customer_orders")
```

Uber

SparkML Pipelines; Estimator, Transformer, Pipeline

```
# Estimator and Transformer
transformer = estimator.fit(spark_dataframe)
Pred_df = transformer.transform(test_df)
# Spark ML Pipelines
estimator_pipeline = Pipeline().setStages([transformer1, transformer2, estimator])
transformer_pipeline = estimator_pipeline.fit(df)
pred_df = transformer_pipeline.transform(test_df)
```



Uber

#1 Feature Transformation with Spark ML Pipelines

```
data_processing_pipeline = Pipeline(stages=[
    IntToFloatEncoder(inputCols=[...],...),
    StringIndexer(inputCols=[...],...),
    OneHotEncoder(inputCols=[...],...)])


train_df, test_df = input_df.randomSplit([0.8, 0.2])

data_transformation_pipeline = data_processing_pipeline.fit

# Transform the original train/test data set with the fitte
transformed_train_data = data_transformation_pipeline.trans
transformed_test_data = data_transformation_pipeline.transf
```

Spark ML Pipelines has many more:

Word2Vec
OneHotEncoder
Tokenizer
Binarizer
Polynomial Expansion
StringIndexer
VectorIndexer
Normalizer
StandardScaler
Bucketizer
Elementwise Product
VectorAssembler,
...

Uber

#2 Deep Learning Estimator; Defining DL Model

```
# Define the model
x = Flatten()(x)
x = Dense(500, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.00005))(x)
x = Dropout(0.5)(x)
output = Dense(1, activation=act_sigmoid_scaled)(x)
model = tf.keras.Model([inputs[f] for f in all_cols], output)

# Define the optimizer
opt = tf.keras.optimizers.Adam(lr=args.learning_rate, epsilon=1e-3)
```

Uber

#3 Deep Learning Estimator

```
# Create the Estimator
from horovod.spark.keras import KerasEstimator

keras_estimator = KerasEstimator(num_proc=10,
                                 store=store,
                                 model=model,
                                 optimizer=opt,
                                 loss='mae',
                                 metrics=[exp_rmspe],
                                 feature_cols=continuous_cols = [ 'Distance', 'Temperature'],
                                 label_cols=['Sales'],
                                 validation='Validation',
                                 batch_size=128,
                                 epochs=10,
                                 verbose=2)
```

#4 Deep Learning Estimator; Train and Predict

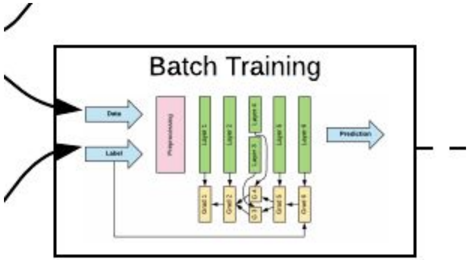
```
# Create the Store
store = hvd.spark.common.HDFSStore.create('/usr/fardin/experiment1')

# Train the model
keras_transformer = keras_estimator.fit(train_df)

# Evaluation of the Model
pred_df = keras_transformer.transform(test_df)
# run any evaluation functions on the pred_df and make decisions
```

Deep Learning Estimator Interface

```
# Save the Model or the estimator
keras_transformer.write().overwrite().save('/user/fardin/exp1/pytorch_raw_pipeline')

# Load the model for more prediction
loaded_transformer = Pipeline.read().load('/user/fardin/exp1/pytorch_raw_pipeline')
pred_df = loaded_transformer.transform(test_df)

# Incremental training
keras_model = loaded_transformer.get_model()
optimizer = loaded_transformer.get_optimizer()
keras_estimator_incremental = hvd.KerasEstimator(num_proc=args.num_proc,
                                                store=store,
                                                model=model,
                                                optimizer=optimizer,
                                                loss='mae',
                                                feature_cols['Distance', 'Temperature'],
                                                label_cols=['Sales'],
                                                batch_size=128,
                                                epochs=10)
```

Uber

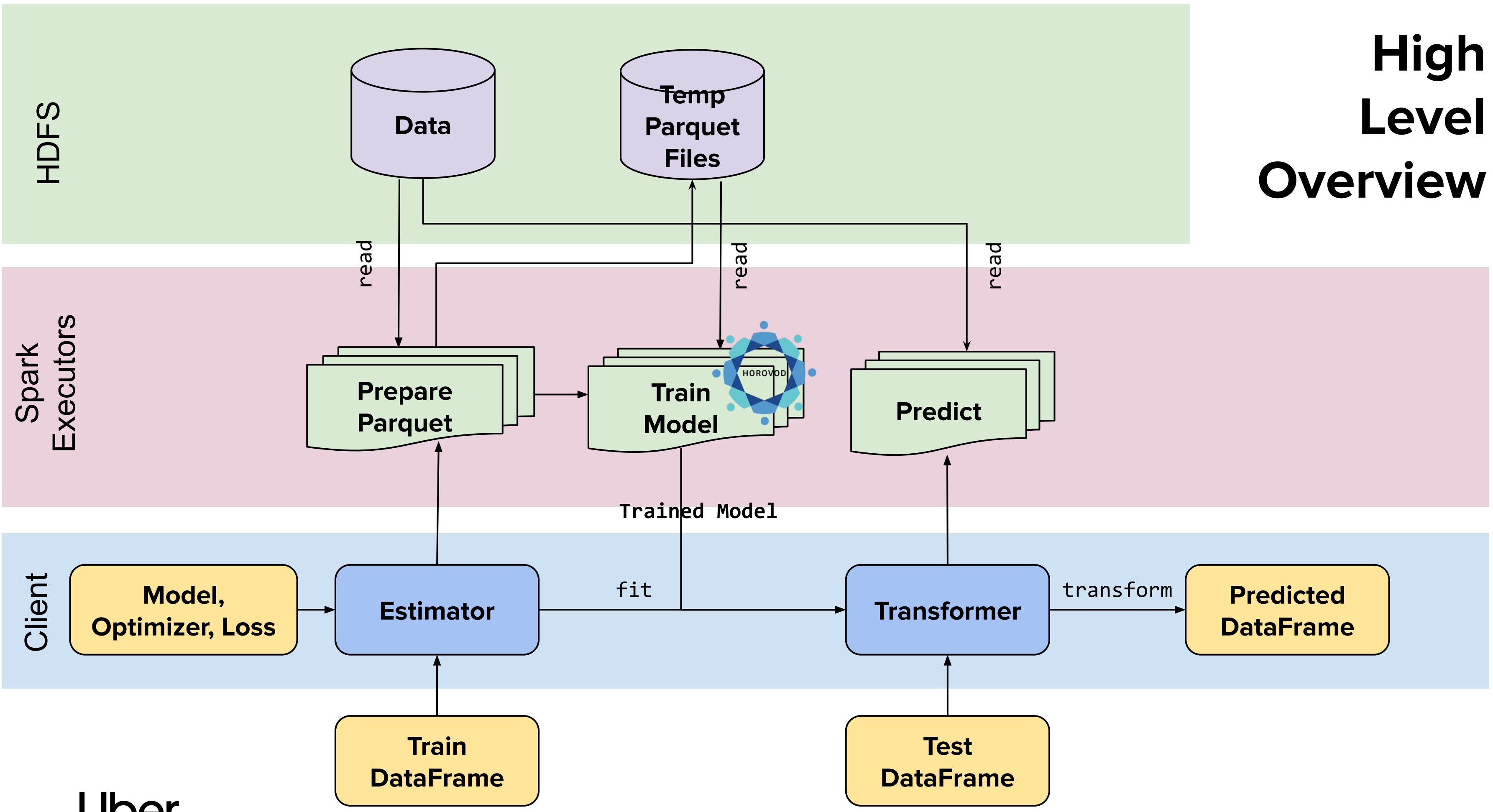
What is under the hood?



+ Apache Spark = ?

How do we combine Deep Learning training with Apache Spark?

High Level Overview



Backend

Data Prep	Deep Learning Model Training	Evaluation	Prediction
Spark	Backend	Spark	Spark, Online, GPU

- Backend should provide
  - `backend.run()` and `backend.num_processes()`
- Internally, we use LambdaDL
- Spark 3 allows resource-aware scheduling. So we can launch the training on GPU instances with Spark.

Uber

Train Function

- Inside the estimator we construct a train function that performs
    - horovod init
    - Data access, reshaping
    - Train loop, loss calculation, metric evaluation

- To execute the train function, it needs to be fully serialized and executed on the backend.

Serialization

Spark ML serialization: serializes the stages into a JSON file.

```
$ pipeline = Pipeline().setStages([])
$ pipeline.write.overwrite.save("sample-pipeline")
$ cat sample-pipeline/metadata/part-00000 | jq
{
  "class": "pyspark.ml.Pipeline",
  "timestamp": 1472747720477,
  "sparkVersion": "2.1.0-SNAPSHOT",
  "uid": "pipeline_181c90b15d65",
  "paramMap": {
    "stageUids": []
  }
}
```

- Model architecture, model weights, and all the param for retraining all serialized
- Keras model and optimizer serialization into JSON
    - Extract the weights and parameters
- Serialization of Torch model and optimizer into JSON
    - Extract the state dict and reconstruct the optimizer

Petastorm: Data Access for Deep Learning Training

Challenges of Training on Large Datasets:

- Shuffling
- Sharding
- Streaming / Buffering / Caching

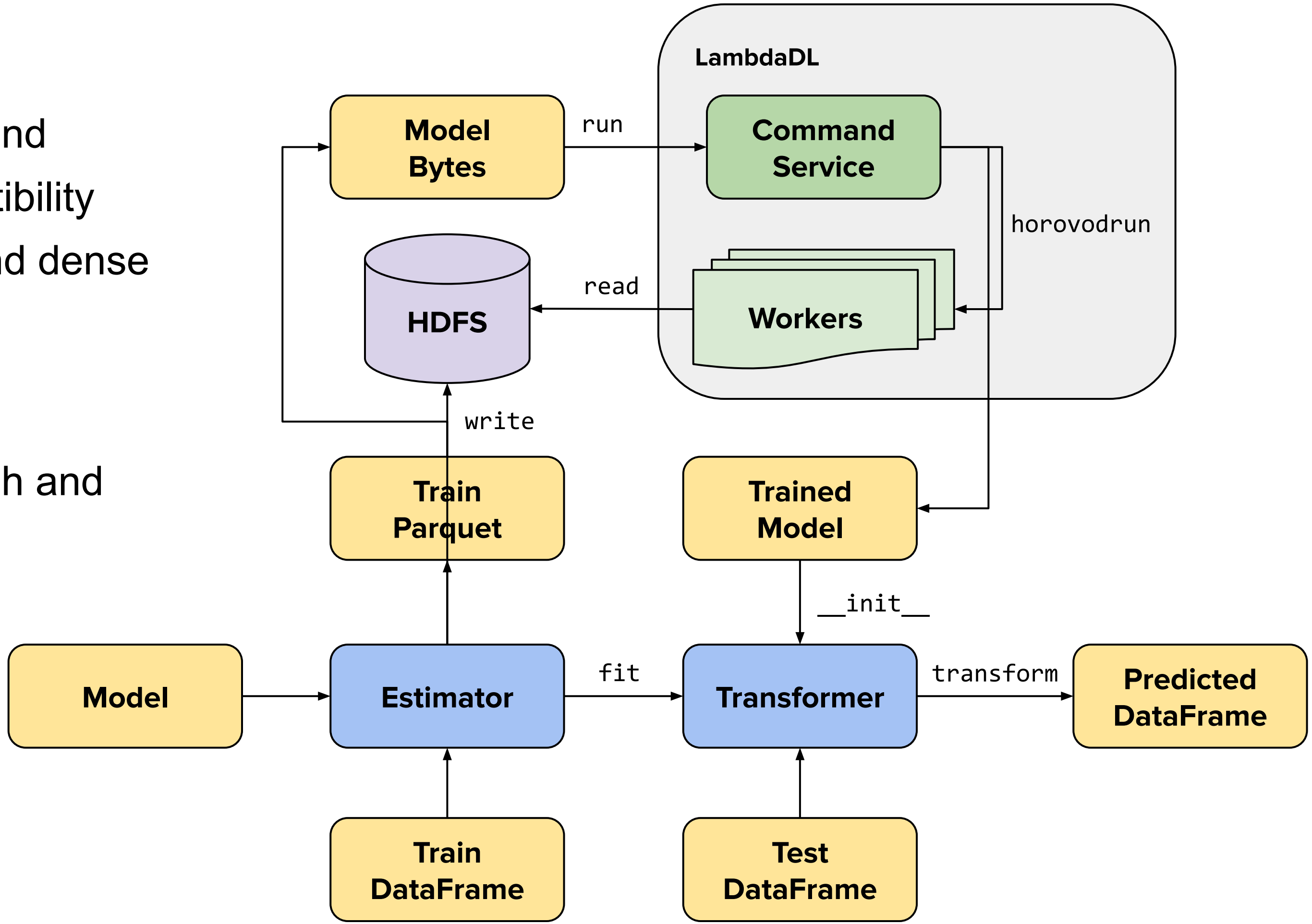
Parquet:

- Large continuous reads (HDFS/S3-friendly)
- Fast access to individual columns
- Faster row queries in some cases
- Written and read natively by Apache Spark
- Spark datatypes and Petastorm compatibility such as sparse and dense vectors

Materializing DF to Parquet on the Store

- Spark datatypes and Petastorm compatibility such as sparse and dense vectors

- Cache the DF hash and reuse it to avoid duplication



Uber

Store

- An interface that provides paths for writing intermediate data.
  - LocalStore
  - HDFSStore
  - MA Workspaces

- During training, we checkpoint model on the store

Success Stories

Case Study: Horovod + Petastorm

Challenges Training on Large Datasets:

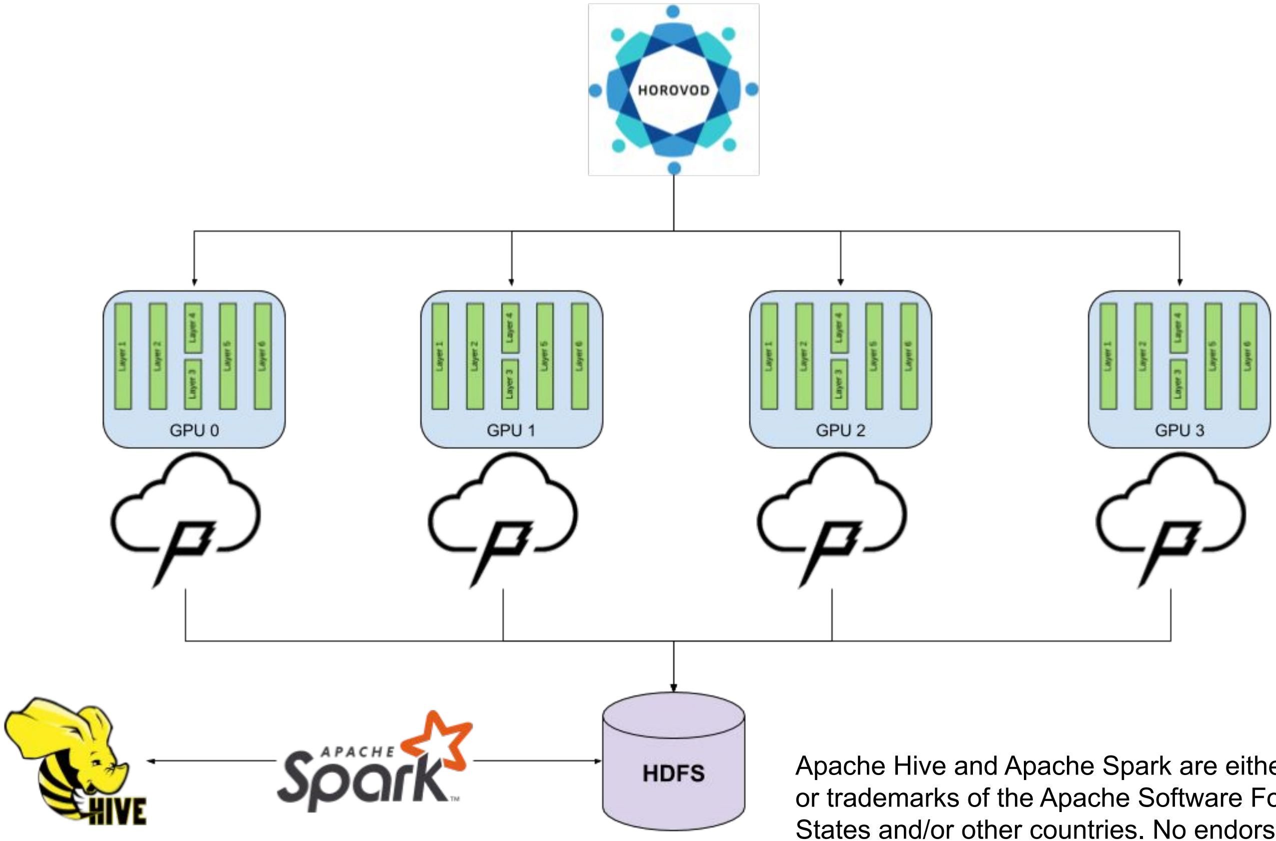
- Shuffling
- Sharding
- Streaming / Buffering / Caching

Parquet:

- Large continuous reads (HDFS/S3-friendly)
- Fast access to individual columns
- Faster row queries in some cases
- Written and read natively by Apache Spark

Uber

Case Study: Horovod + Petastorm



Apache Hive and Apache Spark are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries. No endorsement by The Apache Software Foundation is implied by the use of these marks.

Uber

Horovod in the Community

Used by:

Nvidia

Alibaba

Amazon

Oak Ridge National Laboratory

and many others.

Available in:

AWS

Microsoft Azure

Google Cloud Platform

Databricks Runtime

and more!

Uber

Case Study: Oak Ridge National Laboratory

- Running Horovod on *Summit*, the world's most powerful supercomputer
- Awarded *ACM Gordon Bell Prize* 2018
- *Exascale Deep Learning for Climate Analytics*
    - https://arxiv.org/abs/1810.01993

Credit: SC18 / LBL, http://bit.ly/2xd1uhl

Credit: Carlos Jones / ORNL, http://bit.ly/2YI9bOI

Uber

Case Study: Oak Ridge National Laboratory

- Broke the exaop (1 billion billion calculations / sec) computing barrier
  - **1.13 EF/s** peak, **999.0 PF/s** sustained
  - First time with a deep-learning application
- **4560** Summit nodes, **27360** Volta GPUs, **90.7%** scaling efficiency
- Scaling beyond 1024 GPUs:
  - Hierarchical Allreduce
  - Tensor Fusion
  - FP16 compression

Coming Soon

Elastic Training

- MPI makes it difficult to train with preemptible instances
    - Any instance removal crashes the entire process
    - Cannot add new instances to context at runtime
- Facebook's Gloo (https://github.com/facebookincubator/gloo):
    - Alternative to MPI
    - Raises exception on instance failure
    - Context can be recreated at runtime with new instances
- Challenges:
    - Maintain total batch size as instance count changes
    - Tradeoff frequency of context changes to maximize throughput

Plugin Framework

Horovod as a *research platform*

- Gradient Compression:
  - *Deep Gradient Compression* (https://arxiv.org/abs/1712.01887)
  - *Quantized SGD* (https://arxiv.org/abs/1610.02132)
- Gradient Sparsification:
  - *Block Sparse* (https://arxiv.org/abs/1808.03420)
- Reduced Communication Strategies:
  - *Stochastic Gradient Push* (https://arxiv.org/abs/1811.10792)

RFC: https://github.com/horovod/horovod/issues/1157

Thank you!

http://horovod.ai

Horovod on our Eng Blog: https://eng.uber.com/horovod
Michelangelo on our Eng Blog: https://eng.uber.com/michelangelo
ML at Uber on YouTube: http://t.uber.com/ml-meetup

Uber

Uber