

Distributed Deep Learning Frameworks, Optimizations & Implementations

前瞻深度學習

The Cutting Edge of Deep Learning

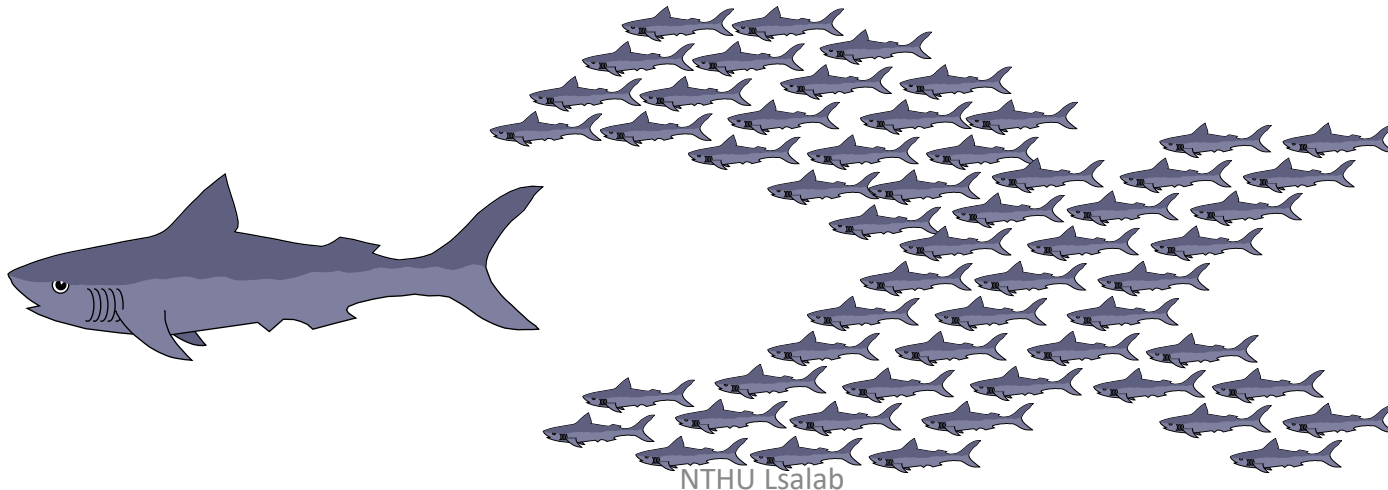
Instructor: Jerry Chou (周志遠)

Outline

- Distributed Computing
 - Scale-up vs Scale-out
- Computing framework & Parallelism
 - Parameter server
 - Data parallelism vs Model parallelism
- Optimization techniques
 - Mini Batch SGD
 - Asynchronous SGD
 - 1-Bit SGD
 - Other parameter swapping techniques...
- Distributed framework implementations
 - DistBelief
 - COTS HPC
 - Project Adam
 - Tensorflow

Why Distributed Computing

- Prediction **accuracy** is increased **by training larger models on vast amounts of data** using efficient and scalable compute clusters **rather than relying solely on algorithmic breakthroughs**
- The more significant advantage of a cluster-based approach to distributed optimization is its ability to handle **large models that cannot be comfortably fit on single machine.**



Scaling Out vs Scaling Up

- Scaling out: using multiple machines in a large cluster to increase the available computing power
- Scaling up: leveraging graphics processing units (GPUs), which can perform more arithmetic than typical CPUs

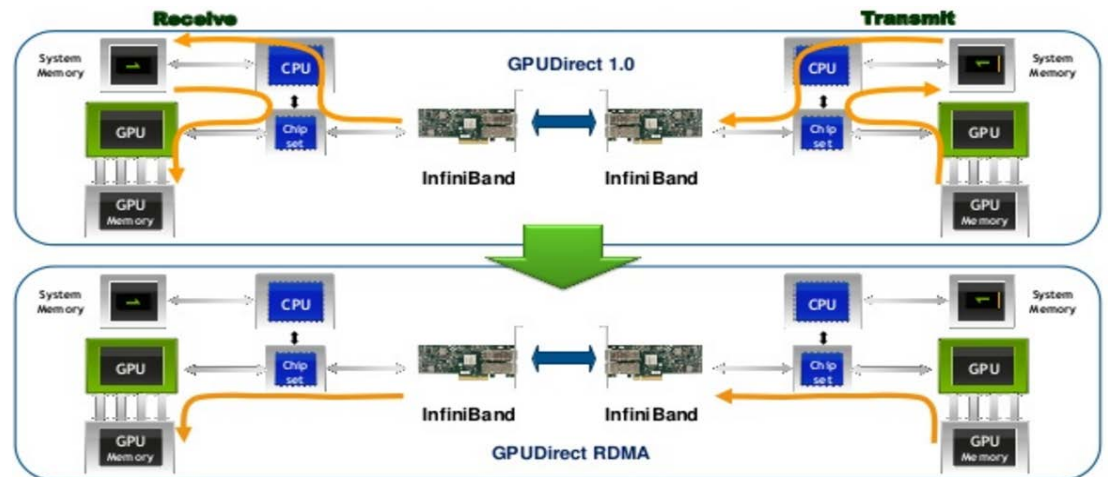


Challenges of GPU

- The mismatch in speed between GPU compute and network interconnects makes it extremely difficult to support data parallelism via a parameter server.
 - Either the GPU must constantly stall while waiting for model parameter updates or the models will likely diverge due to insufficient synchronization.

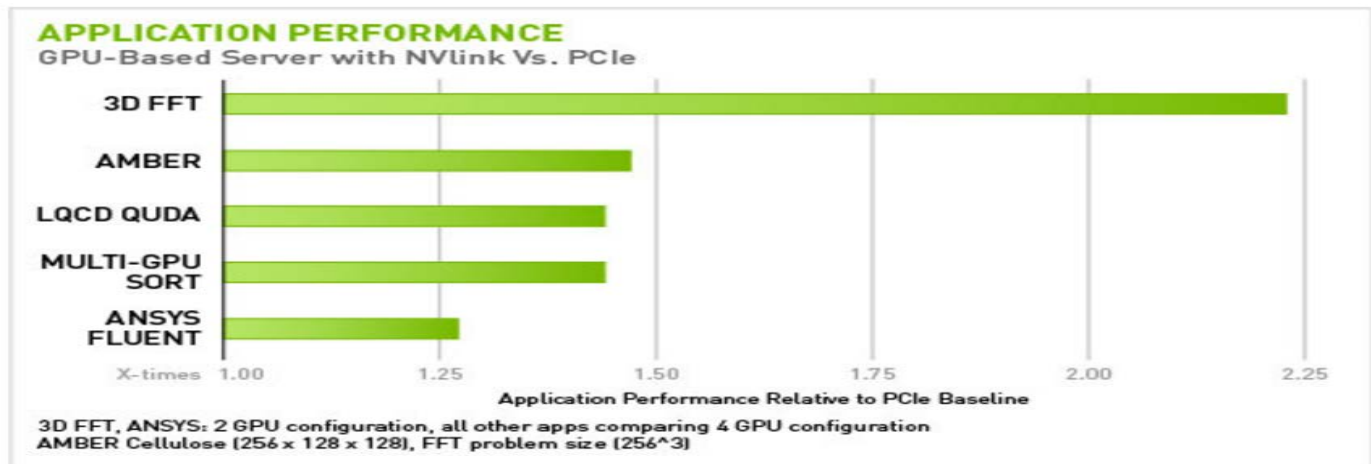
Opportunity of GPU

- GPUDirect
 - **A protocol** that allows third party device can directly read and write CUDA host and device memory
 - Eliminating unnecessary memory copies
 - Dramatically lowering CPU overhead, and reducing latency, resulting in significant performance improvements in data transfer times



Opportunity of GPU

- NV-Link
 - A high-bandwidth, energy-efficient **interconnect** that enables ultra-fast communication between the CPU and GPU, and between GPUs.
 - Data sharing at rates **5 to 12 times faster** than the traditional PCIe Gen3 interconnect



[2] <http://www.nvidia.com.tw/object/nvlink-tw.html>

Nvidia DGX-1

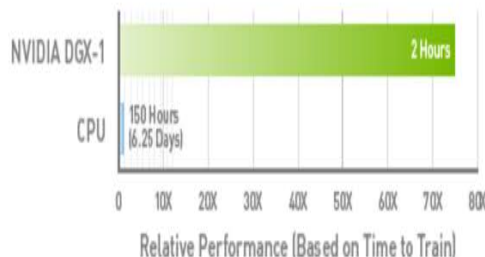


- World's first purpose-built system for deep learning and AI accelerated analytics
- Deliver performance equal to 250 conventional servers
- Built with groundbreaking **NVIDIA Pascal** GPU architecture, powered by **Tesla P100** accelerators and **NVLink** interconnect

SYSTEM SPECIFICATIONS

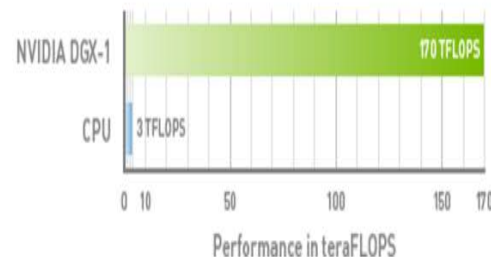
GPUs	8x Tesla GP100
TFLOPS (GPU FP16 / CPU FP32)	170/3
GPU Memory	16 GB per GPU
CPU	Dual 20-core Intel® Xeon® E5-2698 v4 2.2 GHz
NVIDIA CUDA® Cores	28672
System Memory	512 GB 2133 MHz DDR4 LRDIMM
Storage	4x 1.92 TB SSD RAID 0
Network	Dual 10 GbE, 4 IB EDR
Software	Ubuntu Server Linux OS DGX-1 Recommended GPU Driver
System Weight	134 lbs
System Dimensions	866 D x 444 W x 131 H (mm)
Packing Dimensions	1180 D x 730 W x 284 H (mm)
Maximum Power Requirements	3200W
Operating Temperature Range	10 - 35 °C

NVIDIA DGX-1 Delivers 75X Faster Training



Note: Caffe benchmark with AlexNet, training 1.28M images with 90 epochs | CPU server uses 2x Xeon E5-2697 v3 CPUs.

NVIDIA DGX-1 Delivers 56X More Performance



CPU is dual socket Intel Xeon E5-2697 v3. 170 TF is half precision or FP16.

Outline

- Distributed Computing
- Computing framework & Parallelism
 - Parameter server
 - Data parallelism vs Model parallelism
- Optimization techniques
- Distributed framework implementations

Distributed Computing Framework

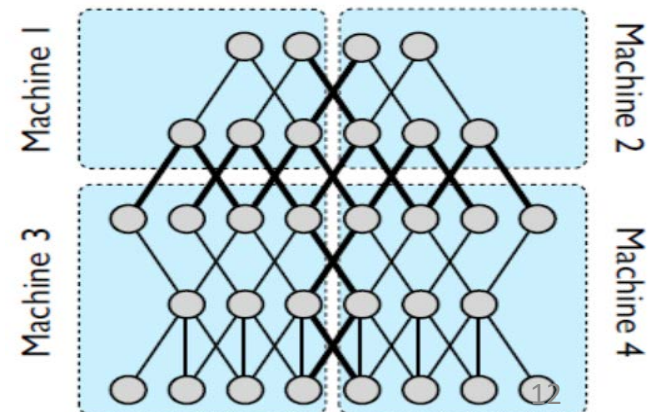
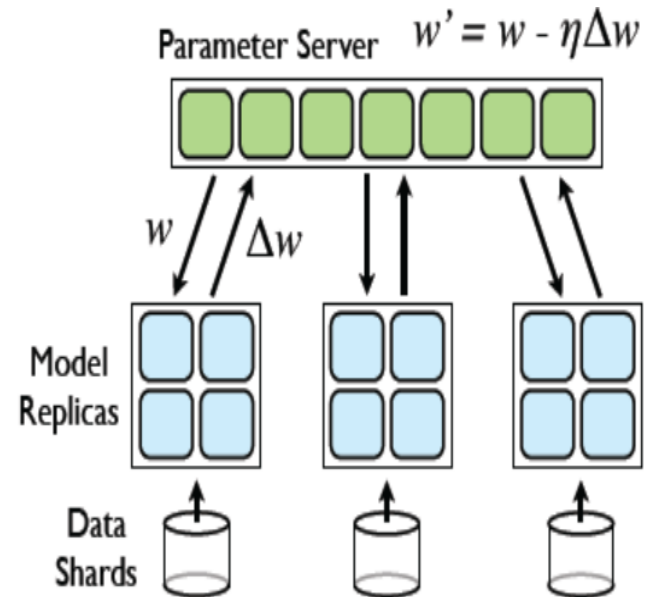
- What is a computing framework?
 - Hide programming and execution complexity from users
 - Define a simplified programming model
 - Provide runtime optimization to maximize system utilization and performance
 - Treat the cluster/datacenter as a single computer
 - Automatically parallelizes computation in each machine using all available cores, and manages **communication, synchronization and data transfer, etc.**

Popular framework for big data

- MapReduce(Hadoop)[7]
 - Follow a simple divide-and-conquer programming model
 - Designed for big data batch processing
 - Heavily rely on storage system without data caching
 - **Not suitable for complex dataflow computation**
- Spark[8]
 - Provide in-memory computation through a layer of distributed shared memory
 - A general function programming model based on SCALA
 - Designed for iterative or interactive computation
 - **Doesn't offer fine-grained scheduling for computation & communication**
- Graph processing framework: GraphLab[9], Pregel[10]
 - Vertex-centric programming model
 - Partition graph-based models with built-in scheduling and consistency mechanisms
 - **Limited theoretical work on whether asynchronous graph based consistency models and scheduling will always yield correct execution of such ML programs**

Basic Architecture for Distributed DL Framework

- Parameter server (Algorithm)
 - Exchange & update model parameters with **high frequency**
 - Parameters are distributed among the parameter server shards, and **updated independently**
- Parallelism (Computing Model)
 - Model replicas for **data parallelism**
 - Model partition for **model parallelism**
- Framework (Runtime Optimization)
 - Automatically parallelizes computation in each machine using all available cores, and manages **communication, synchronization and data transfer** between machines during both training and inference



Data Parallelism vs Model Parallelism

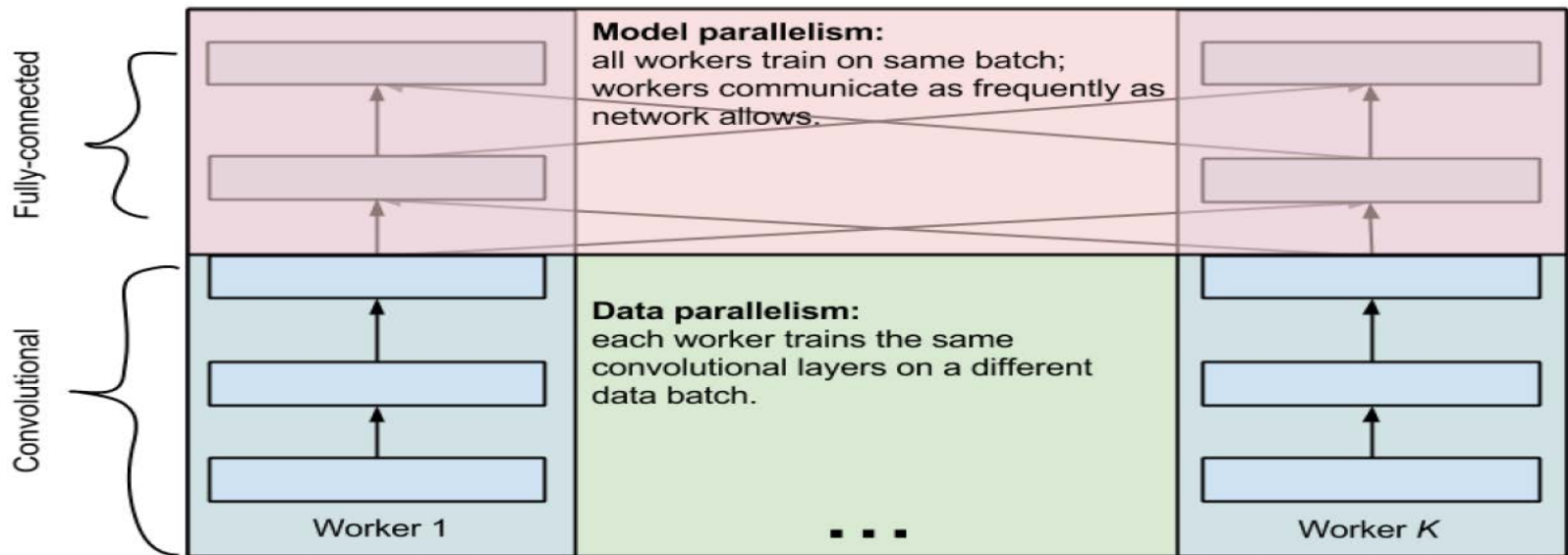
- Data parallelism
 - Create **model replica** and run it on different subset of data.
 - The algorithm **distributes the data** between various cores.
 - Each core independently tries to estimate **the same parameter(s)**
 - Can be easily implemented by general data processing frameworks like Hadoop, Spark.
 - Biggest problem is that during the backward pass you have to **pass the whole gradient to the all others**
 - **The size of network model is bounded by a single node**
 - **More suitable for smaller clusters or networks with fewer parameters**

Data Parallelism vs Model Parallelism

- Model parallelism
 - Each node only manages a **partition of the model**
 - The algorithm sends the **same data** to all the cores.
 - Each core is responsible for estimating **different parameter(s)** which relevant to its own partition
 - Can solve large network model problem
 - **But require frequent synchronization**
 - No tricks to hide the communication needed for synchronization, because we have only partial information for the whole batch, and we **have to wait for the completion of the synchronization to move forward.**

Mixing of the two Parallelism Model?

- Convolutional layers cumulatively contain about 90-95% of the computation, about **5% of the parameters**, and have large representations.
- Fully-connected layers contain about 5-10% of the computation, about **95% of the parameters**, and have small representations.



Mixing of the two Parallelism Model?

- There are several ways to switch from data parallelism to model parallelism
 1. **Broadcast:** Each worker sends its last-stage convolutional layer activities to each other worker. The workers then assemble a big batch of activities to compute the fully-connected activities.
 2. **Vertical Pipeline:** One of the workers sends its last-stage convolutional layer activities to all other workers. The workers then compute the fully connected activities. In parallel with this computation, the next worker sends its last-stage convolutional layer activities.
 3. **Horizontal Pipeline** All of the workers send a share of their last stage convolutional layer activities to all other workers. Overlap this computation time with the next round of communication time.

Outline

- Distributed Computing
- Computing framework & Parallelism
- Optimization techniques
 - Mini Batch SGS
 - Asynchronous SGD
 - Stale Synchronous Parallel PS
 - 1-Bit SGD
 - Sufficient Factor Broadcasting (SFB)
 - Partitioned Linearity Topology Parameter Swapping
- Distributed framework implementations

Mini Batch SGD

- Algorithms:
 - Batch Gradient Descent: use all ***m*** examples in each iteration
 - Stochastic Gradient Descent: use ***1*** examples in each iteration
 - Mini-batch Gradient Descent: use ***b*** examples in each iteration

Say $b = 10, m = 1000$.

Repeat {

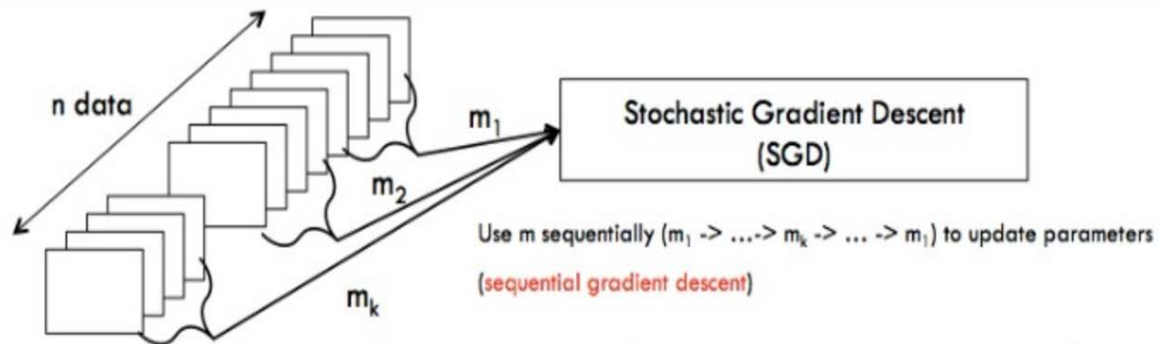
for $i = 1, 11, 21, 31, \dots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \dots, n$)

}

}



[6] <https://www.coursera.org/learn/machine-learning/lecture/9zJUs/mini-batch-gradient-descent>

Mini Batch SGD

- Advantages:
 - **Vectorization**: We can make data parallelism arbitrarily efficient by increasing the batch size (In particular for GPU)
 - **Lower communication cost**: fewer number of iterations comparing to SGD
 - **Smoother update**: the variance of the update is reduced
- Risks
 - Very big batch sizes **adversely affect** the SGD **converges rate** as well as the **quality** of the final solution
 - Noise actually can be useful as it may help escape local minima
- A variant of mini-batch SGD is proposed whose convergence rate does not degrade when the batch size increases.

[12] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. “Efficient mini-batch training for stochastic optimization”. In *Proceedings of the 20th ACM SIGKDD, 2014*

Downpour (Asynchronous) SGD

[Google DistBelief]

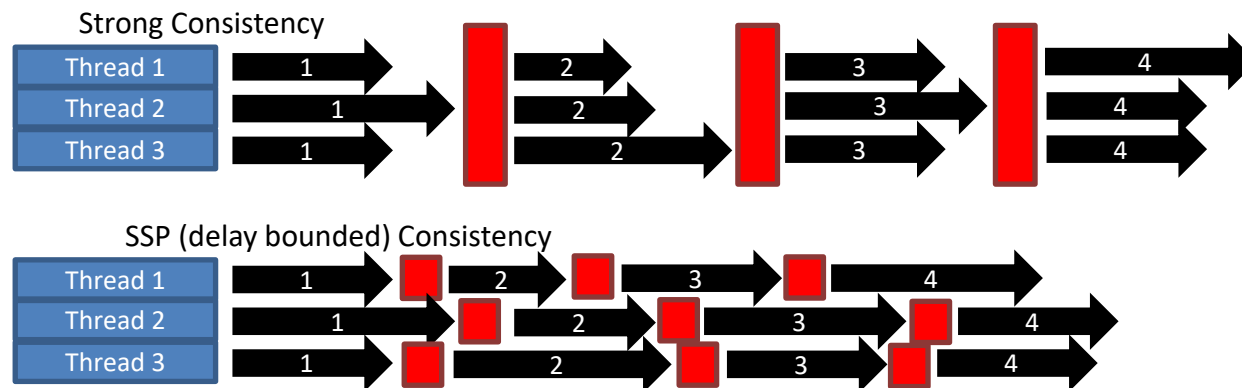
- Parameter updates can be handle **asynchronously**.
 - limiting each model replica to request updated parameters only every n steps
 - send updated gradient values only every m steps
 - Parameter server shards are updated independently with **inconsistent timestamp**
 - Updates may be **out of order**
- In practice, relaxing consistency requirements is remarkably effective, and could achieve even better accuracy

[13] Google(DistBlief), “Large Scale Distributed Deep Networks”, In Neural Information Processing Systems, 2012

Stale Synchronous Parallel PS

[CMU Poseidon]

- Update
 - on workers: update sum of $\Delta_\ell(A^{(t-1)}, D_p)$ over data subsets
 - on PS: update model parameter A with a **consistency scheme**
- SSP consistency model
 - Error-tolerance property of training neural networks
 - Staleness threshold s defines the acceptance range for delays
 - changes no later than s iterations ago are guaranteed to be seen
 - readers may wait for stragglers if it is more than s iterations behind



1-Bit SGD

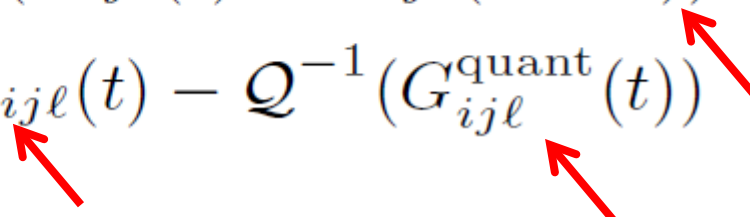
[Microsoft CNTK]

- Idea: quantize the gradients aggressively—to but one bit per value—if the quantization error is carried forward across mini batches (error feedback)
 - This is a common technique in other areas, such as sigma-delta modulation for DACs(Delta-sigma modulation technique for digital-to-analog converter), or image rasterization.

$$\begin{aligned} G_{ijl}^{\text{quant}}(t) &= \mathcal{Q}(G_{ijl}(t) + \Delta_{ijl}(t - N)) \\ \Delta_{ijl}(t) &= G_{ijl}(t) - \mathcal{Q}^{-1}(G_{ijl}^{\text{quant}}(t)) \end{aligned}$$

gradient parameter quantized values

error



- As long as error feedback is used, we can quantize all the way to 1 bit at no or nearly no loss of accuracy.

1-Bit SGD

[Microsoft CNTK]

- $$\begin{bmatrix} 0.16 & \cdots & w_{1,N} \\ \vdots & \ddots & \vdots \\ 0.7 & \cdots & w_{M,N} \end{bmatrix} \rightarrow \begin{bmatrix} 0 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 0 \end{bmatrix}$$

$0.16 < 0.23$
 $0.7 > 0.23$

Column avg. \rightarrow 0.23

Avg. is sent over network

Effect: communication reduced by 32x

- Results:**
 - A 160M-parameter model training processes 3300h of data in under 16h on 20 dual-GPU servers—a 10 times speed-up—albeit at a small accuracy loss

Sufficient Factor Broadcasting (SFB)

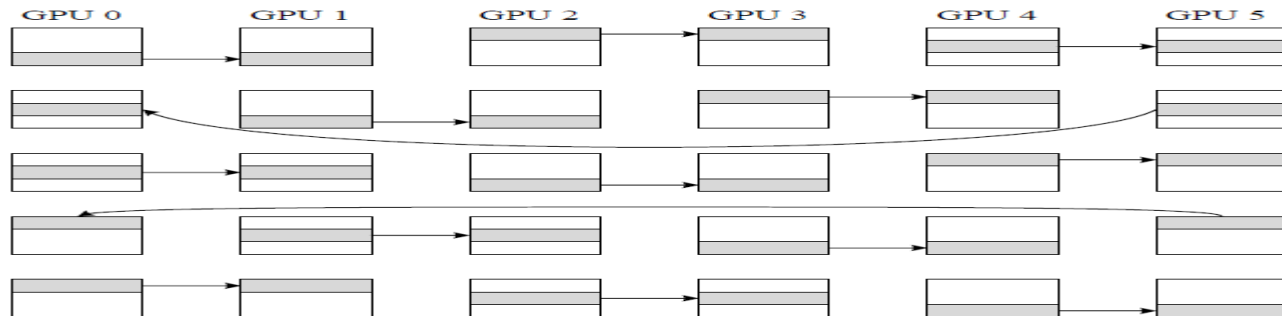
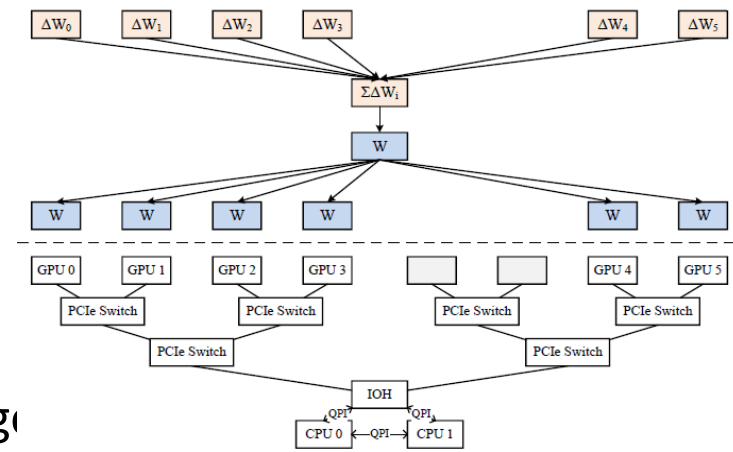
[CMU Poseidon, Google Project Adam]

- Goal: reduce network communication costs for matrix parametrized models; specifically, those that follow an optimization formulation
$$\min_{\mathbf{W}} \quad \frac{1}{N} \sum_{i=1}^N f_i(\mathbf{W}\mathbf{a}_i) + h(\mathbf{W})$$
- “Delta” Update method: $W \leftarrow w - \gamma \Delta W$
 - De-couple ΔW into two vectors u and v : $\Delta W = uv^T$
 - $u = \frac{\partial f(W_{a_i}, b_i)}{\partial f(W_{a_i})}, v = a_i$
 - Broadcast u, v to all other peer workers, instead of ΔW
 - Worker reconstruct matrix updates ΔW locally
- Results:
 - Allow p2p communication instead of using centralized parameter server
 - Tradeoff between the data transfer size and number of messages

Partitioned Linearity Topology

Parameter Swapping

- For a multi-GPU server, GPUs are connected by PCIe switches in a tree topology
 - parameter swapping performance is limited by PCIe bandwidth (4~6GB/s), which is much slower than 200+ GB/s GPU memory bandwidth,
- Partitioned linearity topology:
 - each partition is transferred to and merged while all the $N/2$ data transfer flows don't interfere with each other.



Outline

- Distributed Computing
- Computing framework & Parallelism
- Optimization techniques
- Distributed framework implementations
 - DistBelief
 - COTS HPC
 - Project Adam
 - Tensorflow

Distributed Framework Implementations

Framework	Organization	Model Parallelism	Data Parallelism	GPU	Source
SparkNet	UCB	No	Yes	Yes	https://github.com/amplab/SparkNet
Caffe-MPI	China Inspur	No	Yes	Yes	https://github.com/Caffe-MPI/Caffe-MPI.github.io
MPI-Caffe	VT, U. Indiana	Yes	No	Yes	https://computing.ece.vt.edu/~steflee/m-pi-caffe.html
Poseidon (Petuum)	CMU	No	Yes	Yes	https://github.com/petuum/poseidon
COTS HPC	Google	Yes	No	Yes	N.A.
DistBelief	Google	Yes	Yes	No	N.A.
CNTK	Microsoft	Yes	Yes	Yes	https://github.com/Microsoft/CNTK/wiki
Project Adam	Microsoft	Yes	Yes	No	N.A.
Theano	U. Montreal	Yes	Exp (Platoon)	Yes	http://deeplearning.net/software/theano/introduction.html
TensorFlow	Google	Yes	Yes	Yes	https://www.tensorflow.org/
MXNET	CMU, UW, etc.	Yes	Yes	Yes	https://github.com/dmlc/mxnet

DistBelief

- An earlier work that implements two training algorithms
 - Downpour SGD: an asynchronous stochastic gradient descent procedure supporting a large number of model replicas
 - Sandblaster: a framework that supports a variety of distributed batch optimization procedures, including a distributed implementation of L-BFGS

[13] “Large Scale Distributed Deep Networks”, *In* Neural Information Processing Systems, 2012

[18] “Building high-level features using large scale unsupervised learning”, ICML, 2012.

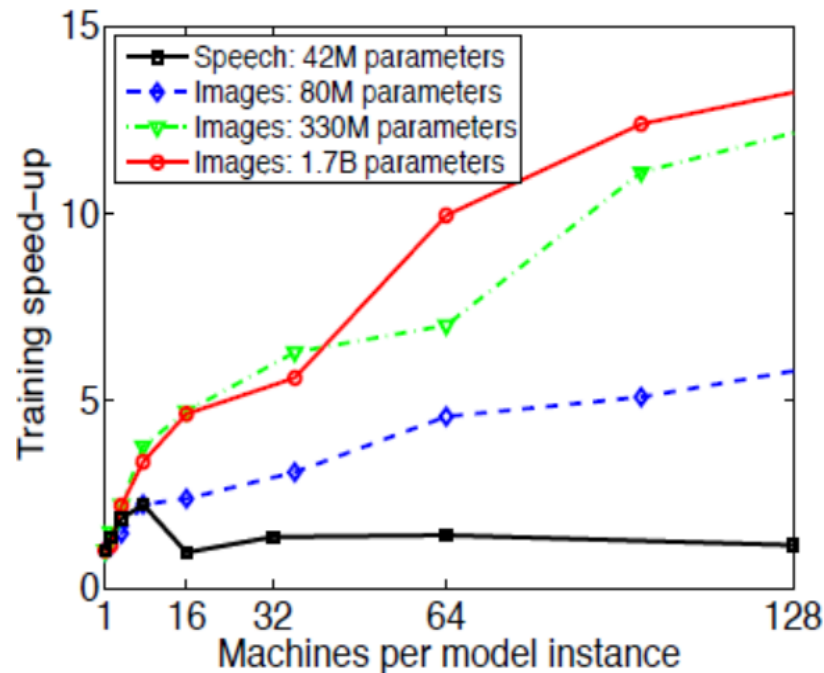
Experimental Setup

- Speech
 - Fully connected network
 - Four hidden layer with sigmoidal activations, and a softmax output layer
 - A total of approximately 42 million model parameters
- Visual
 - ImageNet[19]: convolution network
 - Larger network (more connections) than speech

[19] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *Computer Vision and Pattern Recognition*. CVPR '09.

Scalability

- The network overhead starts to **dominate for speech model after 8 nodes**.
- But the larger **locally-connected image models** can still benefit from more nodes.



Accuracy

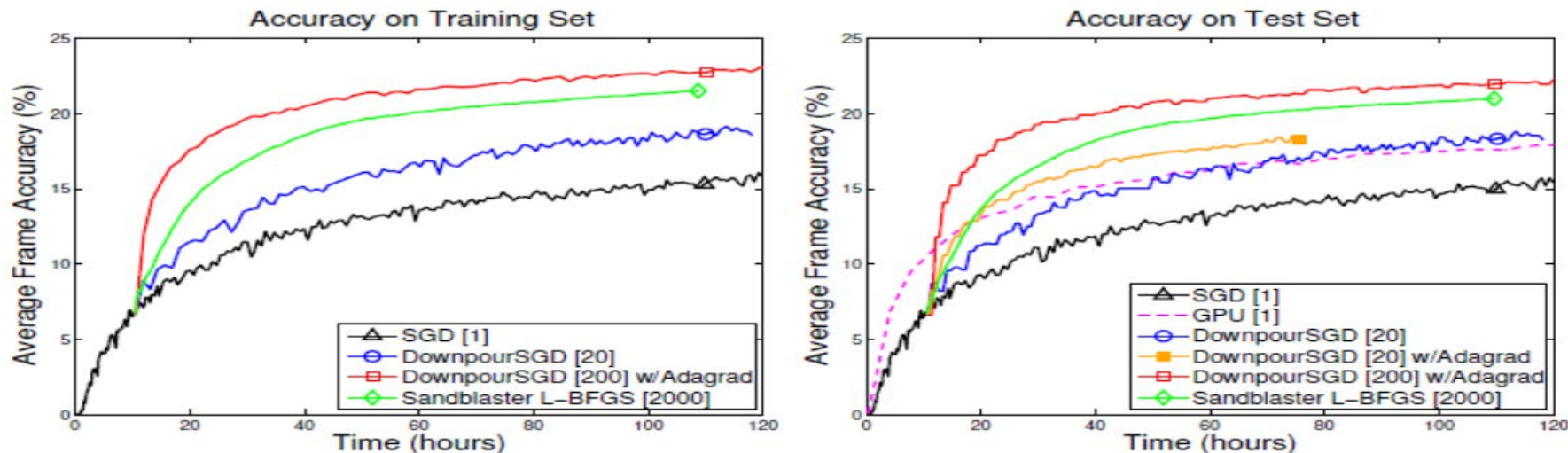


Table 2. Summary of classification accuracies for our method and other state-of-the-art baselines on ImageNet.

Dataset version	2009 (~9M images, ~10K categories)	2011 (~14M images, ~22K categories)
State-of-the-art	16.7% (Sanchez & Perronnin, 2011)	9.3% (Weston et al., 2011)
Our method	16.1% (without unsupervised pretraining) 19.2% (with unsupervised pretraining)	13.6% (without unsupervised pretraining) 15.8% (with unsupervised pretraining)

- Adagrad is a adaptive learning rate algorithm
- At that time, this network achieved a cross-validated classification accuracy of over 15%, a relative improvement over 60% from the best performance we are aware of on the 21k category ImageNet classification task.

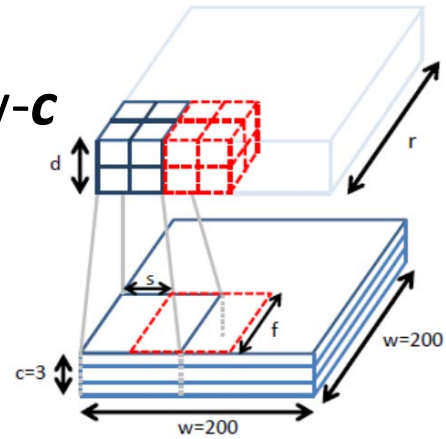
COTS HPC: Commodity Off-The-Shelf High Performance Computing

- Motivation: Limited resource on a single server
 - I/O, power, cooling, and CPU compute demands.
- Solution: An **model parallelism** implementation of a cluster of **GPU** servers with **Infiniband** interconnects and **MPI**
 - GPU: parallel computation power (4GB RAM, 4 x GTX680, 1TFLOPs)
 - Infiniband: high throughput interconnect (FDR, 56 Gbps)
 - MPI: message passing communication programming library (MVAPICH2)
- Challenges:
 - Require highly **optimized GPU code for all major computational operations**
 - Must develop a scheme for distributing the computations over many GPUs and **managing the communication** between them
- Result: **Train neural networks at scales (1Bn) comparable to DistBelief with just 3 machines**

[20] Adam Coates, et al., “Deep learning with COTS HPC systems”, ICML, 2013

Techniques

- **Min batch SGD**: 4D array of size M -by- w -by- w -by- c
 - M is the mini-batch size
 - w is the image width and height
 - c is the number of input channels.
- Efficient operation on $Y=WX$
 - Their own implementation only achieved 300GFLOPS (peak at 1TFLOPS)
 - Referenced the highly optimized **MAGMA BLAS** matrix-matrix multiply kernels, which makes use of advanced techniques including pre-fetching, exploitation of ILP, and careful register usage.
- Strictly **model parallel** scheme for parallelizing across GPUs
 - Use a distributed array abstraction hides virtually all communication from the rest of the computational code



Scalability

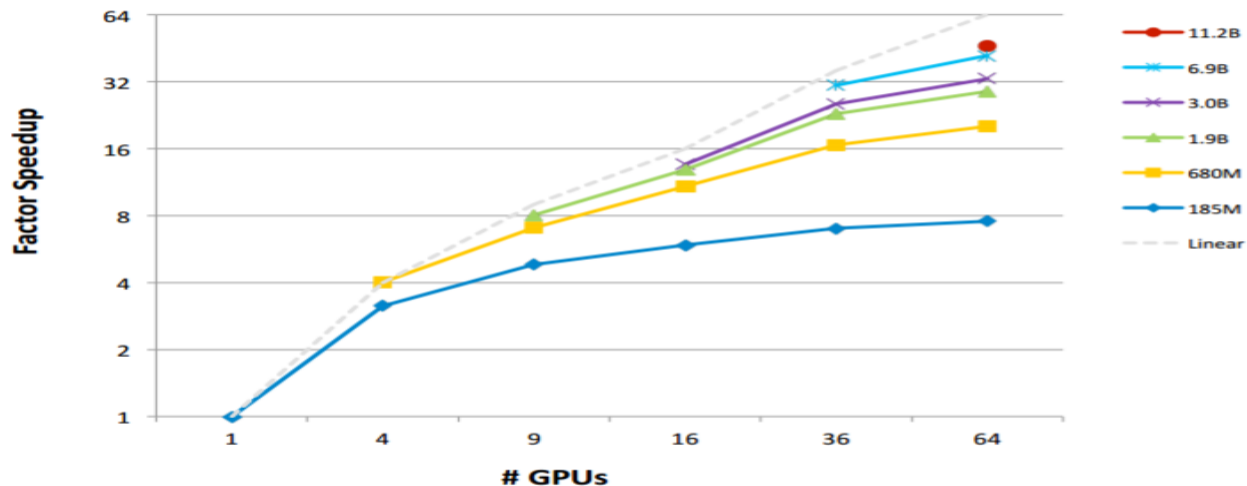


Figure 6. Factor speedup obtained for varying sizes of network and number of GPUs, normalized for the size of the network.

- Using many GPUs does not yield significant increases in computational throughput for small networks, but the system excels when working with much larger networks.

Project Adam

- *Multi-Threaded Training (Data Parallelism)*
 - Each thread allocates a training context for feed-forward evaluation and back propagation.
- *Fast Weight Updates (Synchronization)*
 - shared model weights are accessed and updated locally without using locks.
 - because neural networks are resilient and can overcome the small amount of noise
- *Reducing Memory Copies (Communication)*
 - Local communication uses memory reference
 - Remote communication uses our their own optimized network library implementation

Project Adam

- *Memory System Optimizations (Model Parallelism)*
 - Partition models across multiple machines
 - Fit the working sets for the model layers into the L3 cache
 - Optimize cache locality by creating two custom hand-tuned assembly kernels that appropriately pack and block the data for forward evaluation and back-propagation computation
- *Mitigating the Impact of Slow Machines (Load Balancing)*
 - Only require 75% of the model replicas to complete from each epoch. (the image processing order is randomized to ensure the same set of images are not skipped)
 - *Adapt work stealing technique(not implemented)*

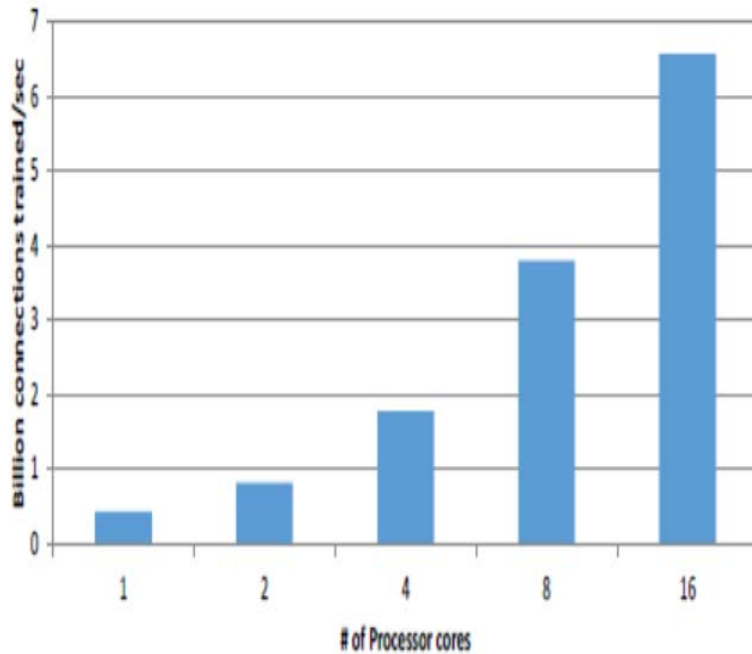
Project Adam

- *Parameter Server Communication*
 - Directly send accumulated weights
 - works well for the convolutional layers since the volume of weights is low due to weight sharing
 - Sufficient Factor Broadcasting
 - Only send the activation and error gradient vectors to the parameter server machines where the matrix multiply can be performed locally to compute the weight updates.

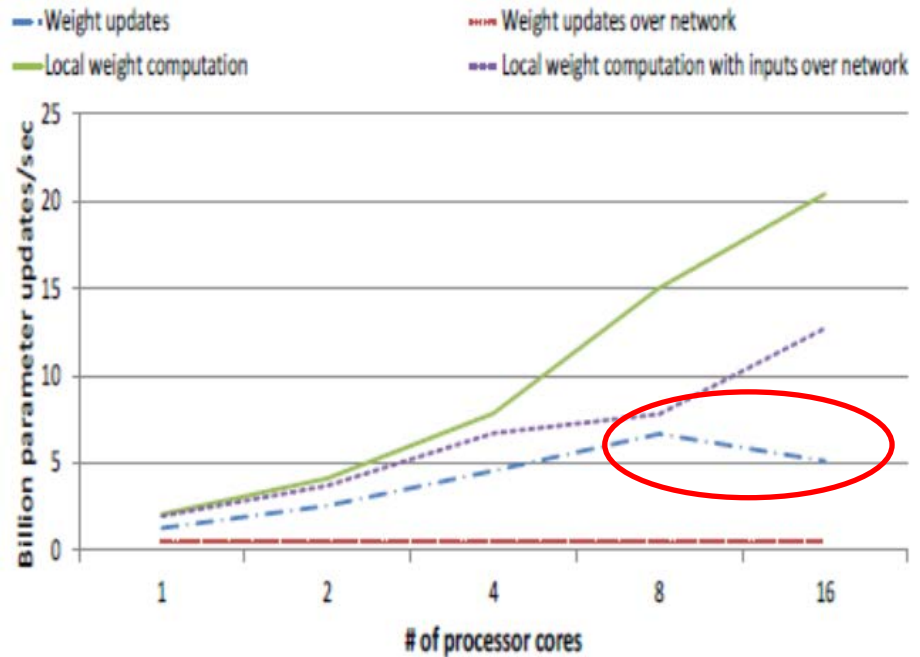
Dataset

- Dataset
 - MNIST: 28X28 images of the 10 handwritten digits, 60K training image, 10K test image
 - ImageNet: 15M high-resolution images from 22K categories
 - Random guessing will result in an accuracy of only around 0.0045%
 - it is unlikely that human performance exceeds 20%
- Machine: 120 nodes:
 - 90 training nodes, 20 parameter servers, 10 backup
 - dual Intel Xeon E5-2450L processors
 - 16 cores, 1.8GHz, 98GB RAM, 2x10Gb NIC, 1x1Gb NIC

Single Node Scalability Results

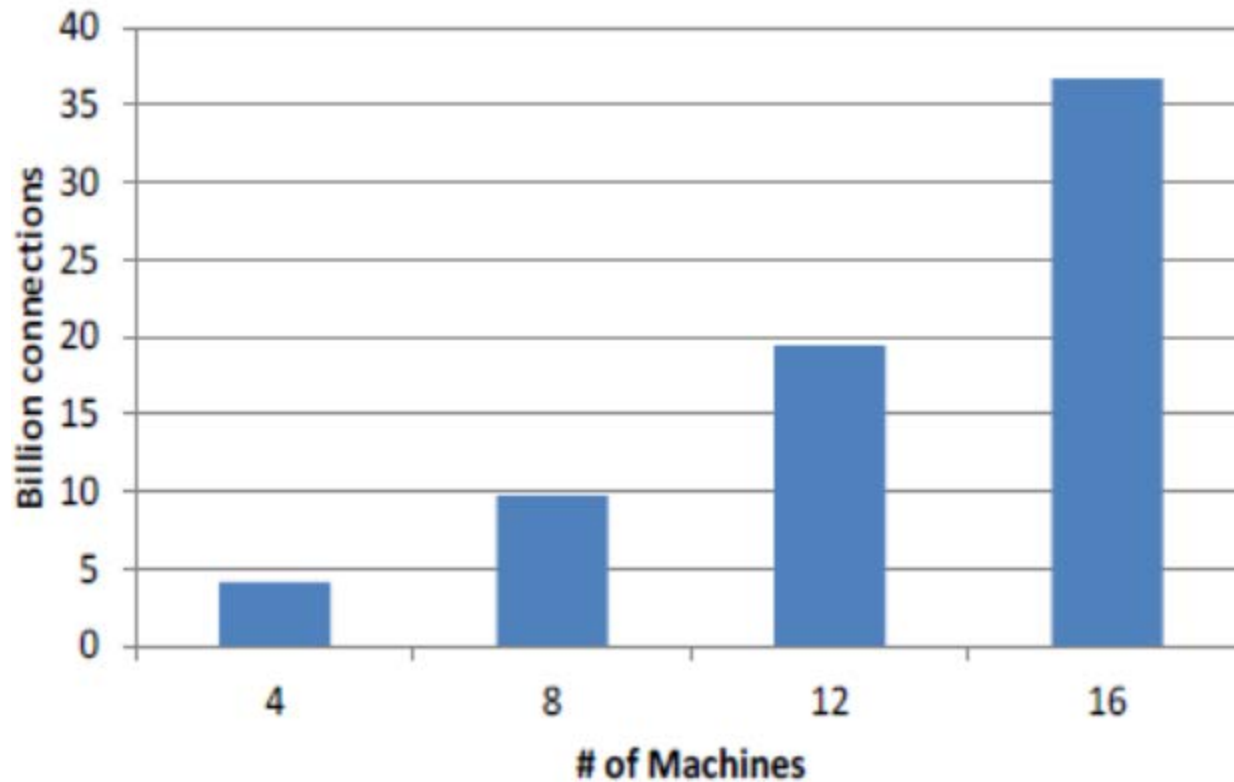


- MNIST
- Single node, no parameter server
- Excellent scaling
- super-linear up to 4 cores due to caching effects



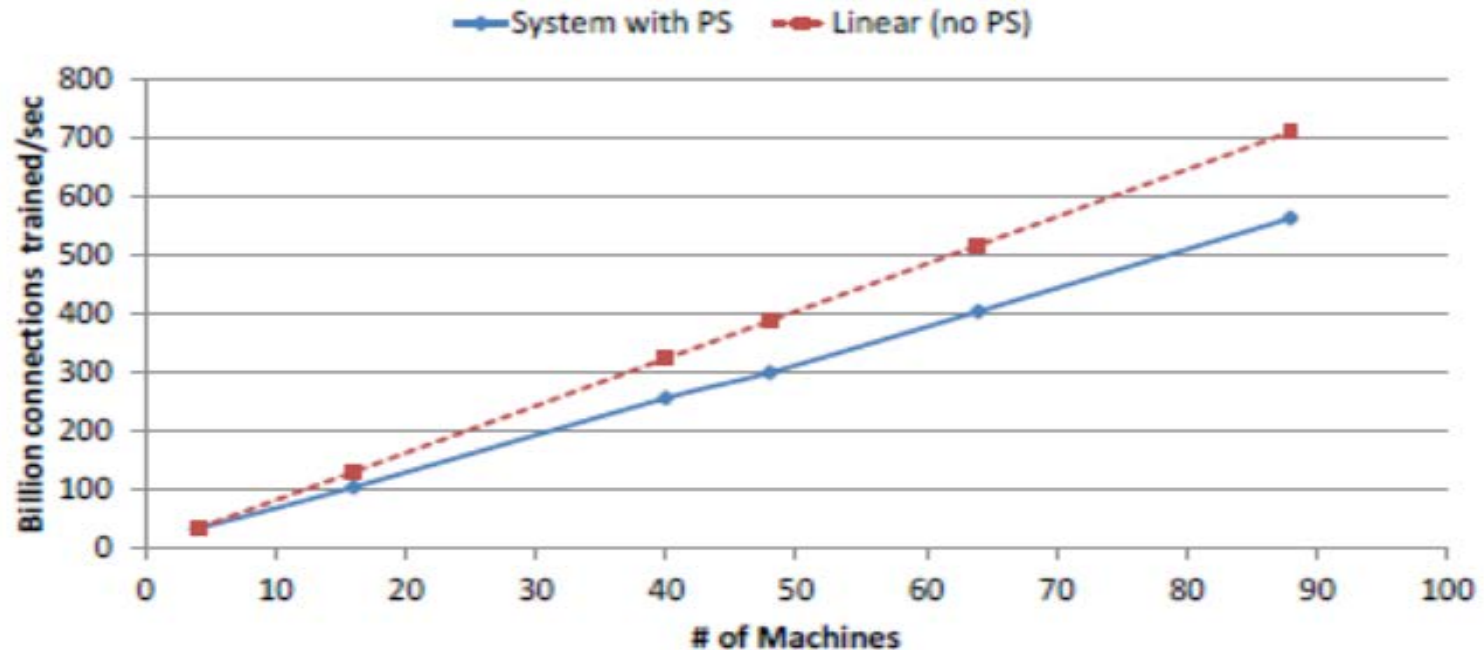
- ImageNet
- Single node, with parameter server
- Network resulting in poor performance and scaling

Model Parallelism



- The size of trained model increases proportionally to the number of machine

Data Parallelism



- ImageNet
- 20 parameter servers
- Each replica consisted of 4 machines (model parallelism)
- #replica increases from 4, 10, 12, 16 to 22 (data parallelism).

Summary

- Adam achieves **high multi-threaded scalability** on a single machine by permitting threads to update local parameter weights **without locks**
- It achieves good **multi-machine scalability** through minimizing communication traffic by performing the weight **update computation on the parameter server machines** and performing **asynchronous batched updates** to parameter values
- Enables training models to **high accuracy** by **exploiting its efficiency to train very large models** and leveraging asynchrony to further improve accuracy

TensorFlow

- **Google's Second-generation system** (succeed Distbelief) for the implementation and deployment of largescale machine learning models.
- Takes computations described using a **dataflow-like model** and **maps them onto a wide variety of different hardware platforms**
 - ranging from running inference on mobile device platforms to training on GPU clusters
- Simplify the real-world use of ML system.

[22] "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", Preliminary White Paper, November 9, 2015

Programming Model

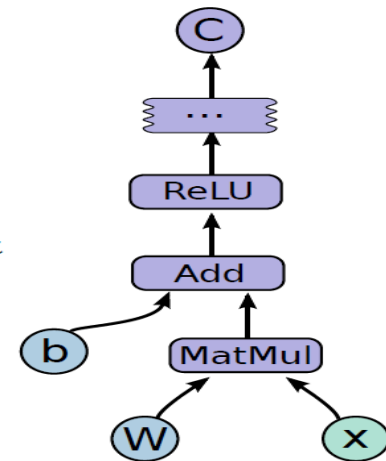
- Computation is described by a directed graph
- A tensor is a typed multi-dimensional array (**ephemeral** by default)
- Variables is a special operation that returns a handle to a **persistent mutable tensor** that survives across executions

```
import tensorflow as tf
```

```
b = tf.Variable(tf.zeros([100])) # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1)) # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x") # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b) # Relu(Wx+b)
C = [...] # Cost computed as a function of Relu
```

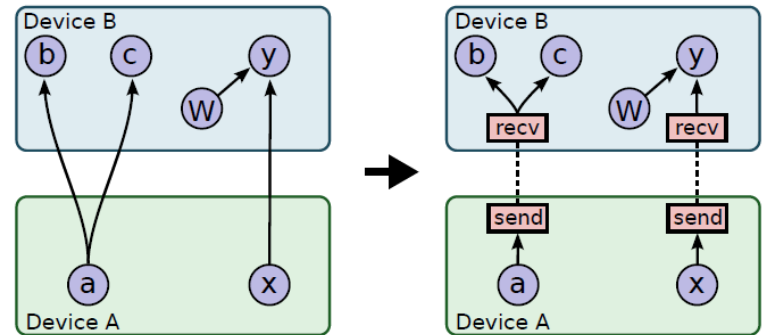
```
s = tf.Session()
for step in xrange(0, 10):
    input = ...construct 100-D input array ... # Create 100-d vector for input
    result = s.run(C, feed_dict={x: input}) # Fetch cost, feeding x=input
    print step, result
```

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration



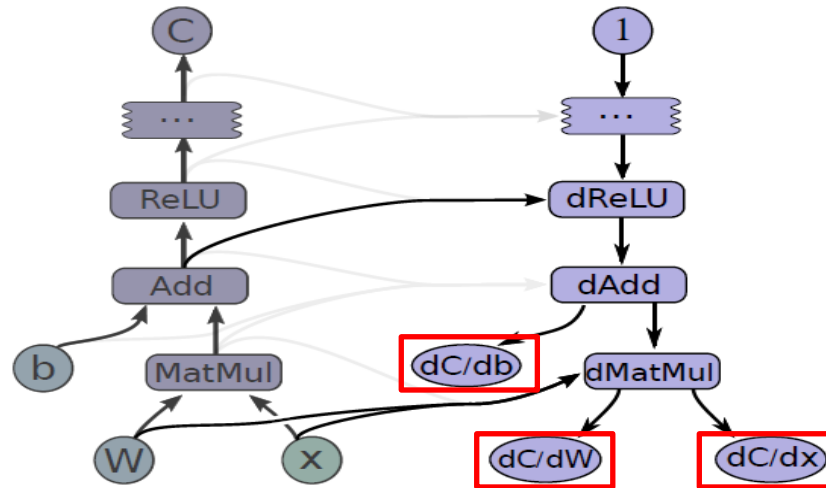
Implementation

- Node placement algorithm: map graph to devices
 - Each node has an estimated cost of input/output data size
 - Run a simulated execution of the graph using greedy heuristics
 - An area of ongoing development within the system
- Cross-device communication
 - Adding the Send and Receive nodes coordinate to transfer data across devices
 - Isolate all communication inside Send and Receive implementations



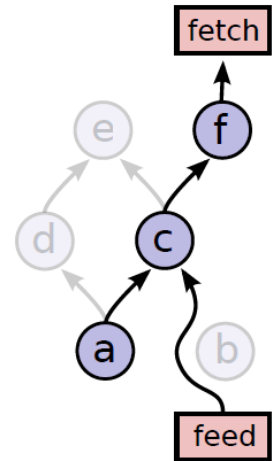
Extension

- Gradient computation
 - Built-in support for automatic gradient computation
 - Gradient nodes are automatically added to the graph
 - But it **complicates optimization**, particularly of memory usage. (**Optimization is a work in progress**)
 - E.g.: `[db, dW, dx] = tf.gradients(C, [b, W, x])`



Extension

- Partial execution
 - Allow users to **execute a subgraph of the entire execution graph**
 - The “**Run**” method allows to execute an arbitrary subgraph of the whole graph, and to **inject arbitrary data along any edge in the graph**, and to **retrieve data flowing along any edge in the graph**
 - The special nodes of “**fetch**” and “**feed**” are added to feed the injected input and fetch the partial results
- Device constraint
 - Users can **control the placement of nodes on devices by providing partial constraints** for a node about which devices it can execute on, such as **limiting the total amount of memory needed on a device**
 - Devices will be filtered from the placement algorithm’s simulator



Extension

- Control flow
 - Introduce a small set of **primitive control flow operators** into TensorFlow and generalize TensorFlow to **handle cyclic dataflow graphs**, such as “**Merge**”, “**Switch**”
 - Much as in the dataflow-machine approach described by Arvind
- Queue
 - Allow different portions of the graph to **execute asynchronously**
 - “**Enqueue**” operations can block until space becomes available in the queue, and “**Dequeue**” operations can block until a desired minimum number of elements are available in the queue
- Container
 - It is used the mechanism within TensorFlow for managing longer-lived mutable state. The **backing store for a Variable** lives in a container.

Optimization

- Common Subexpression Elimination
 - There are many redundant copies of the same computation across layers
 - Merge them into a single node and redirect graph edge
- Controlling data communication and memory usage
 - Analyze the critical paths of graphs, in order to estimate when to start the Receive nodes. We then insert control edges with the aim of delaying the start of these nodes until just before their results are needed.
- Asynchronous(non-blocking) kernels:
 - receive, enqueue, dequeue
- Optimized libraries:
 - BLAS, cuDNN
- Lossy compression:
 - 32→16bits

Similarity to existing works

- Halide^[23]
 - Similar intermediate representation for expressing image processing pipelines
- Dryad^[24], Flume^[25], Spark^[8]
 - Distributed dataflow execution frameworks
- CIEL^[26], Naiad^[27]
 - Generic support for data-dependent control flow
- Dandelion^[28]
 - Executes dataflow graphs across a cluster of heterogeneous devices, including GPUs

Evolution of DL Frameworks

- DistBelief
 - Train a 1.7B deep network 30x larger than previously reported
 - Accuracy of over 15%, a relative improvement over 60% from the best
- Deep learning with COTS HPC systems
 - Train neural networks at scales comparable to DistBelief with just 3 machines
 - 1B networks, 3 nodes(4 NVIDIA GTX680, 1TF), IB (56Gbps)
 - 11B networks, 16 nodes(4 NVIDIA GTX680 , 1TF), IB (56Gbps)
- Project Adam
 - 2B networks, ImageNet, 62 nodes, 10days
 - 36B networks, ImageNet, 16 nodes
 - Achieve a new world record prediction accuracy of 29.8% using only ImageNet training data. (a dramatic 2x improvement over the prior best.)
- Tensorflow
 - A dataflow computing framework for DL
 - Focus on the flexibility of programming model, and the ease of deployment on wide variety of execution environment

Reference

- [1] <https://developer.nvidia.com/gpudirect>
- [2] <http://www.nvidia.com.tw/object/nvlink-tw.html>
- [3] <http://www.nvidia.com/object/deep-learning-system.html>
- [4] <http://timdettmers.com/2014/10/09/deep-learning-data-parallelism/>
- [5] <http://timdettmers.com/2014/11/09/model-parallelism-deep-learning/>
- [6] <https://www.coursera.org/learn/machine-learning/lecture/9zJUs/mini-batch-gradient-descent>
- [7] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters", Commun. ACM 51, 1 (January 2008), 107-113.
- [8] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In USENIX NSDI, 2012.
- [9] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J., "Distributed GraphLab: A framework for machine learning in the cloud". In VLDB, 2012.
- [10] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In SIGMOD '10.

Reference

- [11] Krizhevsky, Alex. "One weird trick for parallelizing convolutional neural networks." CoRR abs/1404.5997, 2014 (arXiv:1404.5997).
- [12] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. 2014. Efficient mini-batch training for stochastic optimization. In ACM SIGKDD, pages 661-670, 2014.
- [13] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q., and Ng, A. 2012. Large Scale Distributed Deep Networks. In Advances in Neural Information Processing Systems. NIPS'12.
- [14] Hao Zhang, Zhiting Hu, Jinliang Wei, Pengtao Xie, Gunhee Kim, Qirong Ho, Eric Xing, "Poseidon: A System Architecture for Efficient GPU-based Deep Learning on Multiple Machines", In USENIX Annual Technical Conference (ATC 2016)
- [15] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, Dong Yu, "1-Bit Stochastic Gradient Descent and its Application to Data-Parallel Distributed Training of Speech DNNs", In Interspeech 2014
- [16] P Xie, JK Kim, Y Zhou, Q Ho, A Kumar, Y Yu, E Xing, "Lighter-Communication Distributed Machine Learning via Sufficient Factor Broadcasting", In Conference on Uncertainty in Artificial Intelligence, 2016

Reference

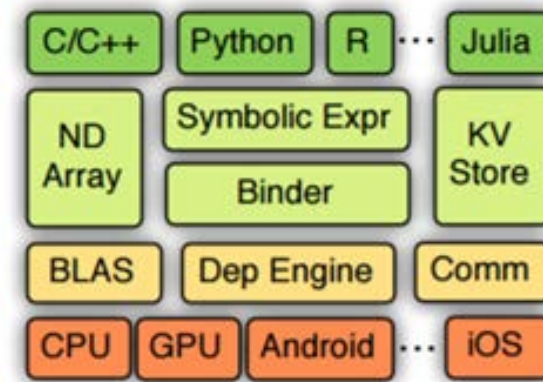
- [17] Yongqiang Zou, Xing Jin, Yi Li, Zhimao Guo, Eryu Wang, and Bin Xiao. 2014. Mariana: tencent deep learning platform and its applications. Proc. VLDB Endow. 7, 13 (August 2014), 1772-1777.
- [18] Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., and Ng, A, "Building high-level features using large scale unsupervised learning". ICML, 2012.
- [19] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR '09.
- [20] A. Coates, B. Huval, T. Wang, D.-J. Wu, and A.-Y. Ng, "Deep Learning with COTS HPC Systems," ICML, 2013.
- [21] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman, "Project Adam: building an efficient and scalable deep learning training system", In Proceedings of OSDI, pages 571-582, 2014.
- [22] Mart´ın Abadi, et al., "TensorFlow: Large-scale machine learning on heterogeneous systems", Preliminary White Paper, November 9, 2015.
- [13] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Fredo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. ACM SIGPLAN Notices, 48(6):519–530, 2013.

Reference

- [24] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In ACM SIGOPS Operating Systems Review, volume 41, pages 59–72. ACM, 2007.
- [25] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R Henry, Robert Bradshaw, and Nathan Weizenbaum. FlumeJava: easy, efficient data-parallel pipelines. In ACM Sigplan Notices, volume 45, pages 363–375. ACM, 2010.
- [26] Derek G. Murray, Malte Schwarzkopf, Christopher Snowton, Steven Smit, Anil Madhavapeddy, and Steven Hand. Ciel: a universal execution engine for distributed data-flow computing. In USENIX NSDI, 2011.
- [27] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Mart´ın Abadi. Naiad: a timely dataflow system. In ACM Symposium on Operating Systems Principles, pages 439–455. ACM, 2013.
- [28] Christopher J Rossbach, Yuan Yu, Jon Currey, JeanPhilippe Martin, and Dennis Fetterly. Dandelion: a compiler and runtime for heterogeneous systems. In ACM Symposium on Operating Systems Principles, pages 49–68. ACM, 2013.

Backup

MXNet



- MXNet(or “mix-net”) is a multi-language ML
- Blends declarative symbolic expression with imperative tensor
- A a superset programming interface to other systems

	Imperative Program	Declarative Program
Execute $a = b + 1$	Eagerly compute and store the results on a as the same type with b .	Return a computation graph; bind data to b and do the computation later.
Advantages	Conceptually straightforward, and often works seamless with the host language’s build-in data structures, functions, debugger, and third-party libraries.	Obtain the whole computation graph before execution, beneficial for optimizing the performance and memory utilization. Also convenient to implement functions such as load, save, and visualization.

Table 1: Compare the imperative and declarative for domain specific languages.

System	Core Lang	Binding Langs	Devices (beyond CPU)	Distributed	Imperative Program	Declarative Program
Caffe [7]	C++	Python/Matlab	GPU	×	×	✓
Torch7 [3]	Lua	-	GPU/FPGA	×	✓	×
Theano [1]	Python	-	GPU	×	×	✓
TensorFlow [11]	C++	Python	GPU/Mobile	✓	×	✓
MXNet	C++	Python/R/Julia/Go	GPU/Mobile	✓	✓	✓

Table 2: Compare to other popular open-source ML libraries

SparkNet

- Contributions:
 - A convenient interface for reading data from **Spark RDDs**
 - A **Scala interface** to the Caffe deep learning framework
 - A lightweight multidimensional **tensor library**
 - Use a **simple parallelization scheme** for stochastic gradient descent
- Experimental evaluation
 - AWS 5 EC2 g2.8xlarge
 - 4GPU, 60GB RAM per node
 - Default Caffe model of AlexNet
 - ImageNet dataset
 - 1GPU is the default Caffe baseline

