Deep Learning Frameworks with Spark and GPUs

Pierce Spitler
Data Scientist

Tim Gasper
Director, Product

bitfusion

The Deep Learning Landscape

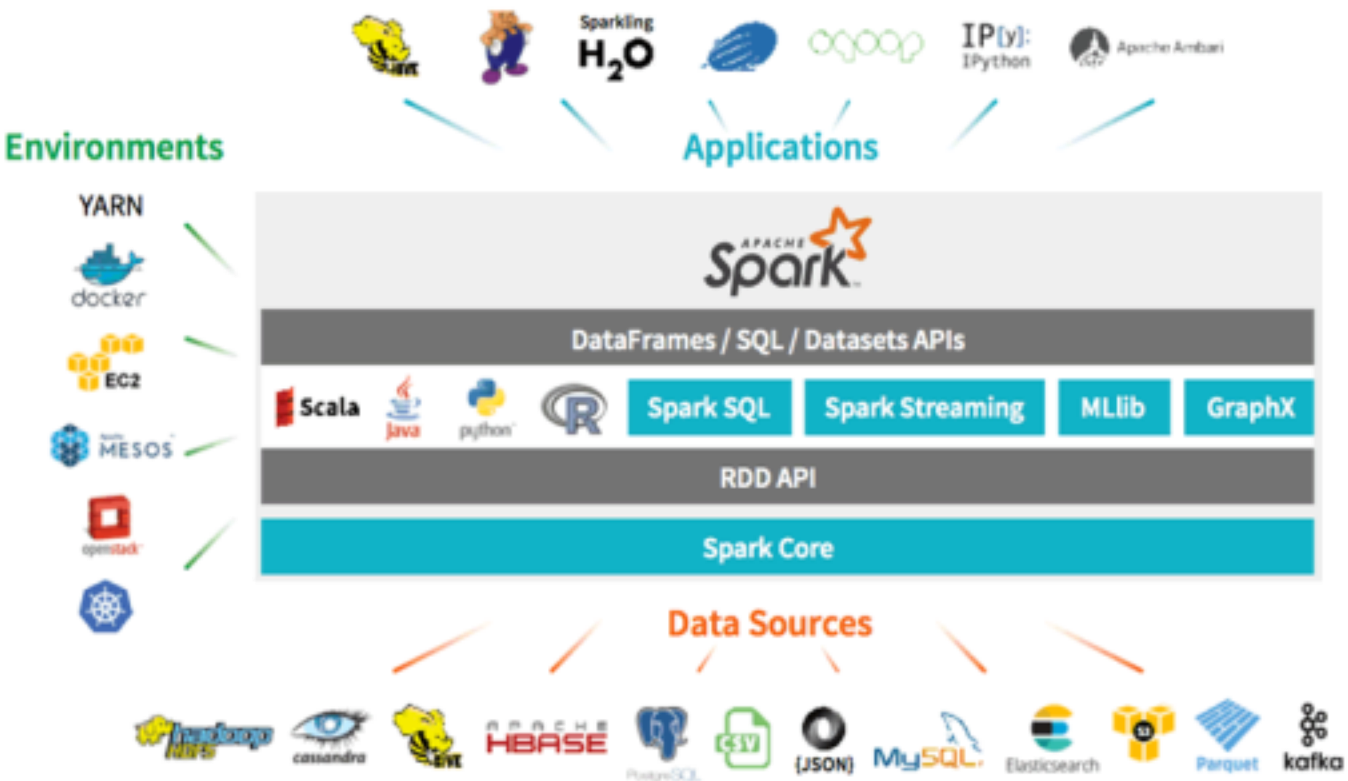
Frameworks - Tensorflow, MXNet, Caffe, Torch, Theano, etc

Boutique Frameworks - TensorflowOnSpark, CaffeOnSpark

Data Backends - File system, Amazon EFS, Spark, Hadoop, etc, etc.

Cloud Ecosystems - AWS, GCP, IBM Cloud, etc, etc

Why Spark



Ecosystem

- Many…
  - Data sources
  - Environments
  - Applications

Real-time data

- In-memory RDDs (resilient distributed data sets)
- HDFS integration

Data Scientist Workflow

- DataFrames
- SQL
- APIs
- Pipelining w/ job chains or Spark Strea
- Python, R, etc.

Approaches to Deep Learning + Spark

Yahoo: CaffeOnSpark, TensorFlowOnSpark

Designed to run on existing Spark and Hadoop clusters, and use existing Spark libraries like SparkSQL or Spark's MLlib machine learning libraries

Should be noted that the Caffe and TensorFlow versions used by these lag the release version by about 4-6 weeks

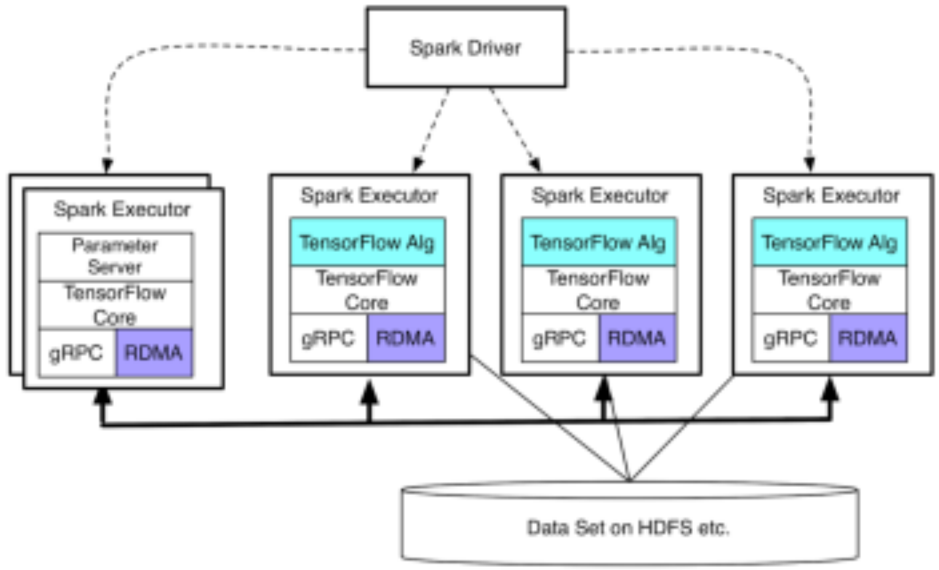


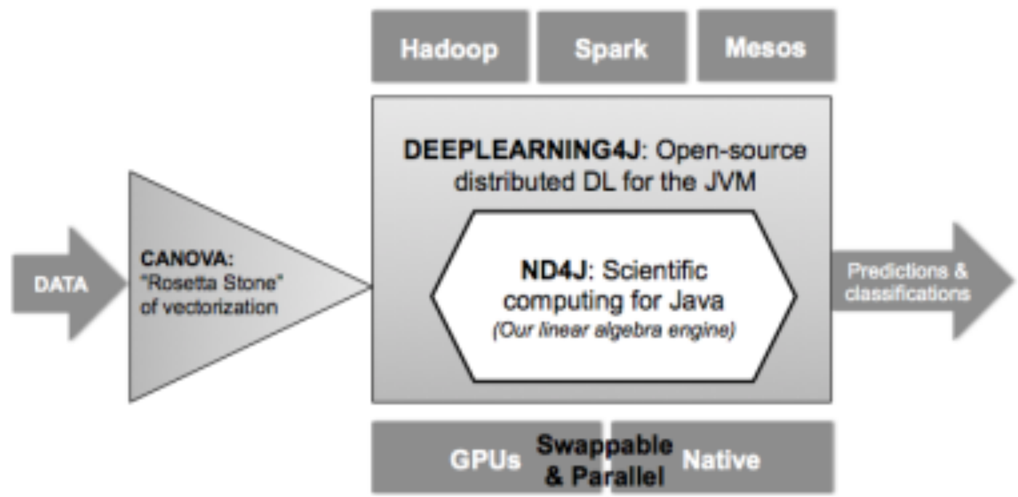
Figure 3: TensorFlowOnSpark system architecture

Source: Yahoo

Skymind.ai: DeepLearning4J

Deeplearning4j is an open-source, distributed deep-learning library written for Java and Scala.

DL4J can import neural net models from most major frameworks via Keras, including TensorFlow, Caffe, Torch and Theano. Keras is employed as Deeplearning4j's Python API.

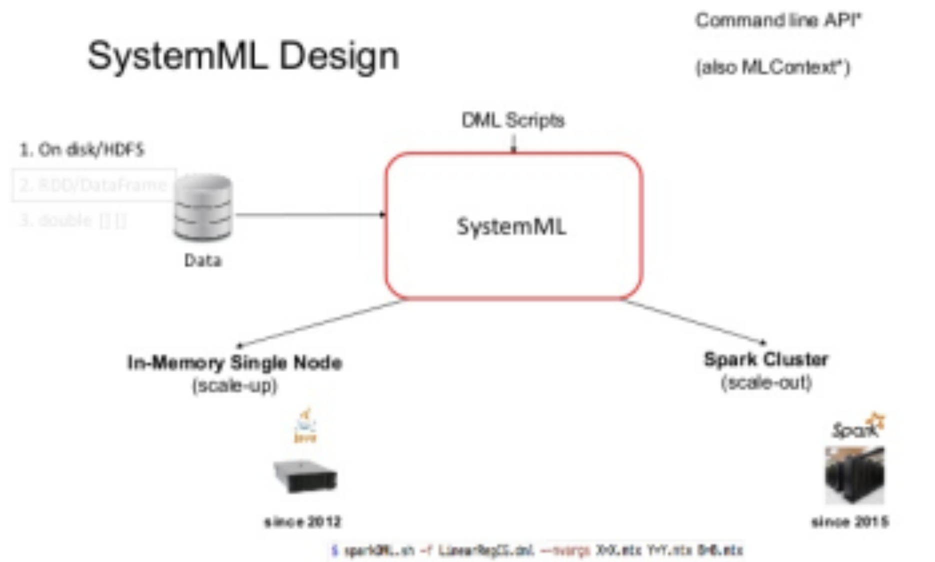


Source: deeplearning4j.org

IBM: SystemML

Apache SystemML runs on top of Apache Spark, where it automatically scales your data, line by line, determining whether your code should be run on the driver or an Apache Spark cluster.

- Algorithm customizability via R-like and Python-like languages.
- Multiple execution modes, including Spark MLContext, Spark Batch, Hadoop Batch, Standalone, and JMLC.
- Automatic optimization based on data and cluster characteristics to ensure both efficiency and scalability.
- Limited set of algorithms supported so far



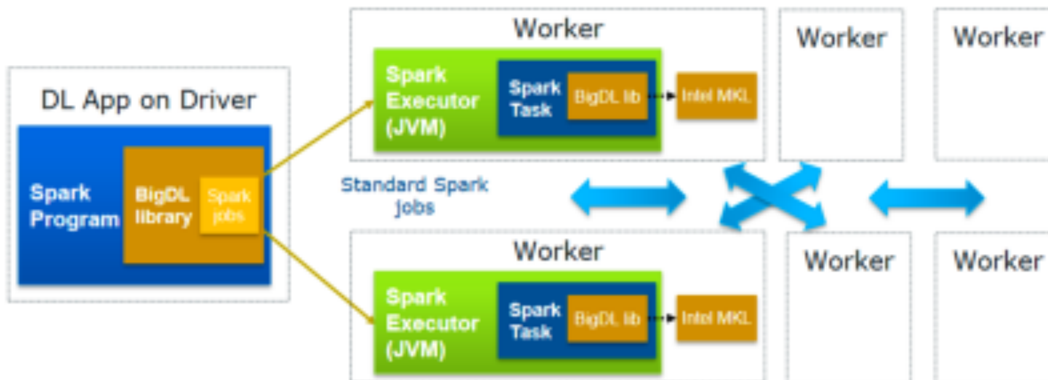
Source: Niketan Panesar

Intel: BigDL

Modeled after Torch, supports Scala and Python programs

Scales out via Spark

Leverages IBM MKL (Math Kernel Library)



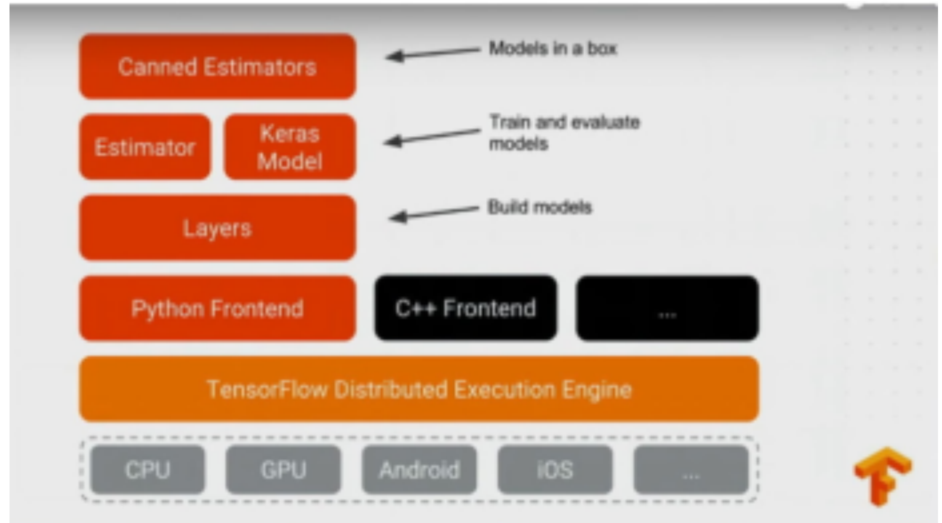
Source: MSDN

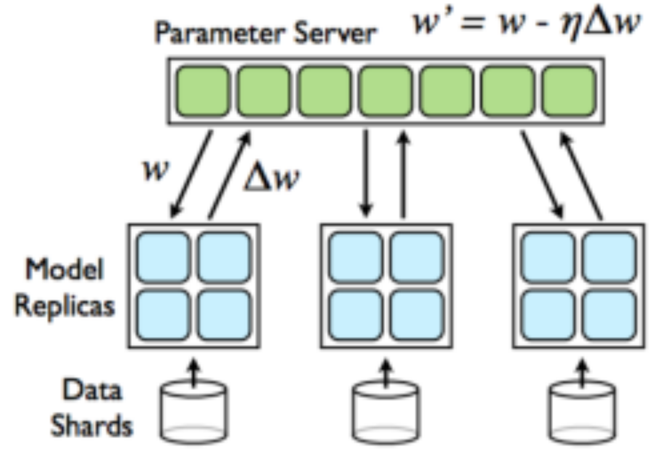
Google: TensorFlow

Flexible, powerful deep learning framework that supports CPU, GPU, multi-GPU, and multi-server GPU with Tensorflow Distributed

Keras support

Strong ecosystem (we'll talk more about this)



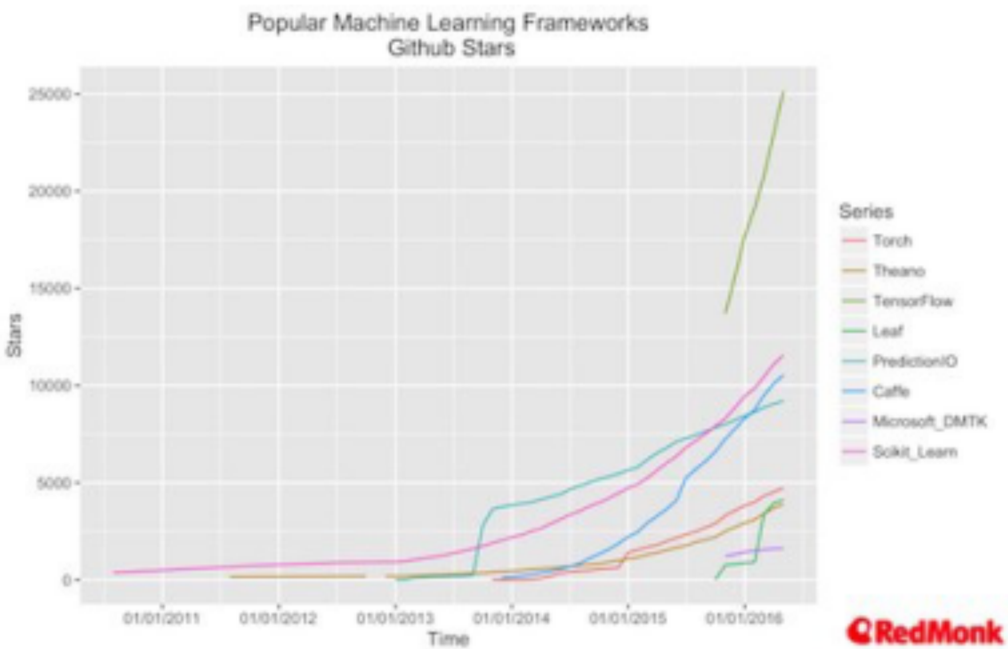


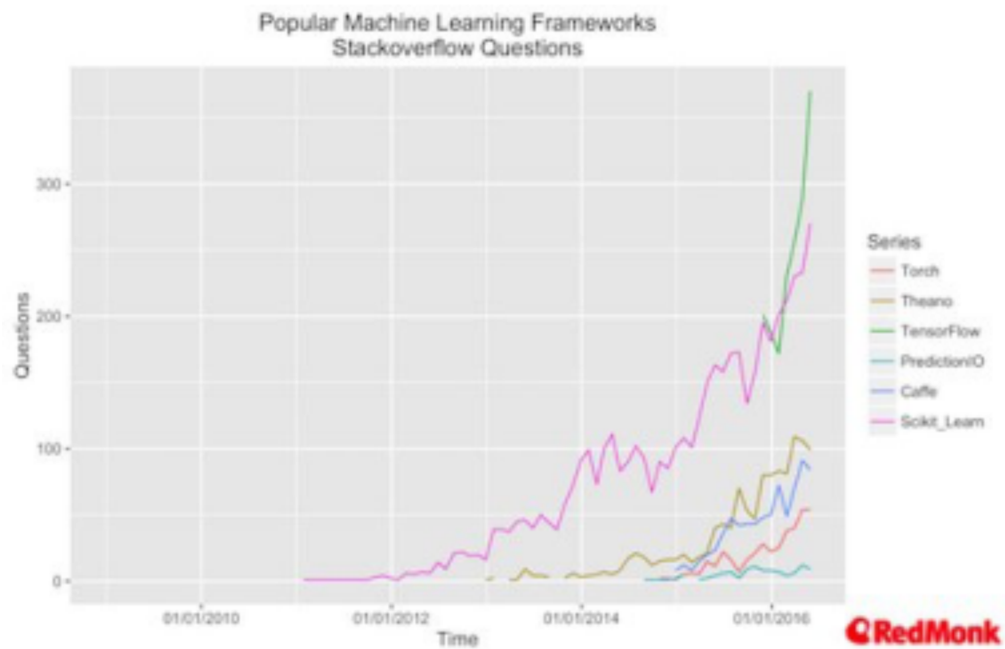
Source: Google

Why We Chose TensorFlow for Our Study

TensorFlow’s Web Popularity

We decided to focus on TensorFlow as it represent the majority the deep learning framework usage in the market right now.



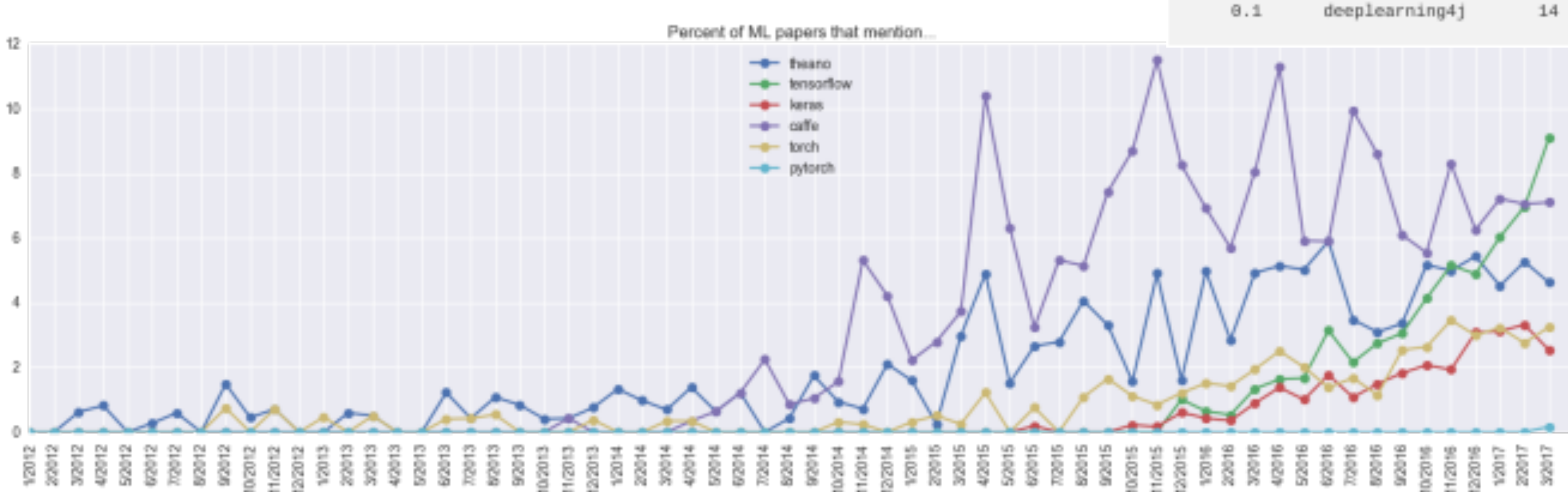


Source: http://redmonk.com/fryan/2016/06/06/a-look-at-popular-machine-learning-frameworks/

TensorFlow’s Academic Popularity

It is also worth noting that TensorFlow represents a large portion of what the research community is currently interested in.

% of papers	framework	has been around for (months)
9.1	tensorflow	16
7.1	caffe	37
4.6	theano	54
3.3	torch	37
2.5	keras	19
1.7	matconvnet	26
1.2	lasagne	23
0.5	chainer	16
0.3	mxnet	17
0.3	cntk	13
0.2	pytorch	1
0.1	deeplearning4j	14



Source: https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106

A Quick Case Study

Model and Benchmark Information

Our model uses the following:

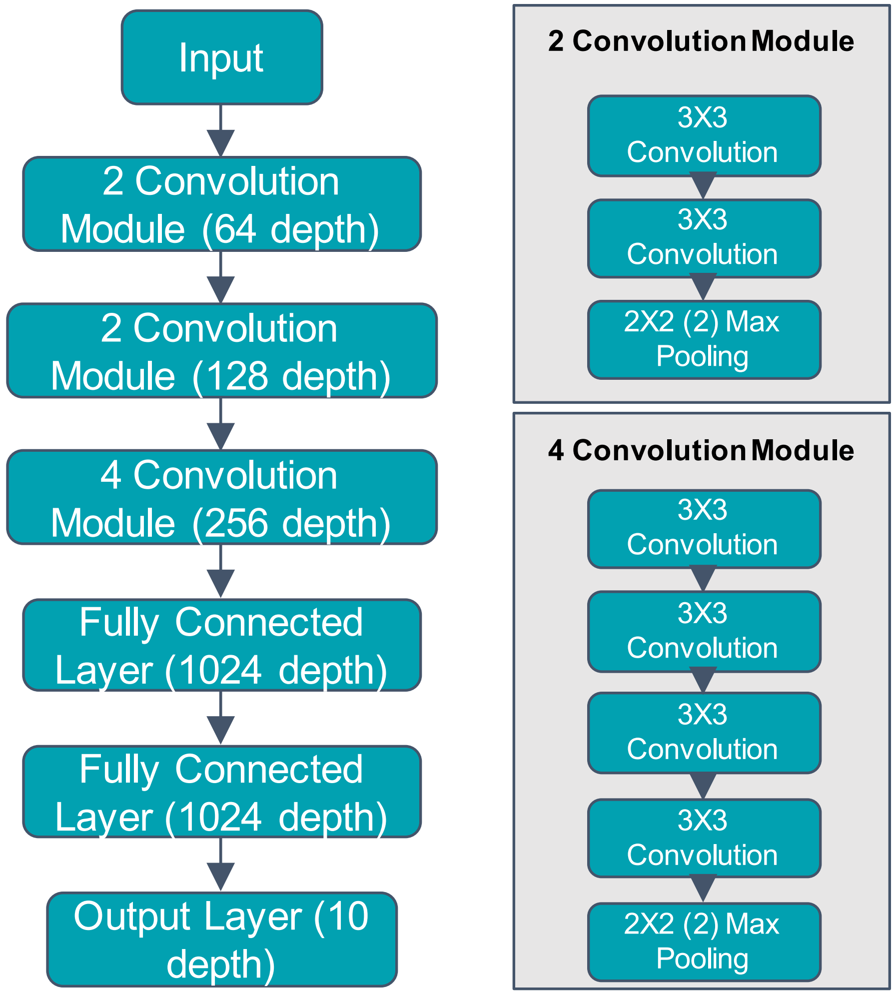
- Dataset: CIFAR10
- Architecture: Pictured to the right
- Optimizer: Adam

For benchmarking, we will consider:

- Images per second
- Time to 85% accuracy (averaged across 10 runs)
- All models run on p2 AWS instances
- Hardware is homogenous!!!

Model Configurations:

- Single server - CPU
- Single server - GPU
- Single server - multi-GPU
- Multi server distributed TensorFlow
- Multi server TensorFlow on Spark



TensorFlow - Single Server CPU and GPU

- This is really well documented and the basis for why most of the frameworks were created. Using GPUs for deep learning creates high returns quickly.

- Managing dependencies for GPU-enabled deep learning frameworks can be tedious (cuda drivers, cuda versions, cudnn versions, framework versions). Bitfusion can help alleviate a lot of these issues with our AMIs and docker containers

- When going from CPU to GPU, it can be hugely beneficial to explicitly put data tasks on CPU and number crunching (gradient calculations) on GPUs with tf.device().

Performance

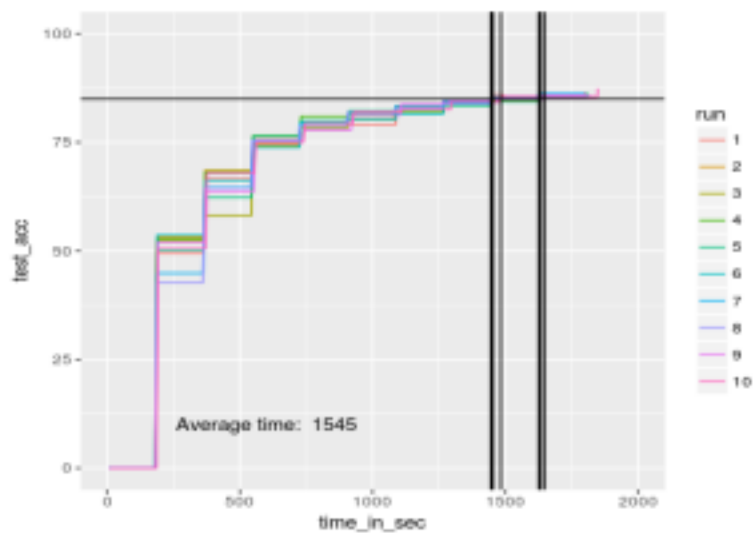
CPU:

- Images per second ~ 40
- Time to 85% accuracy ~ 50500

GPU:

- Images per second ~ 1420
- Time to 85% accuracy ~ 1545 sec

Note: CPU can be sped up on more CPU focused machines.



TensorFlow - Single Server Multi-GPU

Implementation Details

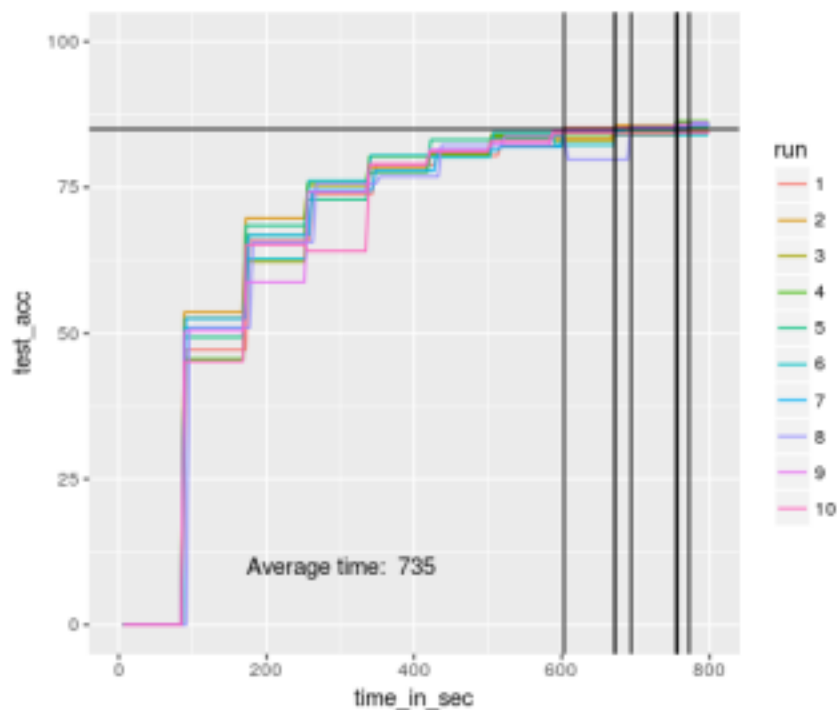
- For this example we used 3 GPUs on a single machine (p2.8xlarge)

Code Changes

- Need to write code to define device placement with tf.device()
- Write code to calculate gradients on each GPU and then calculates an average gradient for the update

Performance

- Images per second ~ 3200
- Time to 85% accuracy ~ 735 sec



Distributed TensorFlow

Implementation Details

For this example we used 1 parameter server and 3 worker servers.

We used EFS, but other shared file systems or native file systems could be used

Code Changes

Need to write code to define a parameter server and worker server

Need to implement special session: tf.MonitoredSession()

Either a cluster manager needs to be set up or jobs need to be kicked off individually on workers and parameter servers

Need to balance batch size and learning rate to optimize throughput

Performance

- Images per second ~ 1630
- Time to 85% accuracy ~ 1460

TensorFlow on Spark

Implementation Details

For this example we used 1 parameter server and 3 worker servers.

Requires data to be placed in HDFS

Code Changes

TensorFlow on Spark requires renaming a few TF variables from a vanilla distributed TF implementation. (tf.train.Server() becomes TFNode.start_cluster_server(), etc.)

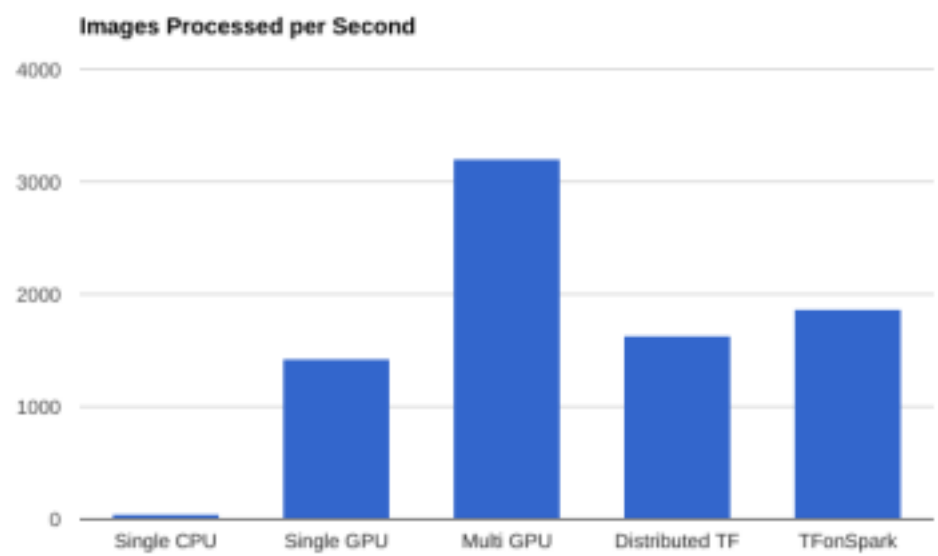
Need to understand multiple utils for reading/writing to HDFS

Performance

- Images per second ~ 1970
- Time to 85% accuracy ~ 1210

Performance Summary

- It should be noted again that the CPU performance is on a p2.xlarge
- For this example, local communication through multi-GPU is much more efficient than distributed communication
- TensorFlow on Spark provided speedups in the distributed setting (most likely from RDMA)
- Tweaking of the batch size and learning rate were required to optimize throughput



When to Use What and Practical Considerations

Practical Considerations Continued

Number of Updates

- If the number of updates and the size of the updates are considerable, multiple GPU configurations on a single server should be

Hardware Configurations

- Network speeds play a crucial role in distributed model settings
- GPUs connected with RDMA over Infiniband have shown significant speedup over more basic gRPC over ethernet
- IB was 2X faster for 2 nodes, 10X for 4 nodes versus TCP (close in performance to gRPC)

gRPC vs RDMA vs MPI

gRPC and MPI are possible transports for distributed TF, however:

- gRPC latency is 100-200us/msg due to userspace <-> kernel switches
- MPI latency is 1-3us/message due to OS bypass, **but** requires communications to be serialized to a single thread. Otherwise, 100-200us latency
- TensorFlow typically spawns ~**100 threads** making MPI suboptimal

Our approach, powered by **Bitfusion Core** compute virtualization engine, is to use native **IB verbs + RDMA** for the primary transport

- **1-3 us per message** across all threads of execution
- Use **multiple channels simultaneously** where applicable: PCIe, multiple IB ports
- Fall back to TCP/IP where fast transports not available

Practical Considerations

Size of Data

- If the size of the input data is especially large (large images for example) the entire model might not fit onto a single GPU
- If the number of records is prohibitively large a shared file system or database might be required
- If the number of records is large convergence can be sped up using multiple GPUs or distributed models

Size of Model

- If the size of the network is larger than your GPU's memory, splitting the model across multi-GPUs (model parallel)

Key Takeaways and Future Work

Key Takeaways

- GPUs are almost always better
- Hardware/Network setup matters a lot in distributed settings!
- Saturating GPUs locally should be a priority before going to a distributed setting
- If your data is already built on Spark, TensorFlow on Spark provides an easy way to integrate with your current data stack
- RDMA and infiniband is natively supported by TensorFlow on Spark

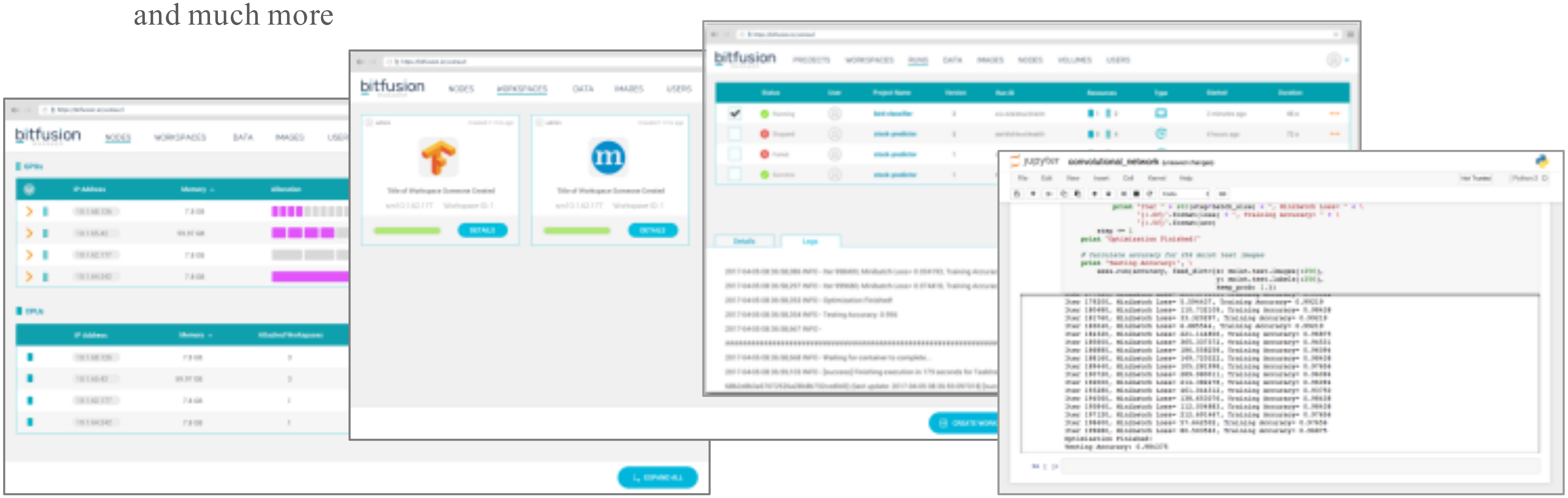
Future Work

- Use TensorFlow on Spark on our Infiniband cluster
- Continue to assess the current state of the art in deep learning
- Assess various cloud/hardware configurations to optimize performance

Bitfusion Flex

End-to-end, elastic infrastructure for building deep learning and AI applications

- Can utilize RDMA and Infiniband when remote GPUs are attached
- Never have to transition beyond multi-GPU code environment (remote servers look like local GPUs and can be utilized without code changes)
- Rich CLI & GUI, interactive Jupyter workspaces, batch scheduling, smart shared resourcing for max efficiency, and much more




INSIGHTS
SQL
STORAGE
IT
AZURE
CLOUD
SECURITY
BIG DATA
PATTERNS
DEVELOPERS
SOCIAL
APPS
PLATFORM
WEB
MEMORY
DEVICES
PEOPLE

Questions?