



TensorFlowOnSpark

Scalable TensorFlow Learning on Spark Clusters

Lee Yang, Andrew Feng

Yahoo Big Data ML Platform Team

What is TensorFlowOnSpark



Why TensorFlowOnSpark at Yahoo?

- Major contributor to open-source Hadoop ecosystem
 - Originators of Hadoop (2006)
 - An early adopter of Spark (since 2013)
 - Open-sourced CaffeOnSpark (2016)
- Large investment in production clusters
 - Tens of clusters
 - Thousands of nodes per cluster
- Massive amounts of data
 - Petabytes of data

Private ML Clusters

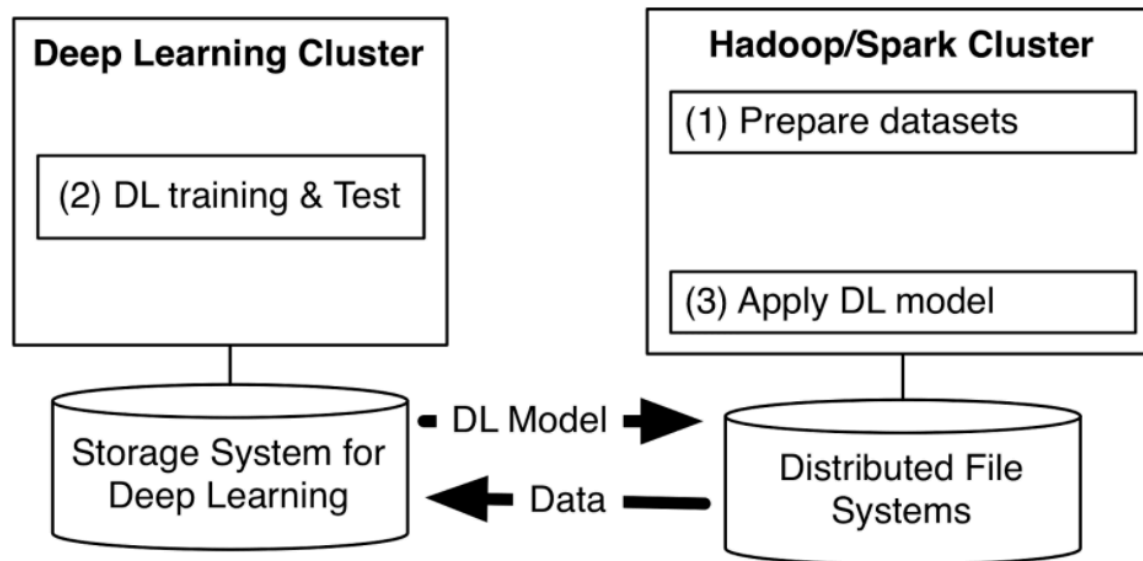


Figure 1: ML Pipeline with multiple programs on separated clusters

Why TensorFlowOnSpark?

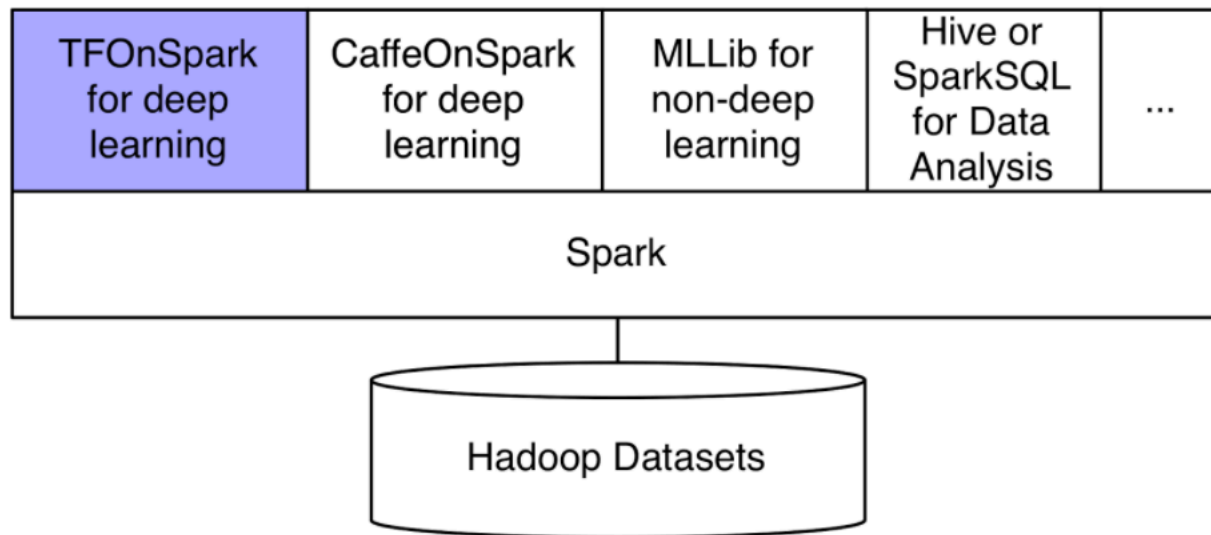
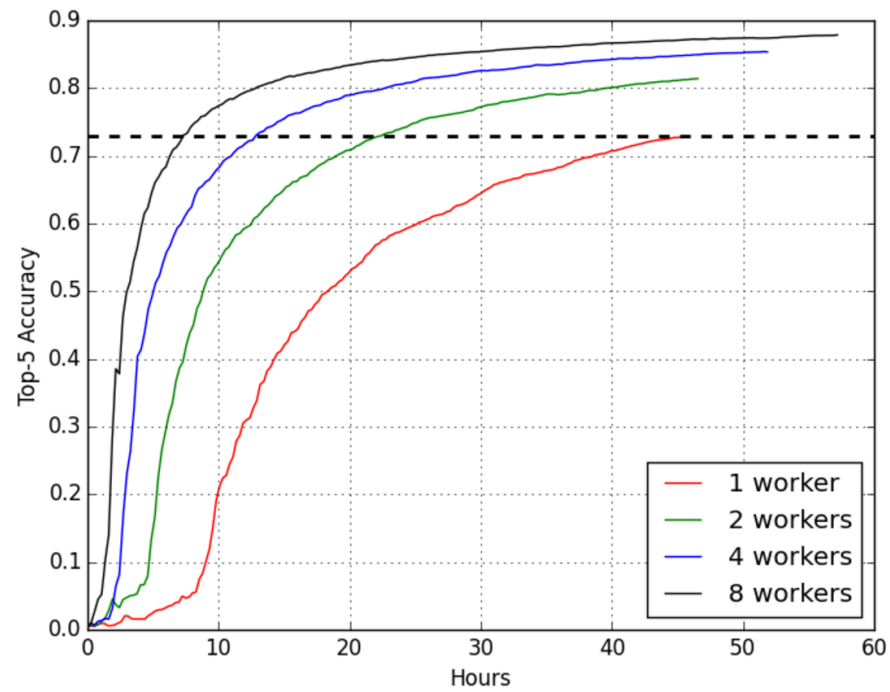


Figure 2: TensorFlowOnSpark for deep learning on Spark clusters

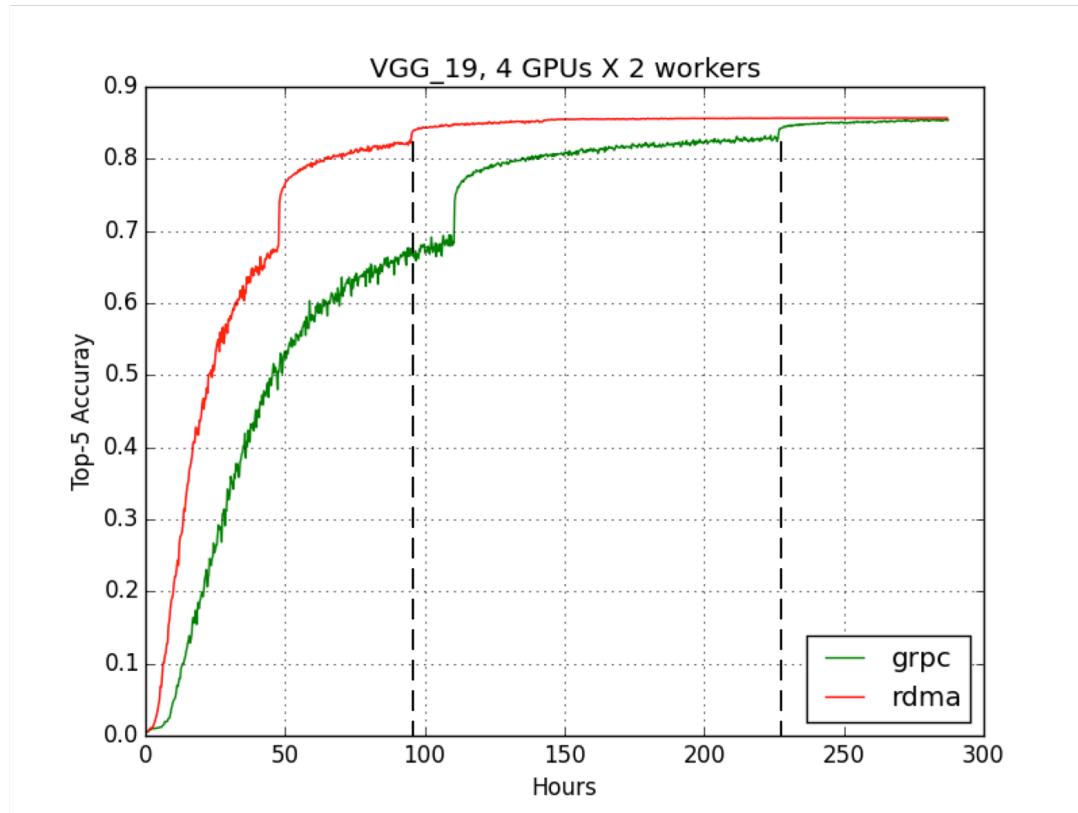
Scaling



Near-linear
scaling

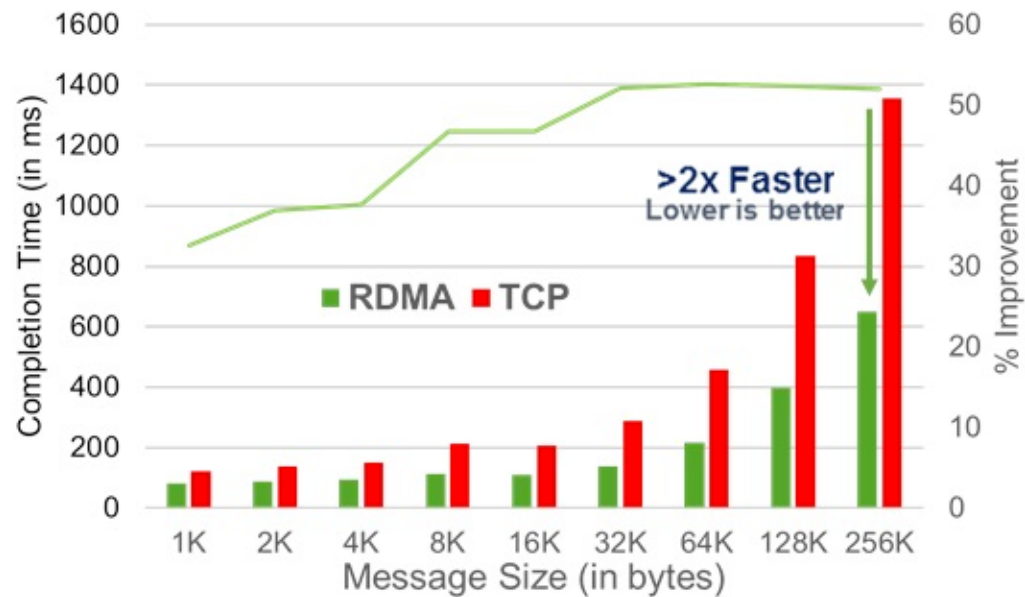
Figure 4: TFoS training of Inception networks

RDMA Speedup over gRPC



2.4X faster

RDMA Speedup over gRPC



<http://www.mellanox.com/solutions/machine-learning/tensorflow.php>

TensorFlowOnSpark Design Goals

- Scale up existing TF apps with minimal changes
- Support all current TensorFlow functionality
 - Synchronous/asynchronous training
 - Model/data parallelism
 - TensorBoard
- Integrate with existing HDFS data pipelines and ML algorithms
 - ex. Hive, Spark, MLlib

TensorFlowOnSpark

- Pyspark wrapper of TF app code
- Launches distributed TF clusters using Spark executors
- Supports TF data ingestion modes
 - feed_dict – RDD.mapPartitions()
 - queue_runner – direct HDFS access from TF
- Supports TensorBoard during/after training
- Generally agnostic to Spark/TF versions

Supported Environments

- Python 2.7 - 3.x
- Spark 1.6 - 2.x
- TensorFlow 0.12, 1.x
- Hadoop 2.x

Architectural Overview

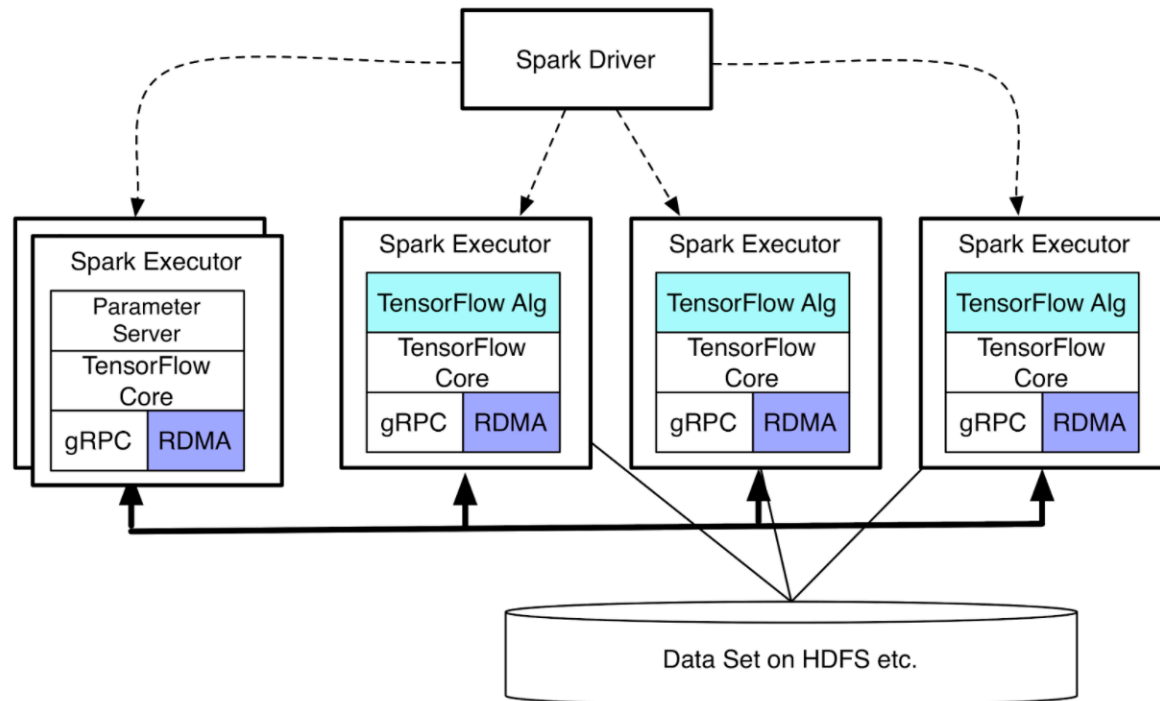


Figure 3: TensorFlowOnSpark system architecture

TensorFlowOnSpark Basics

1. **Launch** TensorFlow cluster
2. **Feed data** to TensorFlow app
3. **Shutdown** TensorFlow cluster

API Example

```
cluster = TFCluster.run(sc, map_fn, args, num_executors,  
num_ps, tensorboard, input_mode)  
cluster.train(dataRDD, num_epochs=0)  
cluster.inference(dataRDD)  
cluster.shutdown()
```

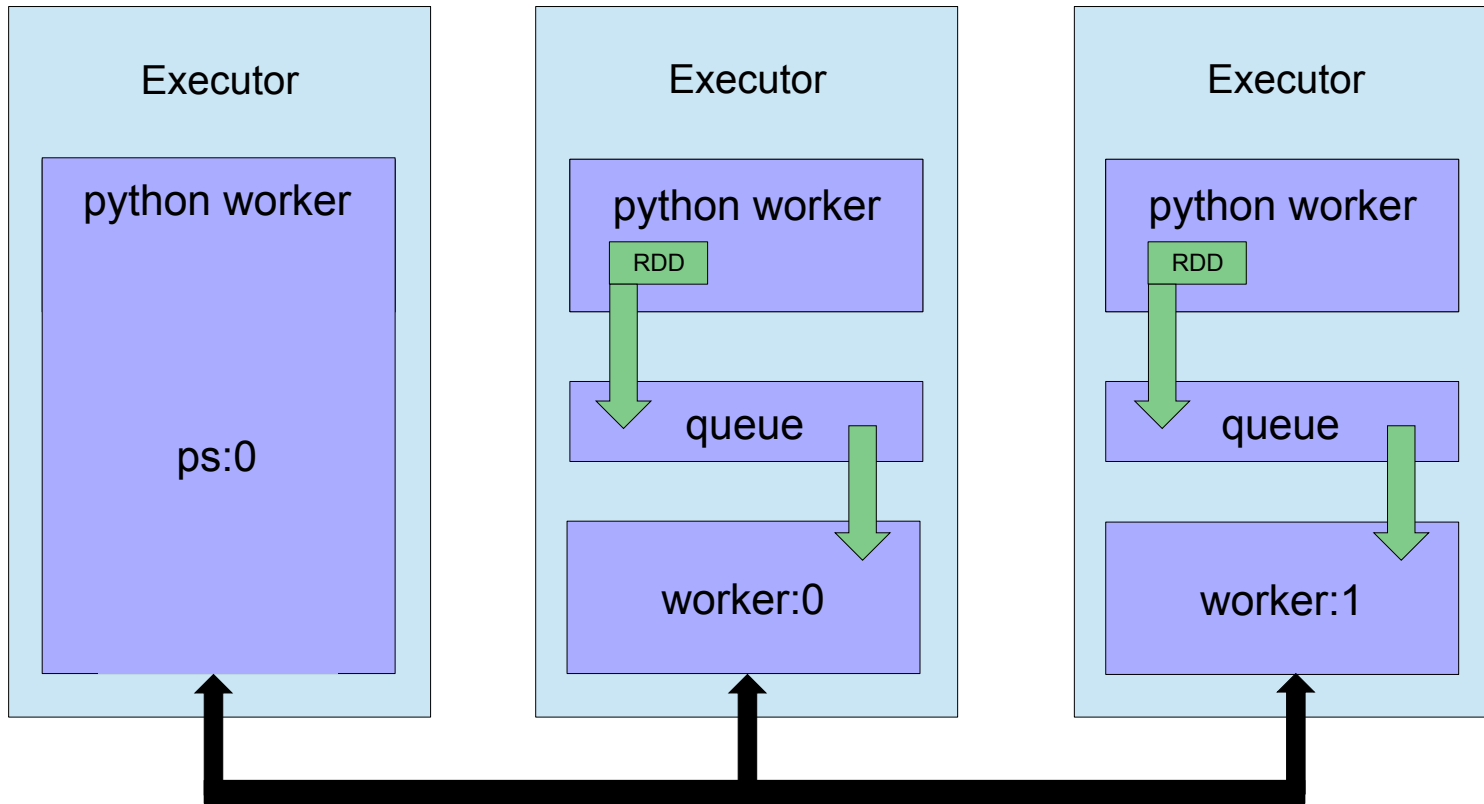
Conversion Example

```
# diff -w eval_image_classifier.py
20a21,27
> from pyspark.context import SparkContext
> from pyspark.conf import SparkConf
> from tensorflowonspark import TFCluster, TFNode
> import sys
>
> def main_fun(argv, ctx):
27a35,36
>     sys.argv = argv
>
84,85d92
<
< def main(_):
88a96,97
>     cluster_spec, server = TFNode.start_cluster_server(ctx)
>
191c200,204
<     tf.app.run()
---
>     sc = SparkContext(conf=SparkConf().setAppName("eval_image_classifier"))
>     num_executors = int(sc._conf.get("spark.executor.instances"))
>     cluster = TFCluster.run(sc, main_fun, sys.argv, num_executors, 0, False, TFCluster.InputMode.TENSORFLOW)
>     cluster.shutdown()
```

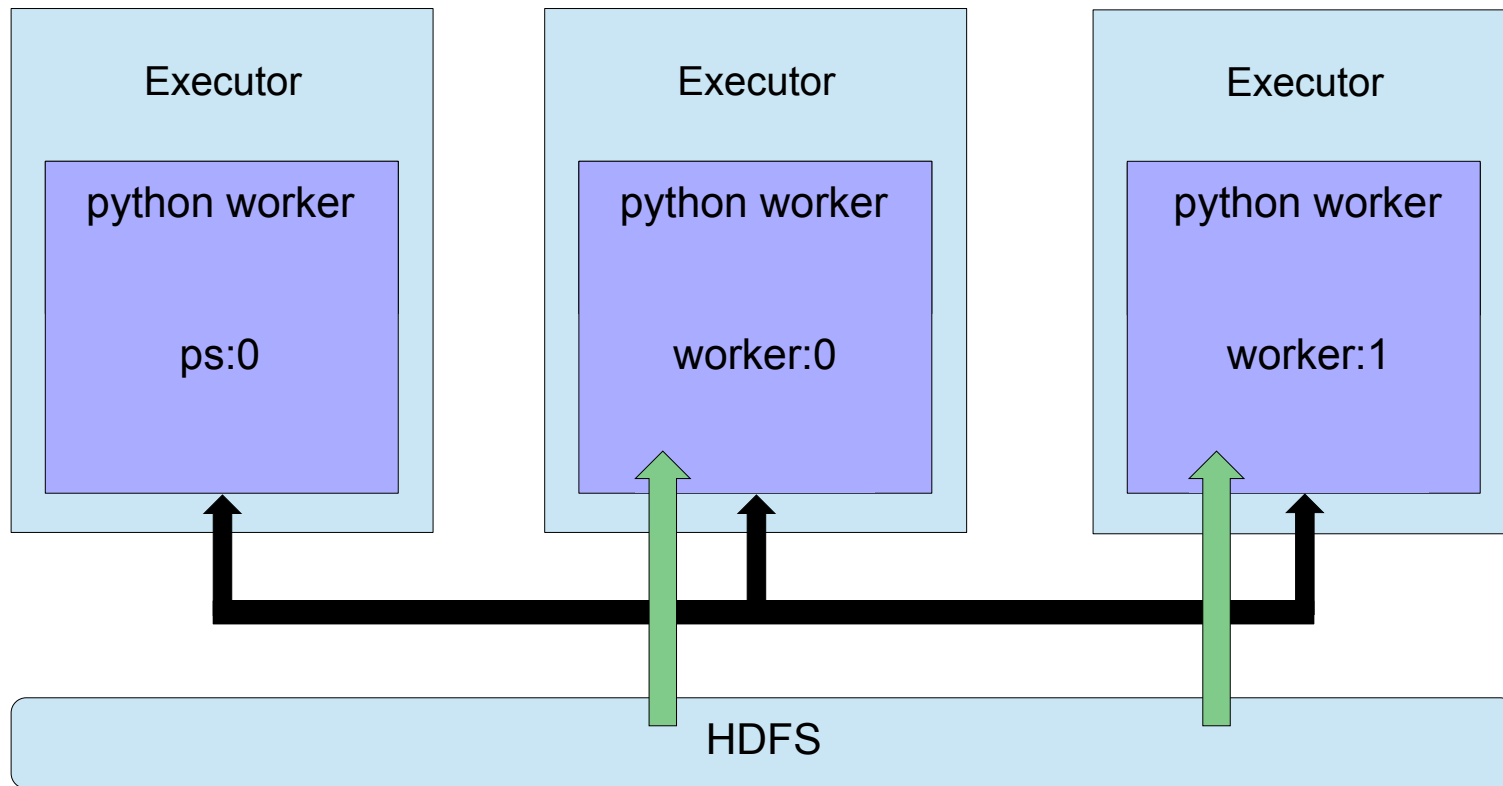
Input Modes

- `InputMode.SPARK`
HDFS → `RDD.mapPartitions` → `feed_dict`
- `InputMode.TENSORFLOW`
`TFReader + QueueRunner` ← HDFS

InputMode.SPARK



InputMode.TENSORFLOW



Failure Recovery

- TF Checkpoints written to HDFS
- InputMode.SPARK
 - TF worker runs in background
 - RDD data feeding tasks can be retried
 - However, TF worker failures will be “hidden” from Spark
- InputMode.TENSORFLOW
 - TF worker runs in foreground
 - TF worker failures will be retried as Spark task
 - TF worker restores from checkpoint

Failure Recovery

- Executor failures are problematic
 - e.g. pre-emption
 - TF cluster_spec is statically-defined at startup
 - YARN does not re-allocate on same node
 - Even if possible, port may no longer be available.
- Need dynamic cluster membership
 - Exploring options w/ TensorFlow team

TensorBoard

The screenshot displays the TensorBoard interface with an orange header bar containing navigation tabs: SCALARS, IMAGES, AUDIO, GRAPHS, DISTRIBUTIONS, HISTOGRAMS, and EMBEDDINGS. On the left, a sidebar provides controls: 'Fit to screen', 'Download PNG', 'Run (1)', 'Session runs (0)', 'Upload Choose File', 'Trace inputs', 'Color' (Structure selected), and a 'Graph' legend. The main area shows a computational graph with nodes: 'Adagrad', 'global_step', 'gradients', 'metric', 'softmax_layer', and 'hidden_layer'. The graph illustrates dataflow (solid lines) and control dependencies (dotted lines) between these operations. A legend at the bottom left defines symbols: Namespace* (grey box), OpNode (white oval), Unconnected series* (white oval with border), Connected series* (white oval with border), Constant (white circle), Summary (black square with 'ii'), Dataflow edge (solid line), Control dependency edge (dotted line), and Reference edge (arrow).

TensorBoard

The screenshot displays the TensorBoard web interface. At the top, there is an orange navigation bar with the 'TensorBoard' logo and several menu items: SCALARS, IMAGES, AUDIO, GRAPHS, DISTRIBUTIONS, HISTOGRAMS, and EMBEDDINGS. On the right side of this bar are icons for refresh, settings, and help.

On the left side, there is a sidebar with several controls:

- A search box: "Write a regex to create a tag group" with a close button (X).
- Two checkboxes: "Split on underscores" and "Data download links", both currently unchecked.
- A dropdown menu for "Tooltip sorting method" set to "default".
- A "Smoothing" slider set to 0.6.
- A "Horizontal Axis" section with three buttons: "STEP" (selected), "RELATIVE", and "WALL".
- A "Runs" section with a search box "Write a regex to filter runs" and a single run selected, indicated by a red checkmark and a red circle icon.
- A "TOGGLE ALL RUNS" button.
- The text "tensorboard_1" at the bottom of the sidebar.

The main content area shows a table of runs with the following data:

global_step	1
metric	2

Below the table, two line graphs are displayed side-by-side:

- The left graph is titled "metric/accuracy/accuracy". The y-axis ranges from 0.450 to 0.950. The x-axis ranges from 0.000 to 6.000k. The plot shows a sharp increase in accuracy from approximately 0.450 at 0.000k steps to about 0.900 by 1.000k steps, after which it remains relatively stable with minor fluctuations around 0.900.
- The right graph is titled "metric/loss/loss". The y-axis ranges from 20.0 to 140.0. The x-axis ranges from 0.000 to 6.000k. The plot shows a sharp decrease in loss from approximately 140.0 at 0.000k steps to about 20.0 by 1.000k steps, after which it remains relatively stable with minor fluctuations around 20.0.

Each graph has a refresh icon and a menu icon (three horizontal lines) below it.

TensorBoard

The screenshot displays the TensorBoard web interface. At the top, an orange navigation bar contains the 'TensorBoard' logo and several tabs: 'SCALARS', 'IMAGES', 'AUDIO', 'GRAPHS', 'DISTRIBUTIONS', 'HISTOGRAMS', and 'EMBEDDINGS'. On the right side of this bar are icons for refresh, settings, and help. Below the navigation bar, the main content area is titled 'image' and shows three columns of image visualizations. Each column is labeled 'image/image/0', 'image/image/1', and 'image/image/2' respectively, with a timestamp 'step 6401 (Tue Mar 21 2017 11:02:11 GMT-0700 (PDT))' below each label. The first two columns show a handwritten digit '7', while the third shows a handwritten digit '0'. Each image has a small icon at the bottom left for zooming. On the left side of the interface, there is a sidebar with a search box 'Write a regex to create a tag group', a checkbox 'Split on underscores', and a 'Runs' section with a filter box 'Write a regex to filter runs' and a checked checkbox. At the bottom left, there is a 'TOGGLE ALL RUNS' button and the text 'tensorboard_1'.

TensorFlow App Development

Experimentation Phase

- Single-node
- Small scale data
- TensorFlow APIs

`tf.Graph`

`tf.Session`

`tf.InteractiveSession`

TensorFlow App Development

Scaling Phase

- Multi-node
- Medium scale data (local disk)
- Distributed TensorFlow APIs

`tf.train.ClusterSpec`

`tf.train.Server`

`tf.train.Saver`

`tf.train.Supervisor`

TensorFlow App Development

Production Phase

- Cluster deployment
- Upstream data pipeline
- Model training w/ TensorFlowOnSpark APIs
 - `TFCluster.run`
 - `TFNode.start_cluster_server`
 - `TFCluster.shutdown`
- Production inference w/ TensorFlow Serving

Example Usage

<https://github.com/yahoo/TensorFlowOnSpark/tree/master/examples>

Common Gotchas

- Single task (TF node) per executor
- HDFS access (native libs/env)
- Why doesn't algorithm X scale linearly?

What's New?

- Community contributions
 - CDH compatibility
 - TFNode.DataFeed
 - Bug fixes
- RDMA merged into TensorFlow repository
- Registration server
- Spark streaming
- Pip packaging

Spark Streaming

```
from pyspark.streaming import StreamingContext
ssc = StreamingContext(sc, 10)
images = sc.textFile(args.images).map(lambda ln: parse(ln))
stream = ssc.textFileStream(args.images)
imageRDD = stream.map(lambda ln: parse(ln))
cluster = TFCluster.run(sc, map_fun, args,...)
predictionRDD = cluster.inference(imageRDD)
predictionRDD.saveAsTextFile(args.output)
predictionRDD.saveAsTextFiles(args.output)
ssc.start()
cluster.shutdown(ssc)
```

Pip packaging

```
pip install tensorflowonspark
${SPARK_HOME}/bin/spark-submit \
  --master ${MASTER} \
  --py-files ${TFoS_HOME}/examples/mnist/spark/mnist_dist.py \
  --archives ${TFoS_HOME}/tfspark.zip \
  ${TFoS_HOME}/examples/mnist/spark/mnist_spark.py \
  --cluster_size ${SPARK_WORKER_INSTANCES} \
  --images examples/mnist/csv/train/images \
  --labels examples/mnist/csv/train/labels \
  --format csv \
  --mode train \
  --model mnist_model
```



https://github.com/yahoo/TensorFlowOnSpark/wiki/GetStarted_Standalone

Next Steps

- TF/Keras Layers
- Failure recovery w/ dynamic cluster management (e.g. registration server)

Summary

TFoS brings deep-learning to big-data clusters

- TensorFlow: 0.12 -1.x
- Spark: 1.6-2.x
- Cluster manager: YARN, Standalone, Mesos
- EC2 image provided
- RDMA in TensorFlow

Thanks!



YAHOO!

And our open-source contributors!

Questions?

<https://github.com/yahoo/TensorFlowOnSpark>