분산 딥러닝 오픈소스 소프트웨어 프레임워크 비교

(TensorFlow, CNTK, Petuum, MxNet)

Gunhee Kim

Computer Science and Engineering

서울대학교

SEOUL NATIONAL UNIVERSITY

August 25, 2016

Deep Learning Open Source Frameworks

TensorFlow

https://www.tensorflow.org

PETUUM

http://www.petuum.com/

dmlc mxnet

http://mxnet.io/

Microsoft CNTK

https://www.cntk.ai/

Spark Caffe

https://github.com/amplab/SparkNet

And many more …

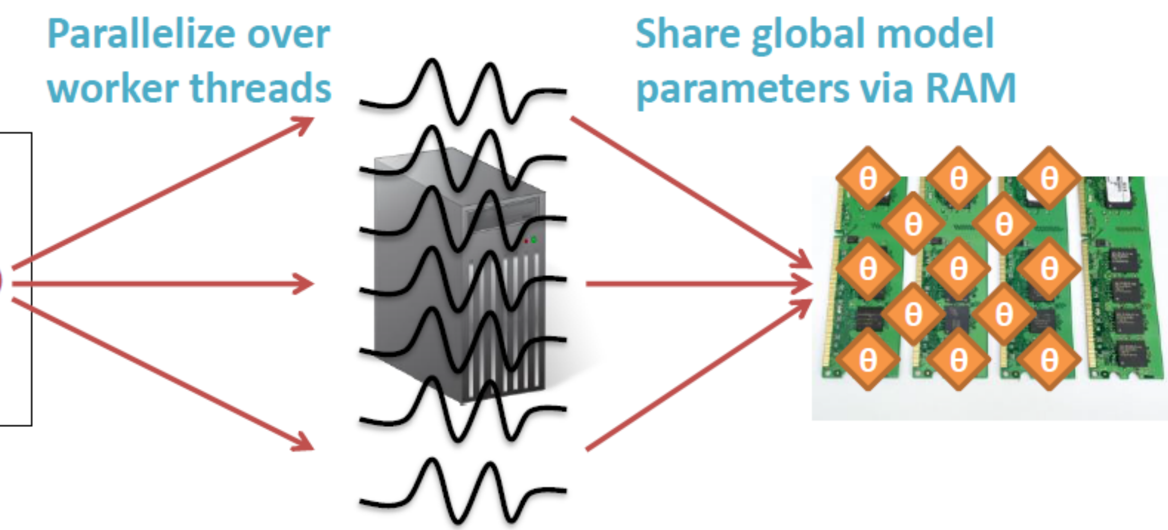
Outline

- Poseidon
  - Introduction to Petuum
  - Distributed Wait-free Backpropagation
  - Structure-Aware Message Passing Protocol
  - Staleness Consistency
- CNTK
  - 1-bit SGD
  - Block Momentum

Distributed ML: one machine to many

- **Setting**: have iterative, parallel ML algorithm
  - E.g. optimization, MCMC algorithms
  - For topic models, regression, matrix factorization, DNNs, etc

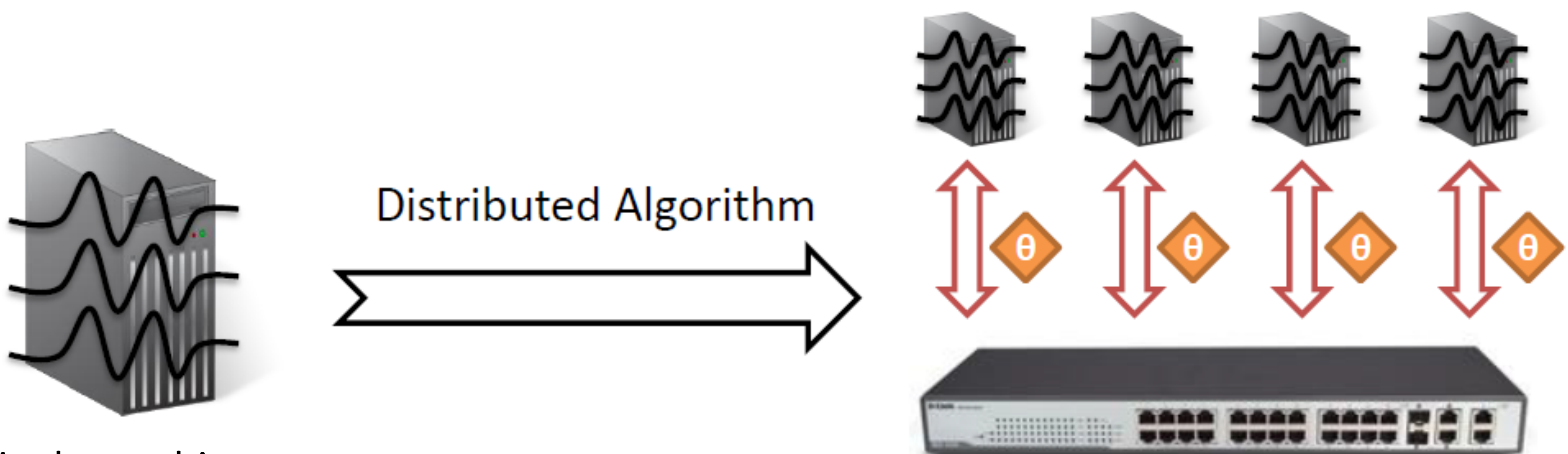
```
for (t = 1 to T) {
  doThings()
  parallelUpdate(x,θ)
  doOtherThings()
}
```



(Credits: Q. Ho's NIPS 13 slides)

Distributed ML: one machine to many

- **Want**: scale up by distributing ML algorithm
  – Must now share parameters over a network
- Seems like a simple task…

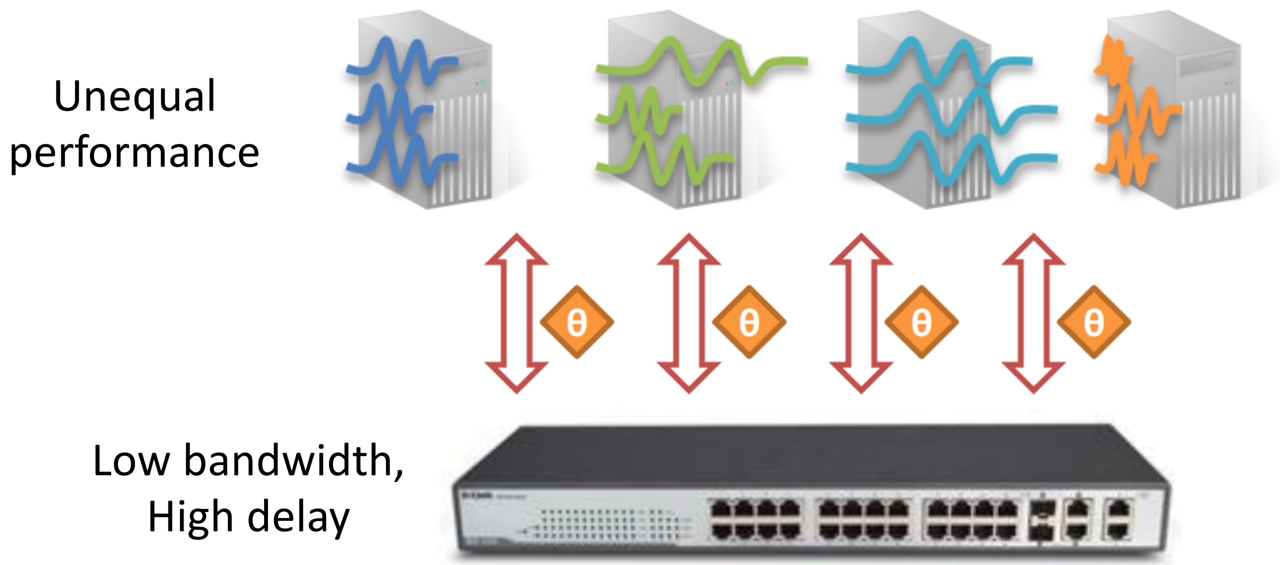


Single machine, multiple threads

Multiple machines, communicating over network switches

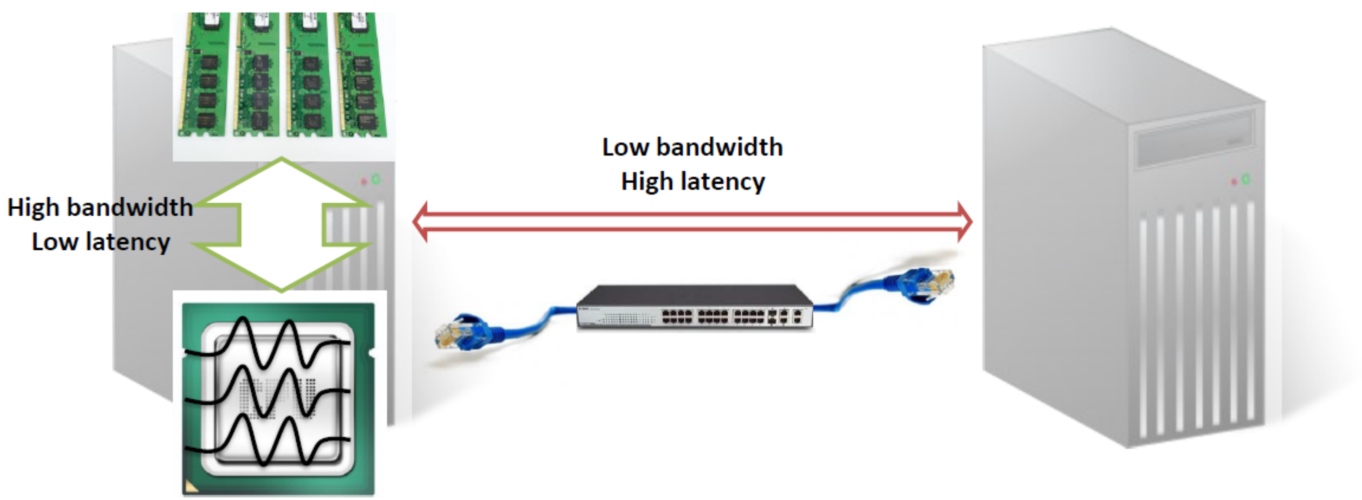
Distributed ML Challenges

- Not quite that easy…
- **Two distributed challenges:**
  - – Networks are slow
  - – "Identical" machines rarely perform equally



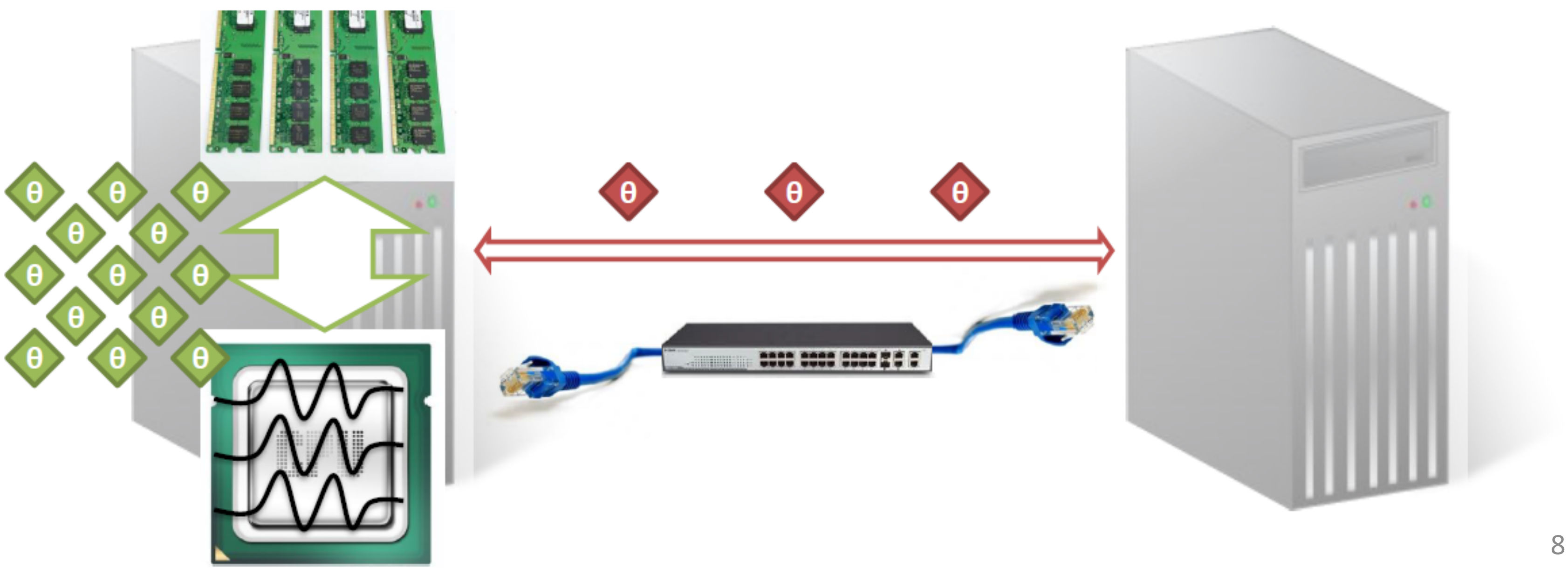
Networks are (relatively) slow

- Low network bandwidth:
  - 0.1-1GB/s (inter-machine) vs $\geq$ 20GB/s (CPU-RAM)
  - Fewer parameters transmitted per second
- High network latency (messaging time):
  - 10,000-100,000 ns (inter-machine) vs 100 ns (CPU-RAM)
  - Wait much longer to receive parameters



Networks are (relatively) slow

- Parallel ML requires frequent synchronization
  - Exchange 10-1000K scalars per second, per thread
  - Parameters not shared quickly enough → communication bottleneck

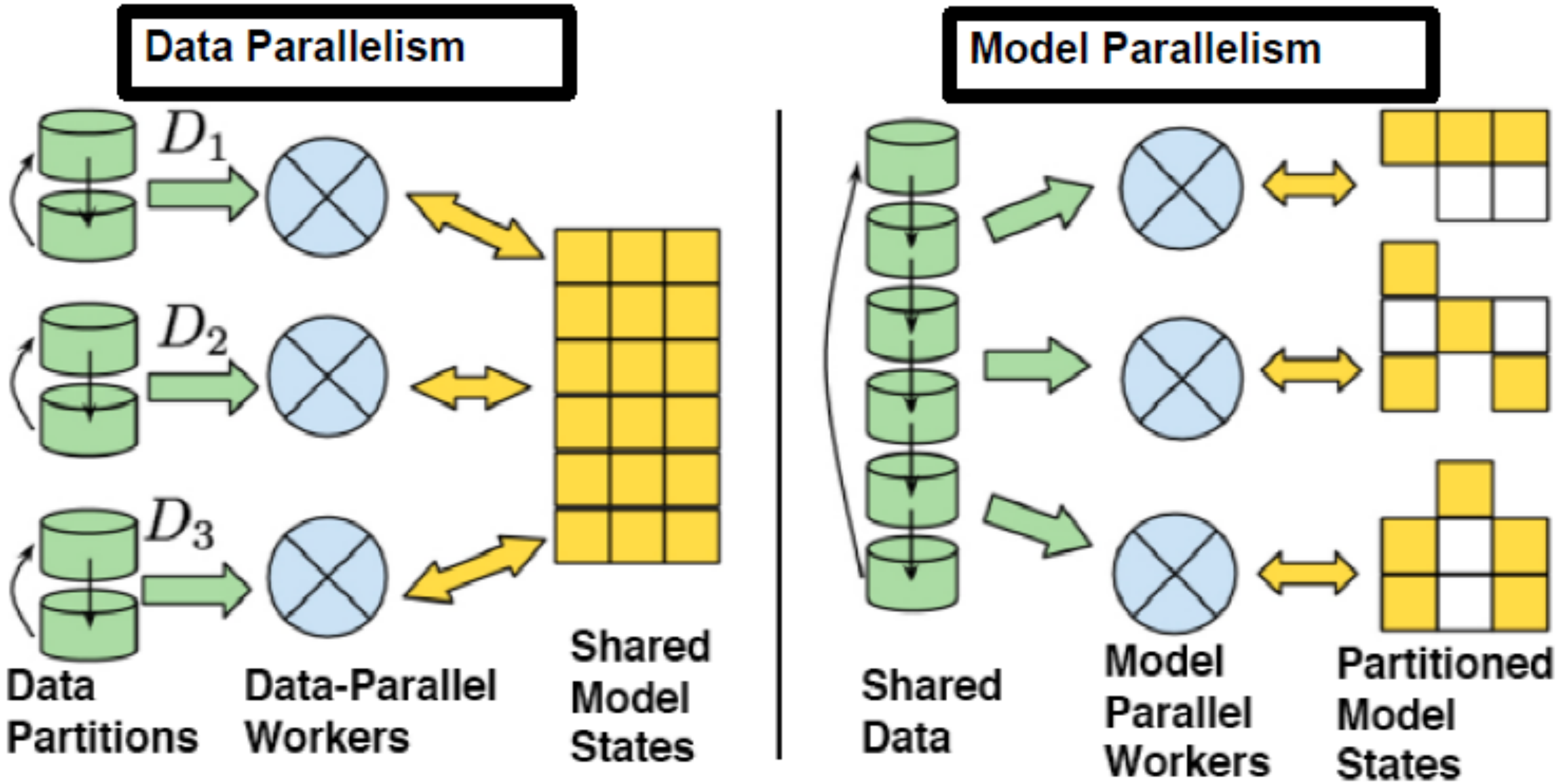


Machines don't perform equally

- Even when configured identically
- Variety of reasons:
  - Vibrating hard drive
  - Background programs; part of a distributed filesystem
  - Other users
  - Machine is a VM/cloud service
- Occasional, random slowdowns in different machines




Parallelization Strategies



$\mathcal{D}_i \perp \mathcal{D}_j \mid \theta,\ \forall i \neq j$

$\vec{\theta}_i \not\perp \vec{\theta}_j \mid \mathcal{D},\ \exists (i,j)$

(Credits: Eric Xing's WWW 15 slides)

Petuum Overview

- A distributed ML framework
  - Speeds up ML via data-, model-parallel insights
  - https://petuum.github.io/
- Key modules

Bösen parameter server for data-parallel Big Data AI & ML

Strads scheduler for model-parallel High Speed AI & ML

Poseidon for multi-GPU Distributed Deep Learning

Intrinsic Properties of ML Programs

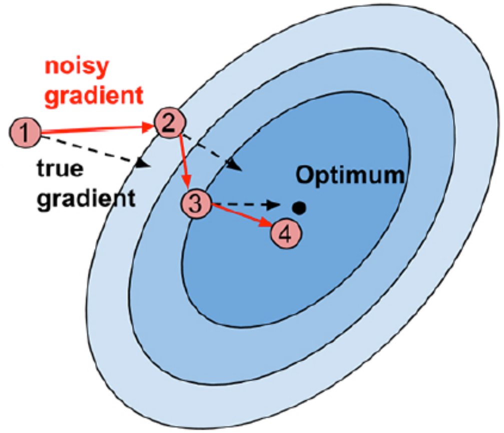
- ML is optimization-centric, and admits an iterative convergent algorithmic solution rather than a one-step closed form solution

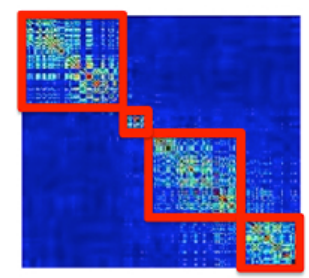
$$A^{(t)} = F(A^{(t-1)}, \Delta_{\mathcal{L}}(A^{(t-1)}, D))$$

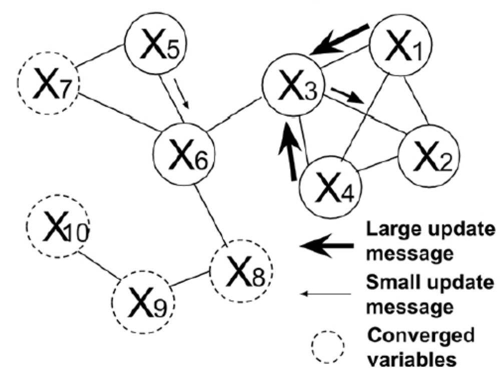
  - D: data, $\mathcal{L}$: loss
  - $\Delta_{\mathcal{L}}()$: update function performs computation on data D and model state A
  - Examples: (i) SGD and coordinate descent for fixed-point in optimization, (ii) MCMC and variational methods for graphical models, (iii) proximal optimization and ADMM for structured sparsity problems, among others

Intrinsic Properties of ML Programs

- Iterative convergent algorithms
  - **Error tolerance**: often robust against limited errors in intermediate calculations
  - **Dynamic structural dependency**: changing correlations between model parameters critical to efficient parallelization
  - **Non-uniform convergence**: parameters can converge in very different number of steps

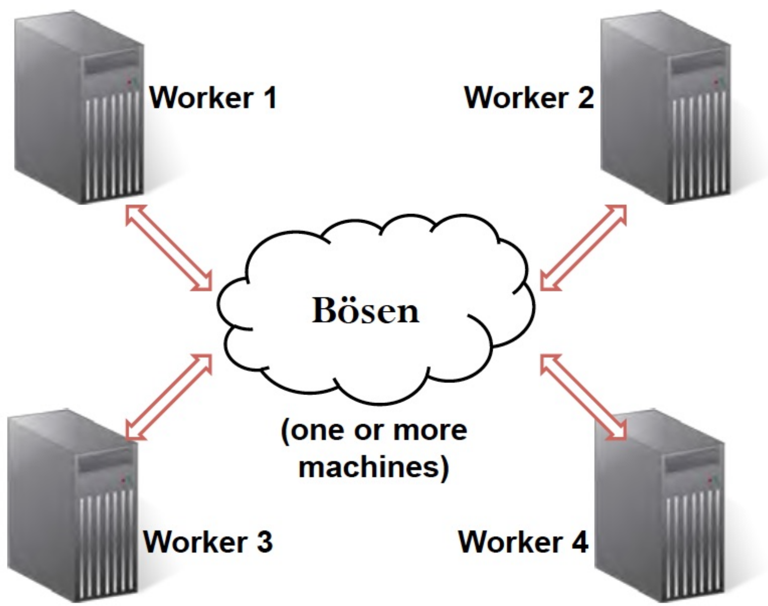






Bösen: Parameter server for data-parallelizm

- A bounded-asynchronous distributed key-value store
  - Data-parallel programming via distributed shared memory (DSM) abstraction
  - Managed communication for better parallel efficiency & guaranteed convergence



Single Machine Parallel

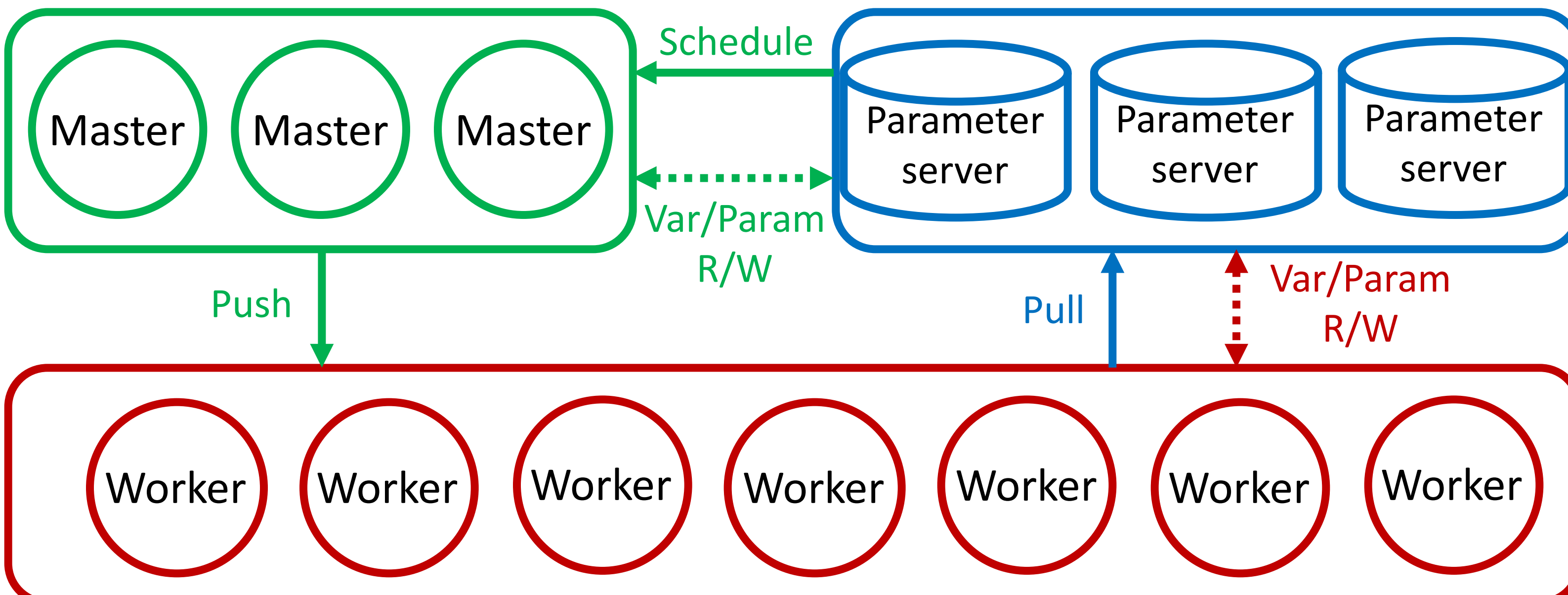
```
UpdateVar(i) {
  old = y[i]
  delta = f(old)
  y[i] += delta
}
```

Distributed with Bösen

```
UpdateVar(i) {
  old = PS.read(y,i)
  delta = f(old)
  PS.inc(y,i,delta)
}
```

Strads: Scheduler for model-parallelizm

- A structure-aware load-balancer and task prioritizer
  - Model-parallel programming via a scheduler interface
  - Explore structural dependencies and non-uniform convergence within ML models for best execution order



Poseidon

- Scalable open-source framework for large-scale distributed deep learning on CPU/GPU clusters
- http://www.petuum.com/poseidon.html
- Builds upon

PETUUM (http://petuum.github.io/)

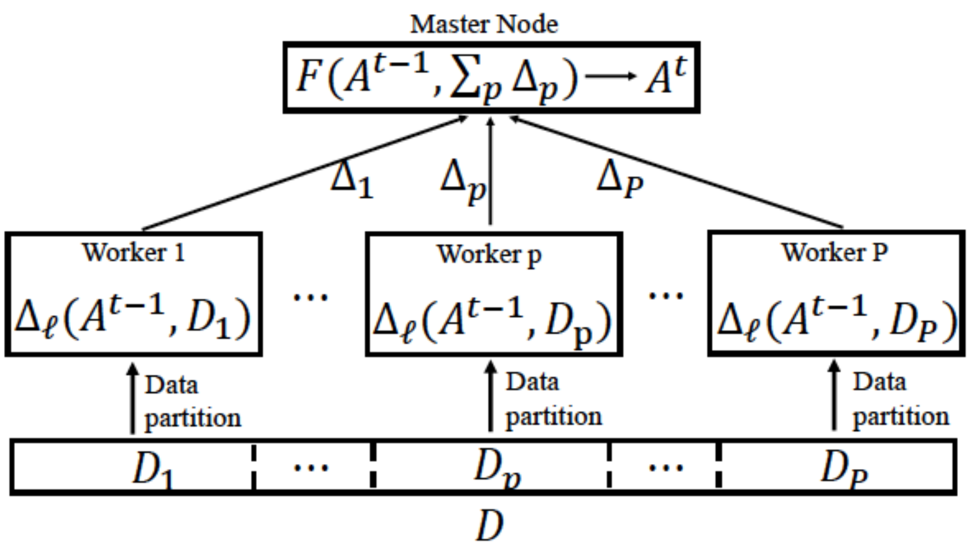
Decaf / Caffe
a Berkeley Vision Project
(http://caffe.berkeleyvision.org/)

- Maximize the speedup with a fully **data parallel** scheme for distributed deep learning

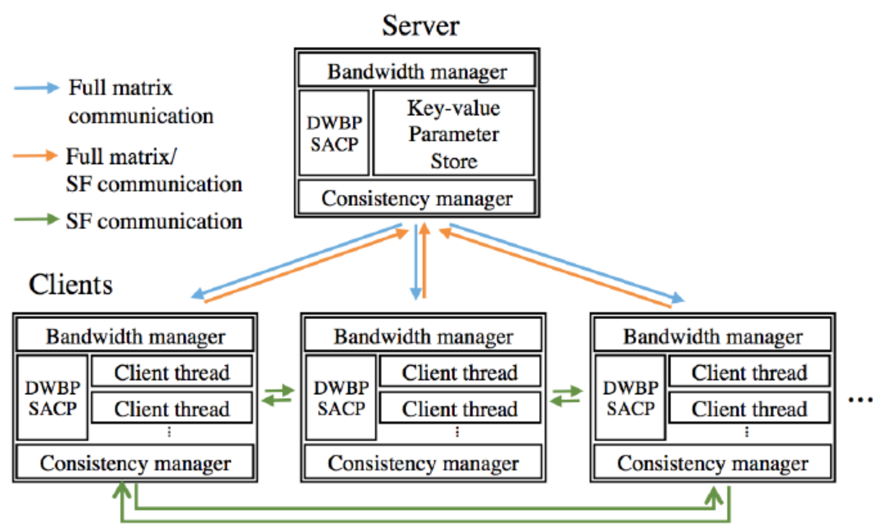
Overview: A Three-level Structure

- Server-client + multiple client threads
- Peer-to-peer + server-client communication

Abstraction of iterative-convergent algorithm in a data parallel setting

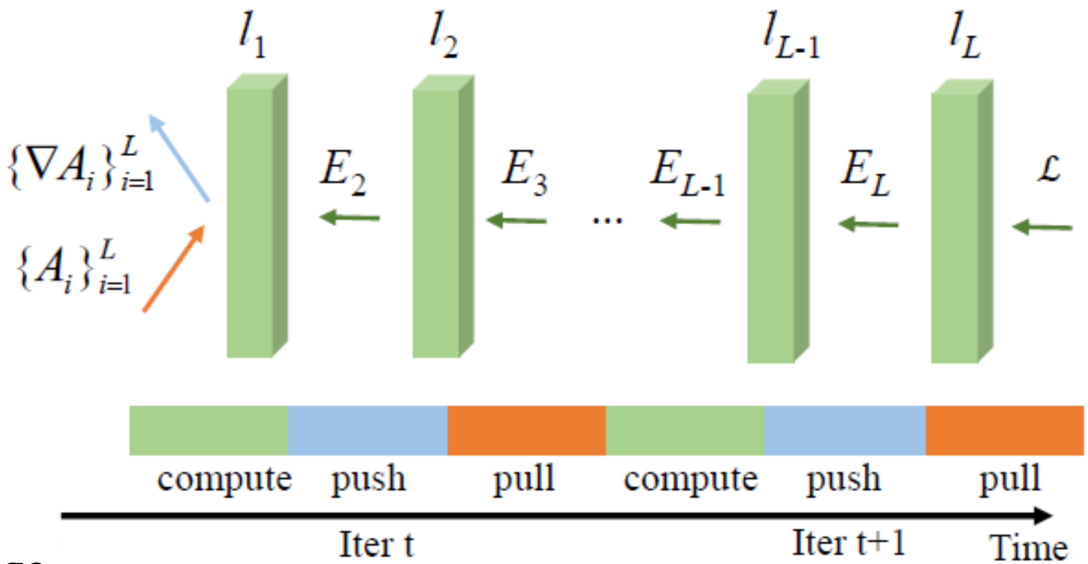


Overview of distributed architecture of Poseidon



Distributed Wait-free Backpropagation

- Original BP
  - Backpropagation followed by feedforward
  - Start communication when BP reaches l_1
  - Worker cannot proceed until communication finished



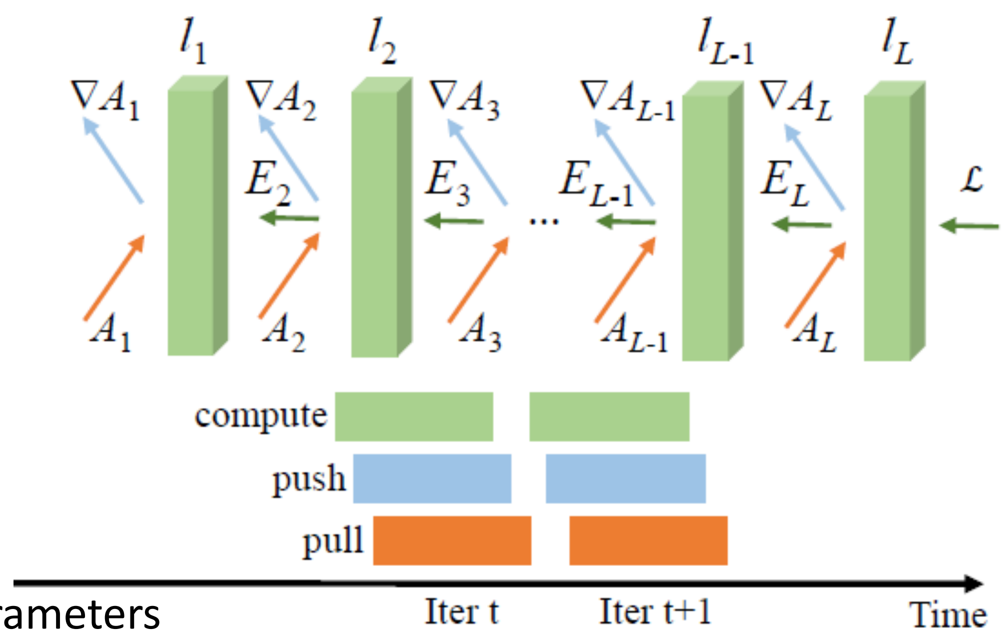
$E_{1\ldots L}$: error message

$A = \{A_i\}_{i=1}^L$: layer parameters

$\nabla A^p = \{\nabla A_l^p\}_{i=1}^L$: parameter updates

Distributed Wait-free Backpropagation

- DWBP
  - Each layer l_i do not affect upper layers $\{l_{i+1}, \dots, l_L\}$
  - Concurrently scheduled computations of lower layers and communications of upper layers during BP



$E_{1\dots L}$: error message

$A = \{A_i\}_{i=1}^{L}$: layer parameters

$\nabla A^p = \{\nabla A_l^p\}_{i=1}^{L}$: parameter updates

Structure-Aware Message Passing Protocol

- Sufficient Factor Broadcasting (SFB)
  - Parameters are in a matrix form
  - Decompose the parameter matrix into two vectors
- Structure-aware Communication Protocol (SACP)
  - Hybridizes the client-server PS scheme with the P2P scheme

Sufficient Factor Communication (SFB)

- CNN represents parameters as a set of matrices
- Parameters in FC layers exceed bandwidth of the network

Parameters	CONV Layers (#/%)	FC Layers (#/%)
AlexNet	2.3M / 3.75	59M / 96.25
VGG-16	7.15M / 5.58	121.1M / 94.42

- Sufficient factors can reduce # of parameters to be communicated

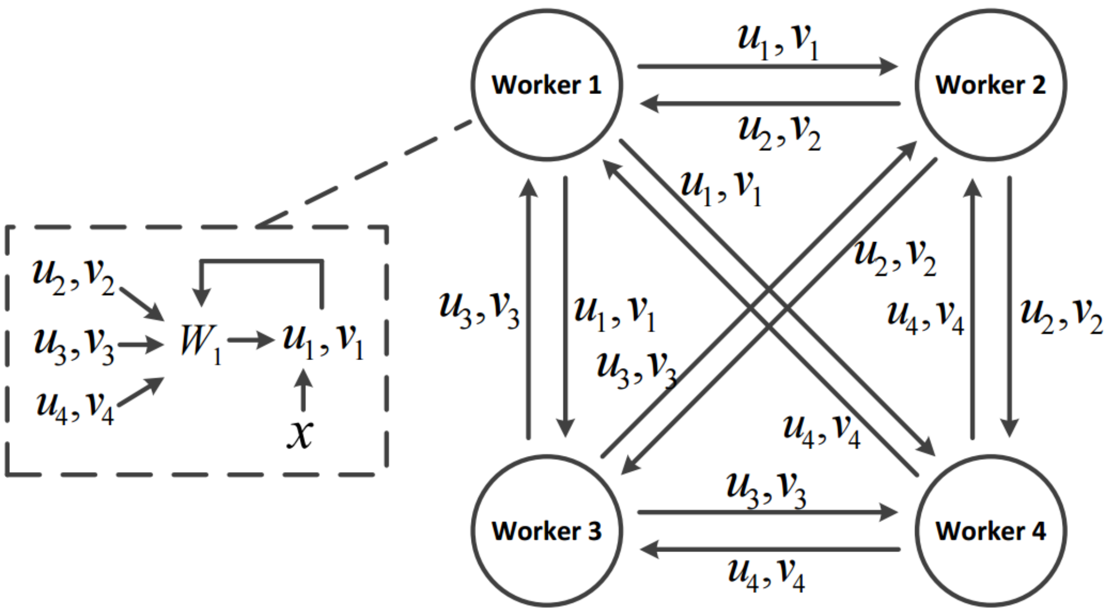
$$\nabla W = \underbrace{uv^T}_{\text{Sufficient Factors}}$$

Sufficient Factor Communication (SFB)

(1) Decouple ∇W_p into two vectors u_p and v_p

(2) Broadcast u_p and v_p to all other peer workers and receive

(3) Reconstruct $\{\nabla W_i\}_{i=1}^{P}$ using $\{u_i, v_i\}_{i=1}^{P}$ and updates.



Pengtao Xie, et al. "Distributed Machine Learning via Sufficient Factor Broadcasting." *arXiv preprint arXiv:1409.5705v2* (2015).

Sufficient Factor Communication (SFB)

- During BP, in each layer
  - (gradient) = (error message) (activation)

$$\nabla W = \frac{\partial \mathcal{L}}{\partial W} = E_{i+1} a_i^T$$

- Broadcast the two decomposed vectors to all other peer workers

Sufficient Factor Communication (SFB)

- Compared to traditional server-client on FC layer

$$\underbrace{(P\text{-}1)^2K(M+N)}_{\text{SFB}} \text{ vs } \underbrace{2PMN}_{\text{Server-client}}$$

P : # of workers
K : batch size
M, N : size of matrix

  - 7.1 times faster than server-client, since P,K << M,N in modern CNN

- Compared to Microsoft Adam on FC layer

$$\underbrace{(P\text{-}1)^2K(M+N)}_{\text{SFB}} \text{ vs } \underbrace{PK(M+N)+PMN}_{\text{Microsoft Adam}}$$

  - Adam employs SF with server-client scheme
  - 4 times faster than Microsoft Adam

Structure-aware Communication Protocol

- Intelligently determines optimal communication method

Algorithm 3: The Structure-aware Communication Protocol (SACP)

At iteration t **on worker** p:

Input: Layer l_i, $M \times N$ gradients ∇A_i^p, number of workers P, batch size K.

Task : Pull out gradients ∇A_i^p and then update A_i^p.

Server-client

if *l_i is not an FC layer* **then**
    Send ∇A_i^p to the master node.
    Synchronize updated A_i from the master node.
else
    Recast ∇A_i^p into two SFs, *i.e.* , $\nabla A_i^p = u_i^p {v_i^p}^\top$;
    if $(P-1)^2 K(M+N) \leq PK(M+N) + PMN$ **then**

SFB

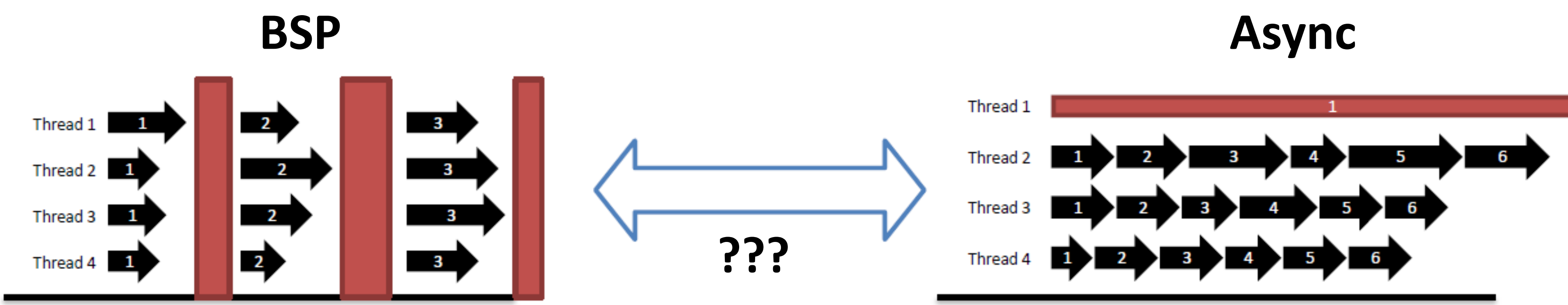
        Broadcast u_i^p, v_i^p to all other workers.
        Receive SFs $u_i^j, v_i^j, j \neq p$ from all other workers.
        Update A_i: $A_i \leftarrow A_i + \sum_j u_i^j {v_i^j}^\top + \Lambda(A_i)$.
    else

Microsoft Adam

        Send u_i^p, v_i^p to the master node.
        Synchronize updated A_i from the master node.

Staleness Consistency for Data-Parallelism

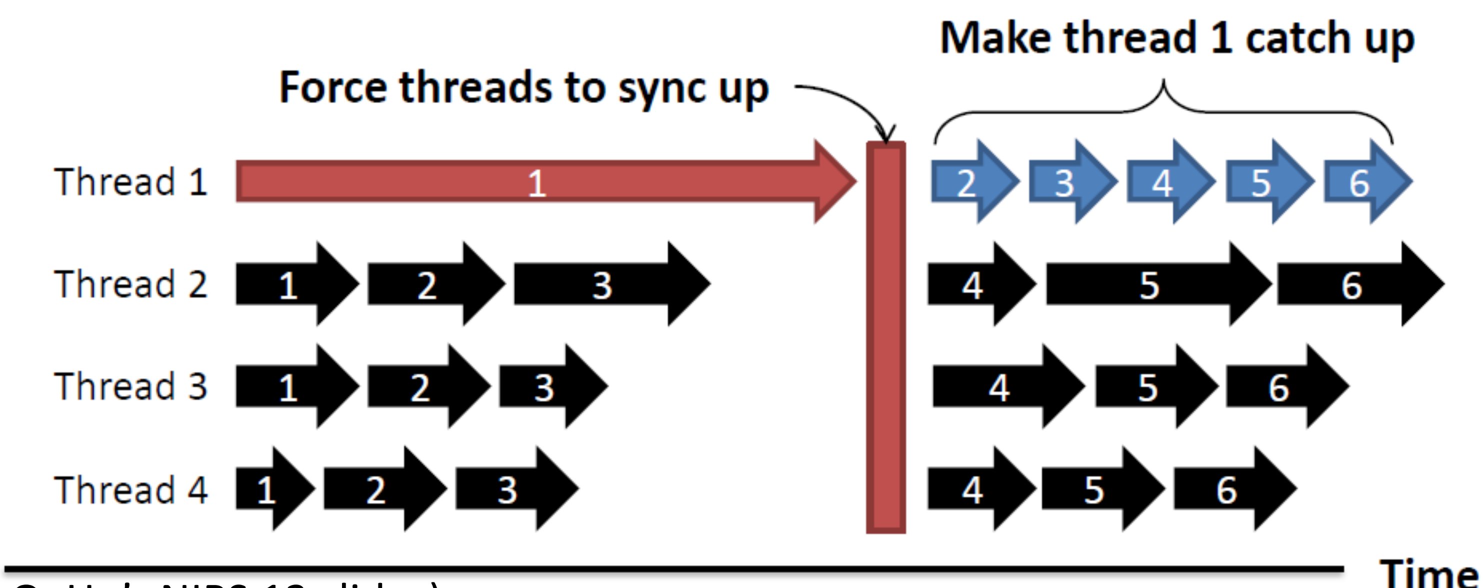
- Make parameter update consistent across the machines
- Existing ways are either safe/slow (BSP), or fast/risky (Async)
- Need “Partial” synchronicity
- Need straggler tolerance



Ho, Qirong, et al. "More effective distributed ml via a stale synchronous parallel parameter server." *Advances in neural information processing systems*. 2013.

Middle Ground

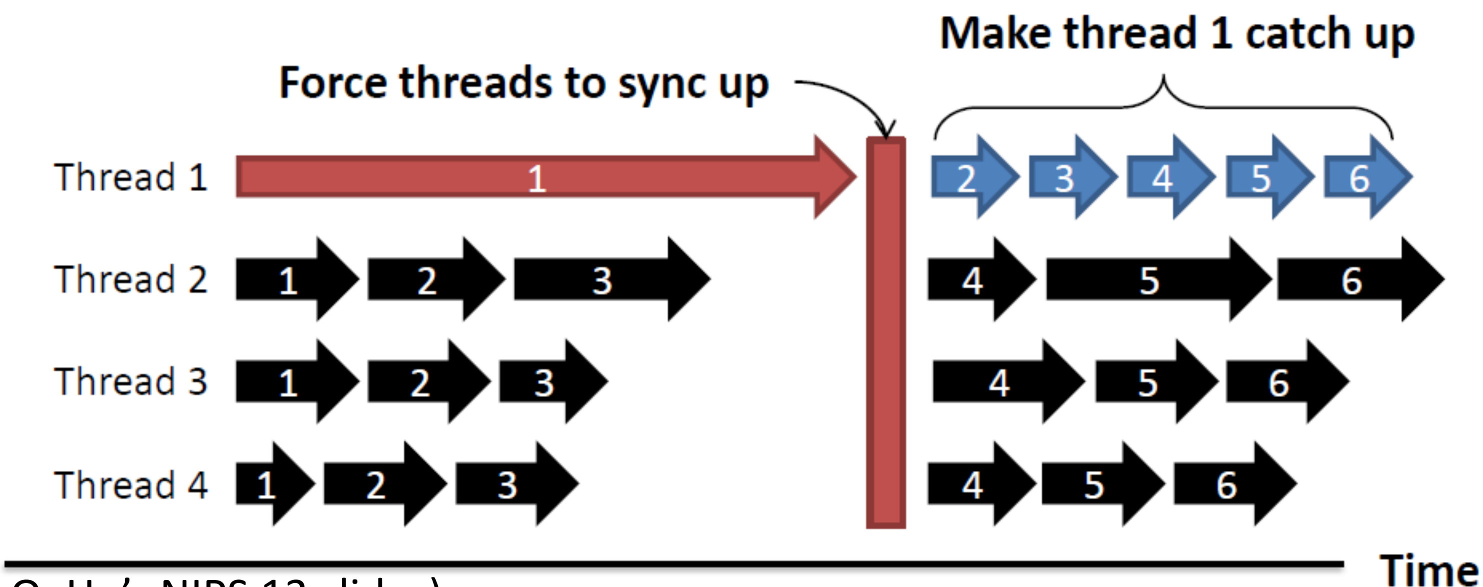
- "Partial" synchronicity
  - Spread network comms evenly (don't sync unless needed)
  - Threads shouldn't wait – but mustn't drift too far apart!
- Straggler tolerance
  - Slow threads must somehow catch up



(Credits: Q. Ho's NIPS 13 slides)

Middle Ground

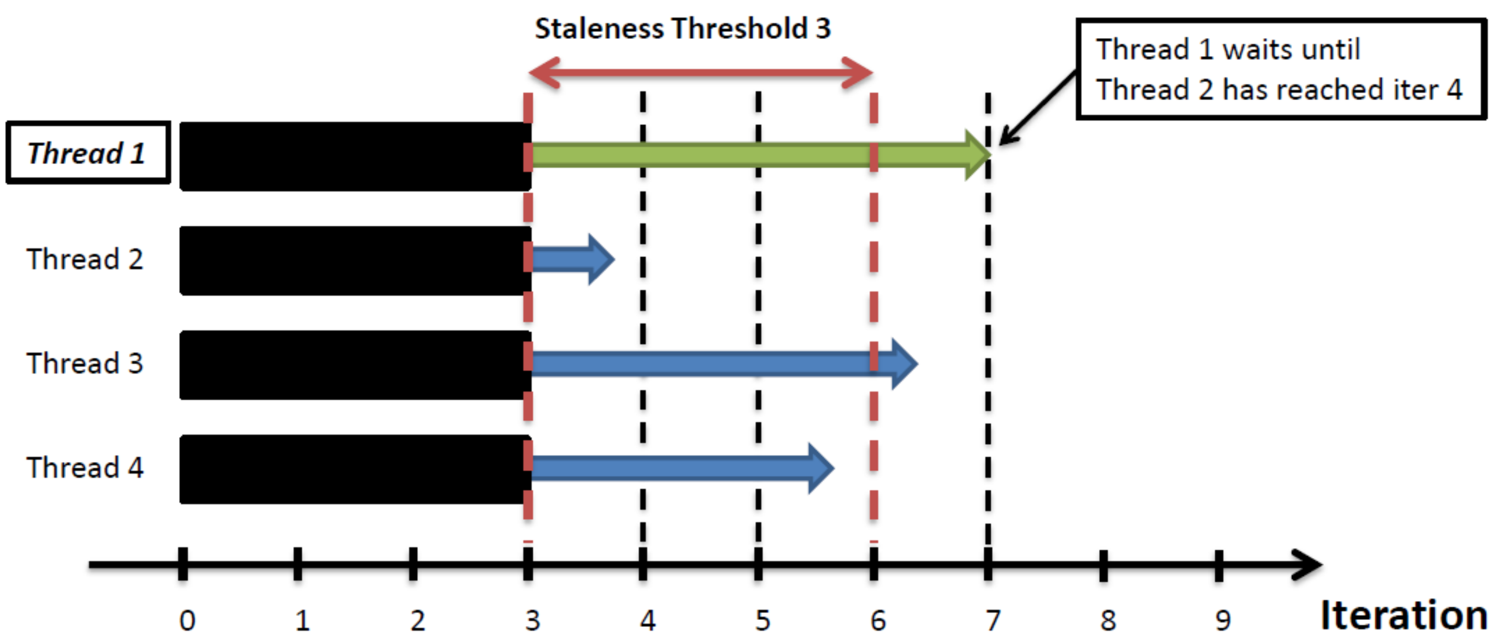
How do we realize this?



(Credits: Q. Ho's NIPS 13 slides)

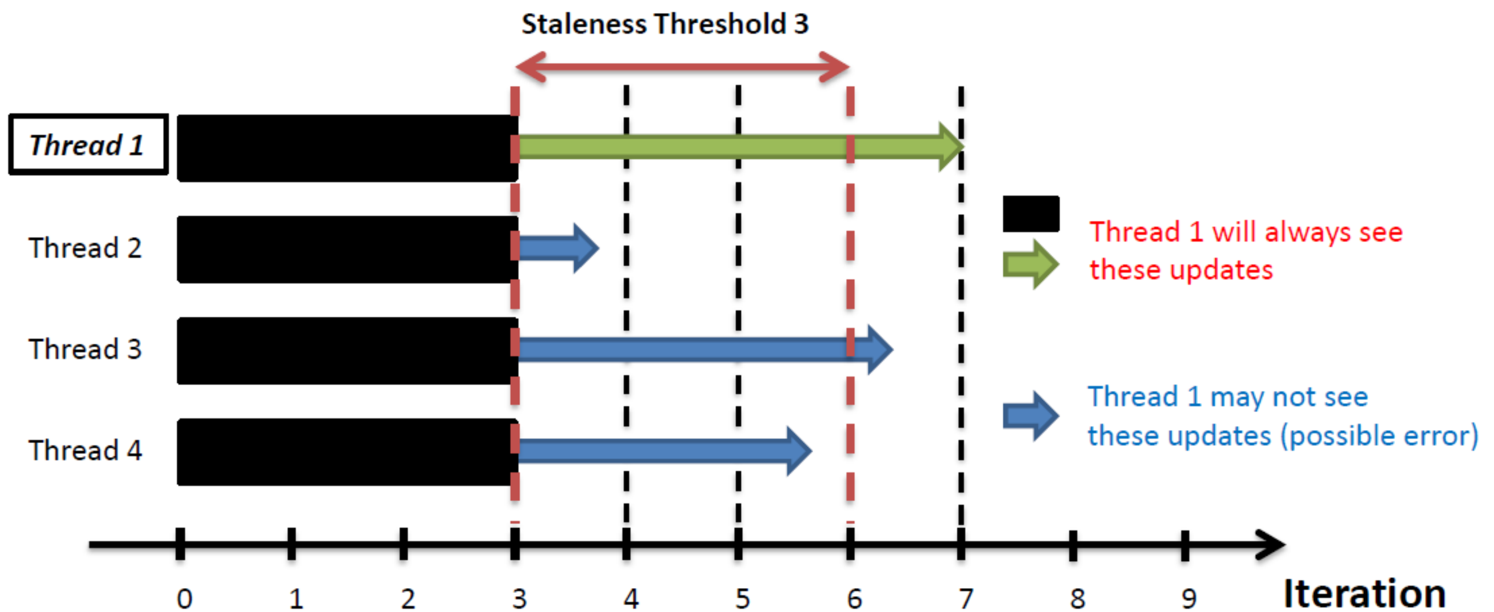
Stale Synchronous Parallel (SSP)

- Note: x-axis is now <u>iteration count</u>, not time!
- Fastest/slowest threads not allowed to drift >S iterations apart
- Threads cache local (stale) versions of the parameters, to reduce network syncing



Stale Synchronous Parallel (SSP)

- Protocol: check cache first; if too old, get latest version from network
- Consequence: fast threads must check network every iteration
  - Slow threads only check every S iterations – fewer network access, so catch up!



SSP provides best-of-both-worlds

- SSP combines best properties of BSP and Async

- BSP-like convergence guarantees
  - Threads cannot drift more than S iterations apart
- Asynchronous-like speed
  - Threads usually don't wait (unless there is drift)

- SSP is a spectrum of choices
  - Can be fully synchronous (S=0) or very asynchronous ($S \rightarrow \infty$)
  - Or just take the middle ground, and benefit from both!

BWBP + SACP + SSP

Algorithm 1: CNN training with data-parallelism on Poseidon

Slave nodes:

```
Partition training data D equally into {D_i}_{i=1}^P and
distribute them to all P nodes.
Replicate the initial model parameters A to every worker
thread p as A_p.
for t = 1 to T do
    foreach worker thread p ∈ {1,2,··· ,P} do
        Take a batch of training data D_p^t from D_p.
        Perform forward pass.
        Perform backpropagation pass following the
        DWBP algorithm (See Algorithm.2).
        Update local synchronization states to the
        consistency manager (see section 4).
```

Master node:

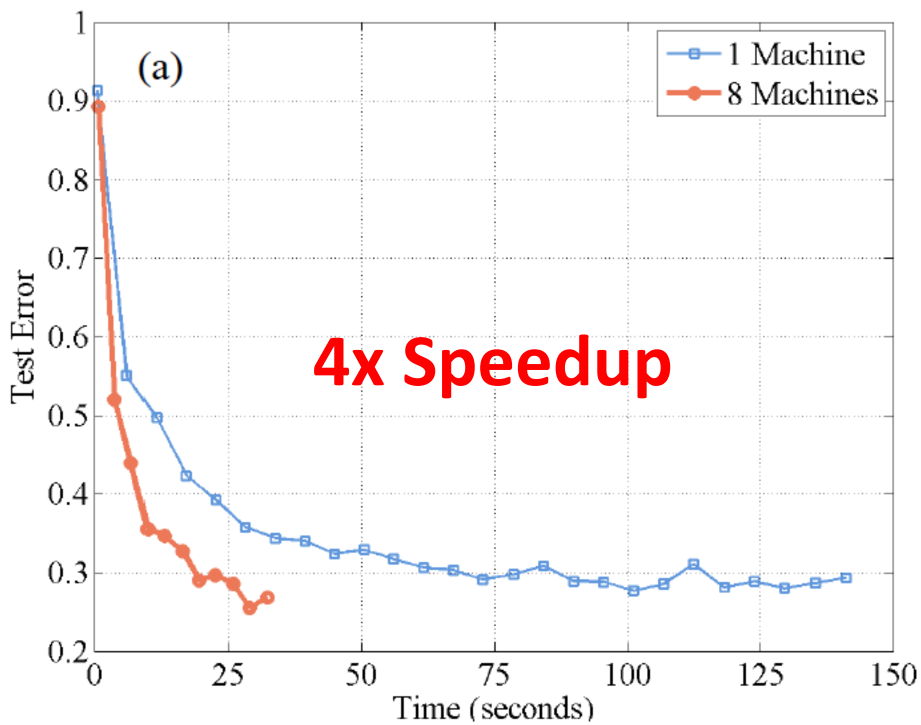
```
for t = 1 to T do
    Collect gradients that are sent by worker nodes.
    Updates the part of model parameters for which a
    corresponding gradient is received.
    Push updated model parameters to worker nodes
    according to the consistency manager.
```

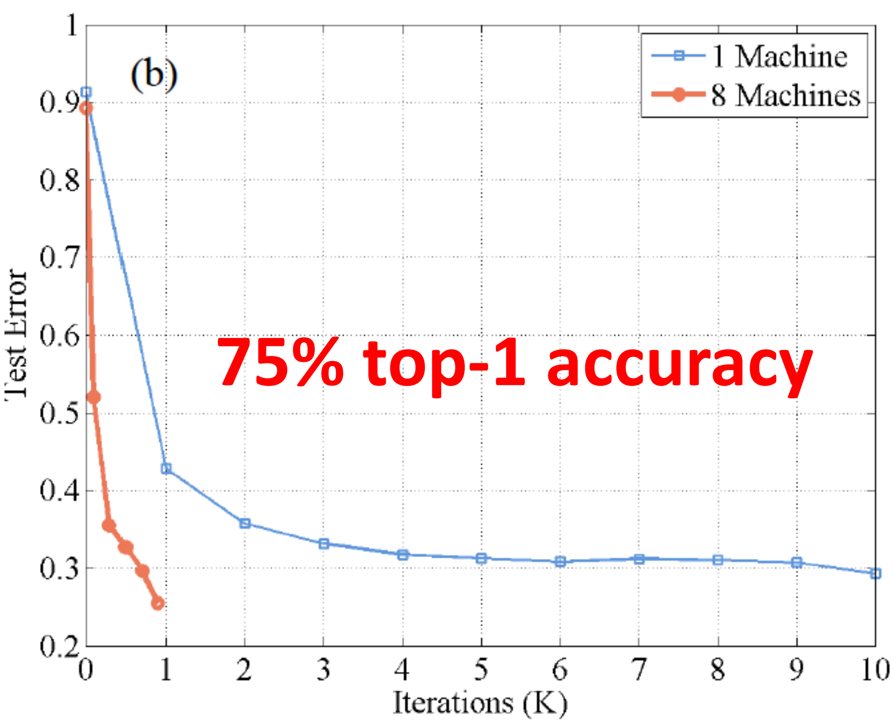
Evaluation

- Cluster Configuration
  - 4 x 16 core 2.1GHz AMD Operation 6272 CPUs
  - 128 GB of RAM
  - NVIDIA Tesla K20C GPU with 4799MB memory
  - 40GBe network for connecting NFS and workers
  - Caffe with CUDA 6.5 and CUDNN R2
- Datasets
  - CIFAR-10
  - ILSVRC2012 (AlexNet and GoogleNet)
  - ImageNet 22k (with other frameworks)

Classification on CIFAR-10

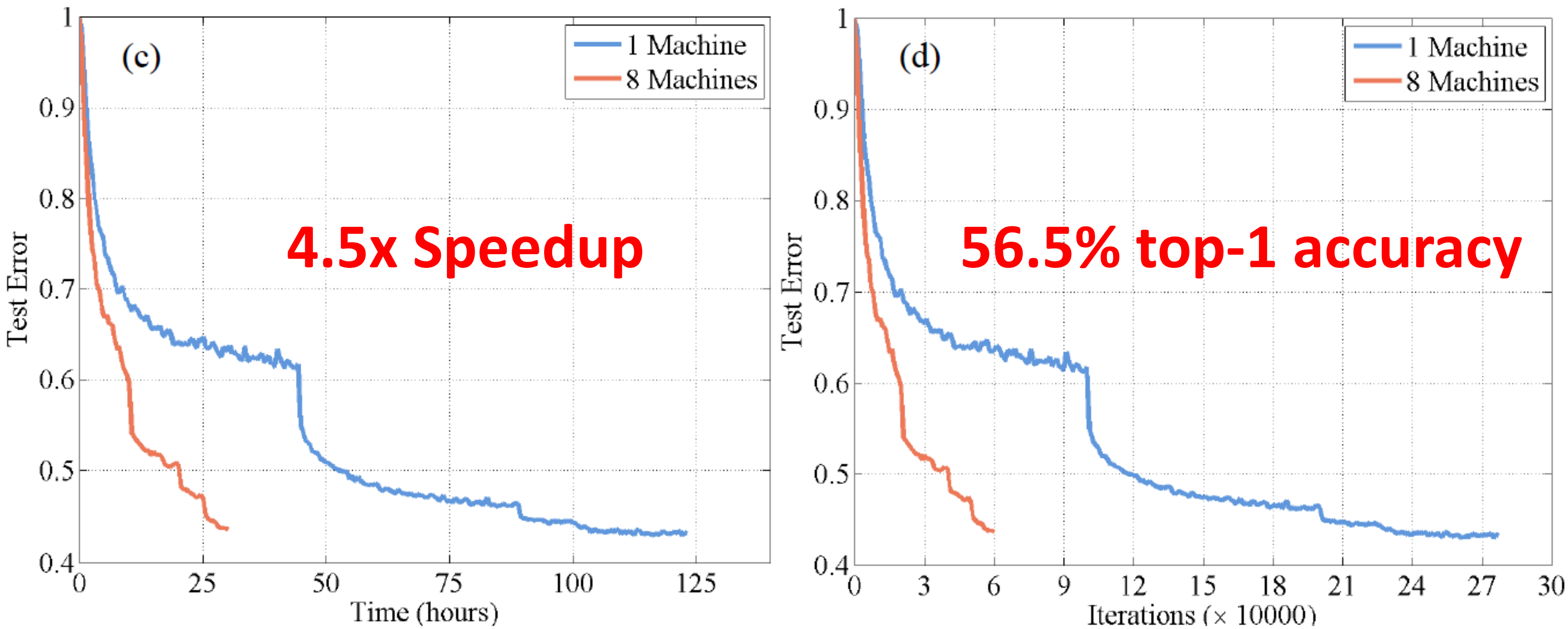
- 32 x 32 images of 10 classes, with 6K images per class
- 3 CONV + 1 FC + Softmax, total 145,578 parameters
- 8 GPU nodes





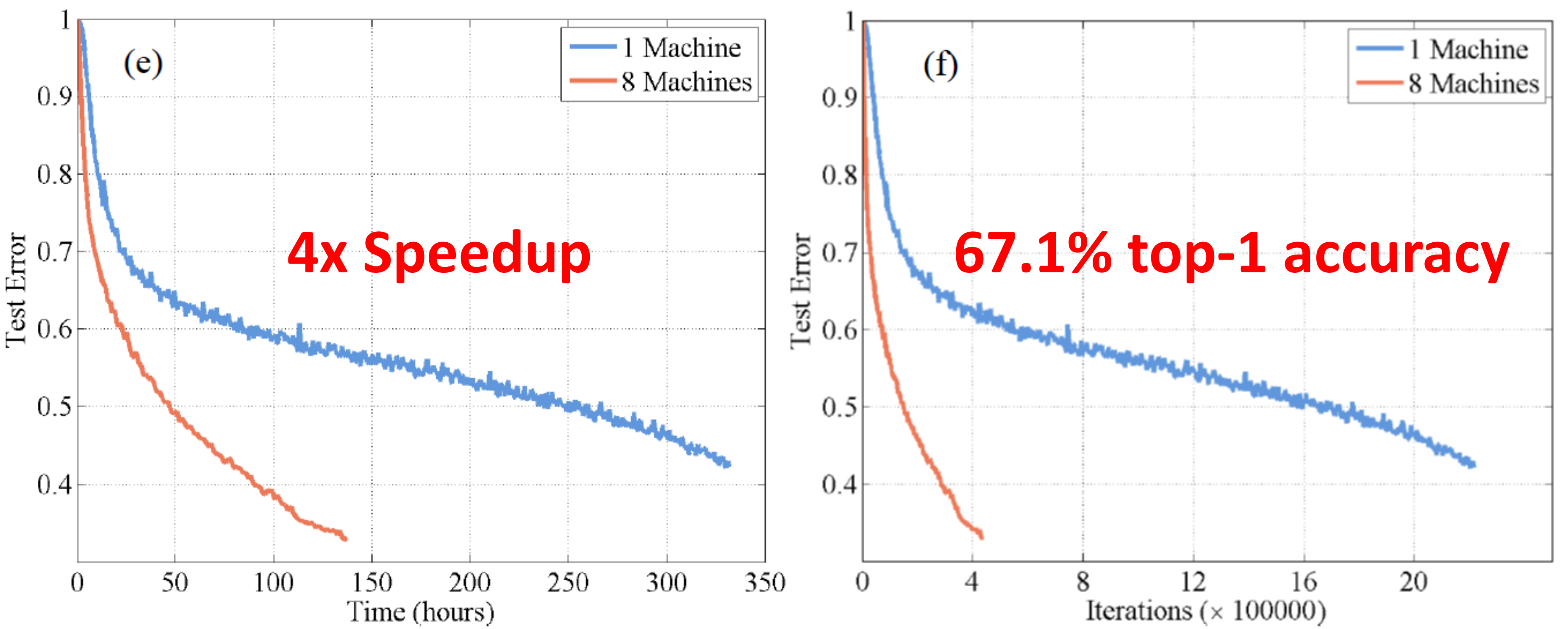
Classification on ILSVRC 2012 with AlexNet

- 256 x 256 x 3 images of 1k classes, total 1.3M images
- 5 CONV + 2 FC + Softmax, total 61.3M parameters
- 8 GPU nodes



Classification on ILSVRC 2012 with GoogLeNet

- 256 x 256 x 3 images of 1k classes, total 1.3M images
- 22-layer CNN, total 5M parameters
- 8 GPU nodes



Classification on ImageNet 22k

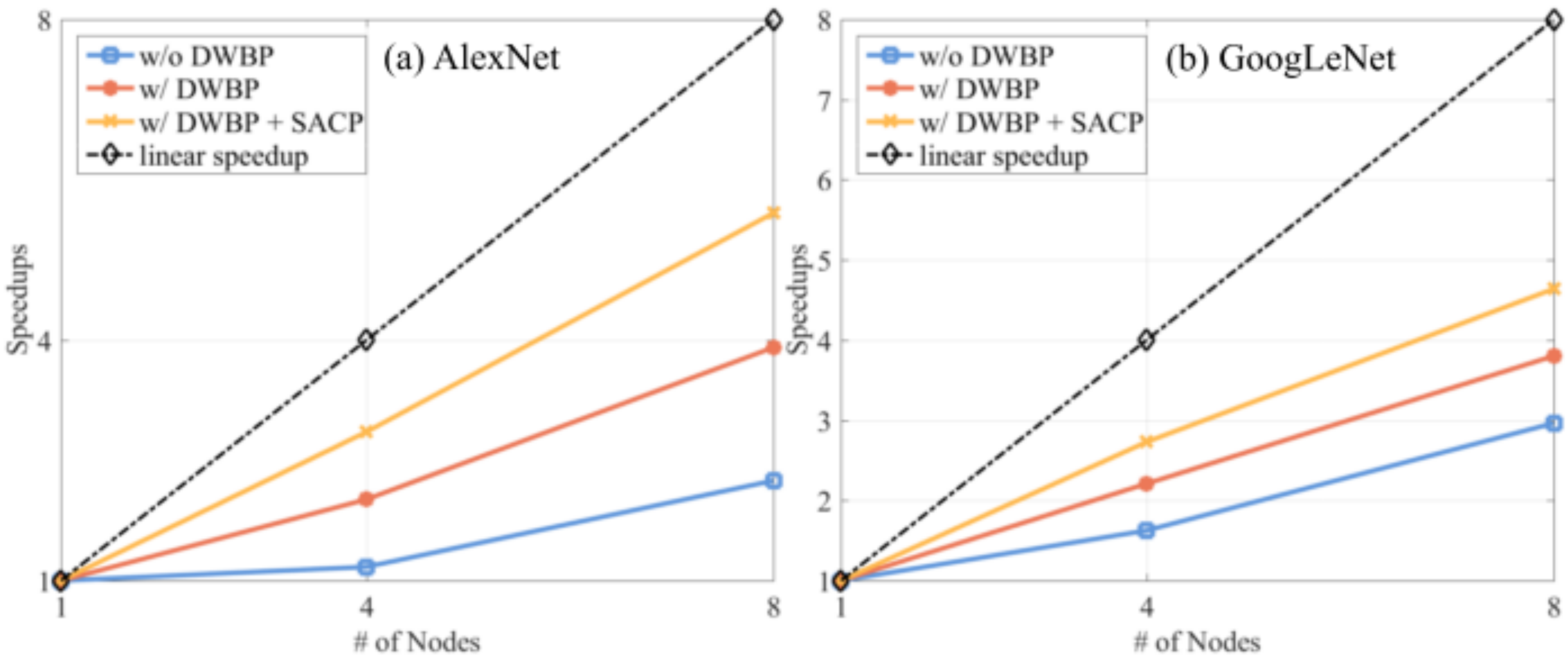
- 14,197,087 labeled images from 21,841 categories
- AlexNet-like CNN: 5 CONV + 2 FC, total 120M parameters
- 7.9% higher performance compared to Le et al.'s framework

Framework	Data	# machines/cores	Time	Train accuracy	Test accuracy
Poseidon	7.1M ImageNet22K for training, 7.1M for test	8 / 8 GPUs	3 days	41%	23.7%
Adam [2]	7.1M ImageNet22K for training, 7.1M for test	62 machines/?	10 days	N/A	29.8%
MxNet [20]	All ImageNet22K images for training, no test	1/4 GPUs	8.5 days	37.19%	N/A
Le *et al.* [15] **w/ pretrain**	7.1M ImageNet 22K, 10M unlabeled images for training, 7.1M for test	1,000/1,6000 CPU cores	3 days	N/A	15.8%

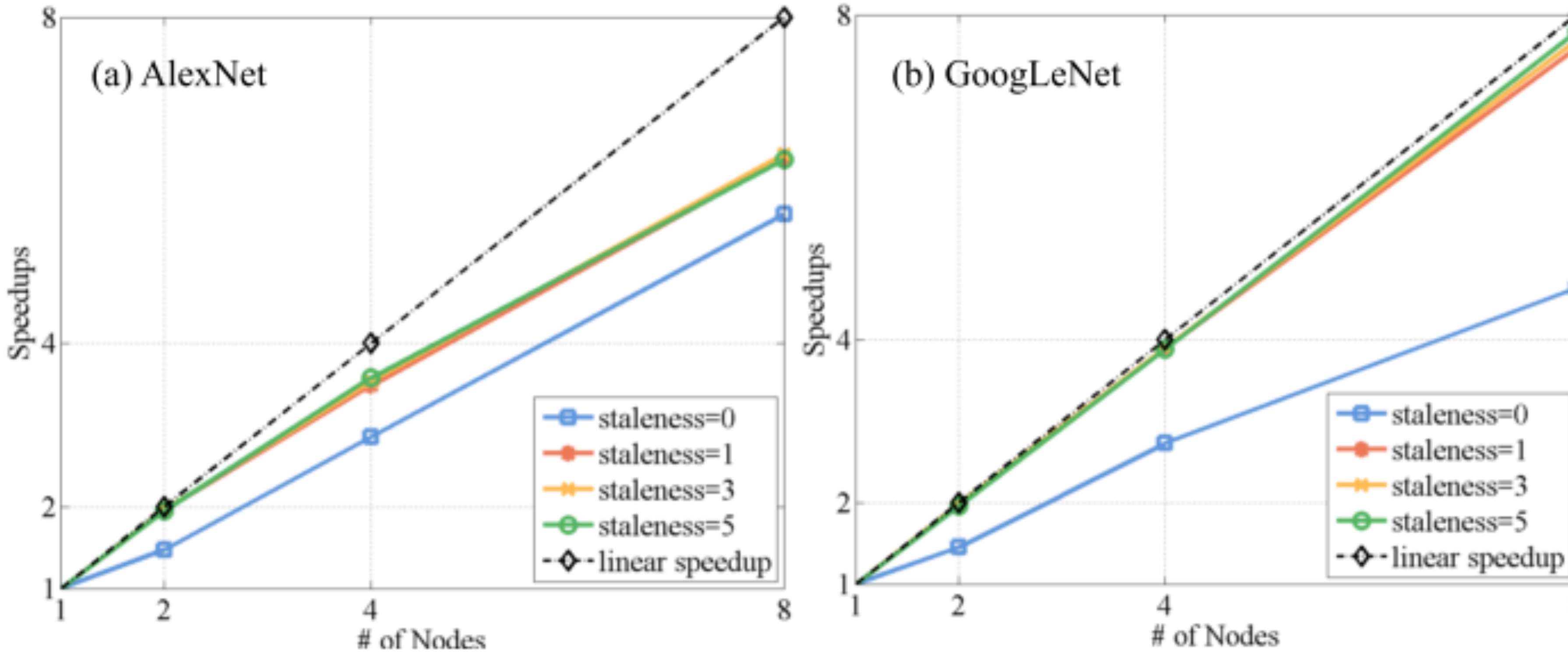
Table 3. Comparisons of the image classification results on ImageNet 22K.

DWBP and SACP

- Set staleness to 0 (i.e. BSP)
- More loss when increasing the number of nodes



SSP Consistency Model



Conclusion

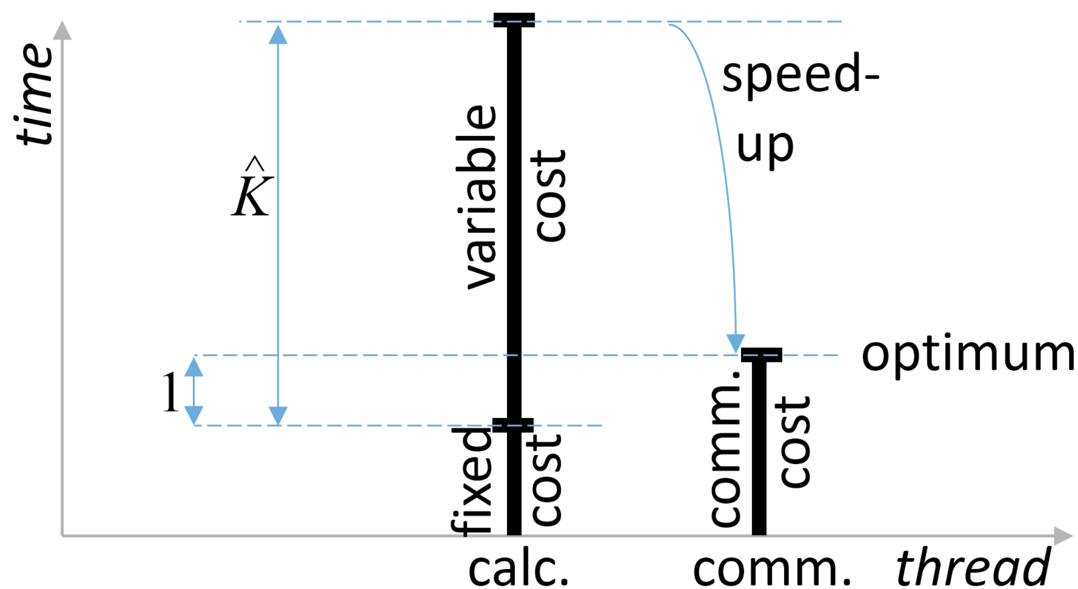
- Present Poseidon, a highly scalable and efficient system architecture for large-scale deep learning on GPU clusters.
- Poseidon achieves state-of-the-art speedups in accelerating the training of modern CNN structures, at the same time guarantee the correct convergence

Outline

- Poseidon
  - Introduction to Petuum
  - Distributed Wait-free Backpropagation
  - Structure-Aware Message Passing Protocol
  - Staleness Consistency
- CNTK
  - 1-bit SGD
  - Block Momentum

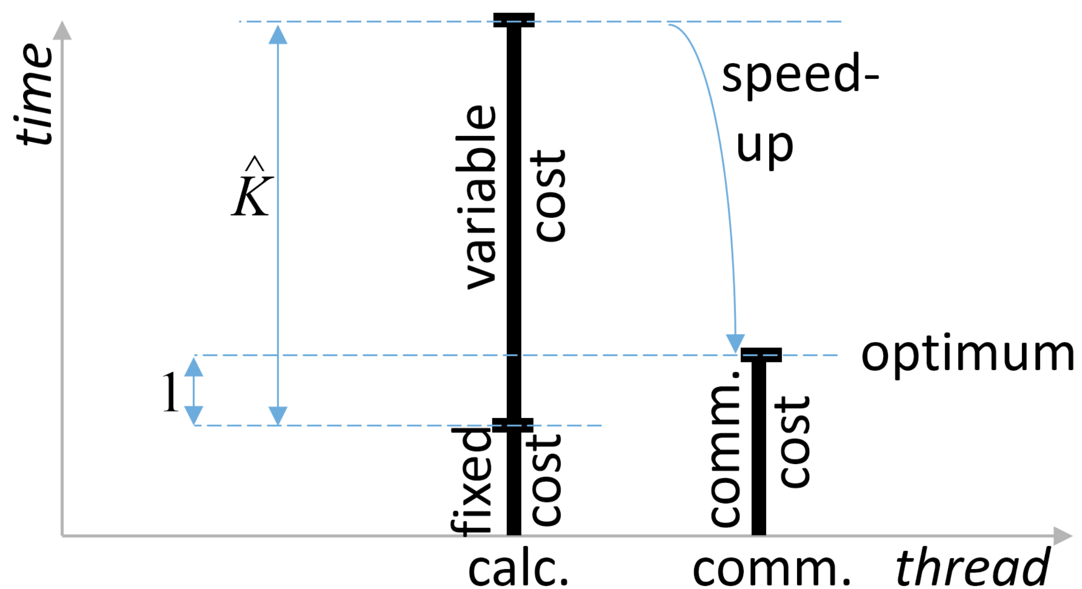
Data Parallel Training

- Data-parallelism
  - Distribute each mini-batch over workers, then aggregate
- Challenge
  - Communication cost
  - Optimal iff computation and communication time per mini-batch is equal (assuming overlapped processing)



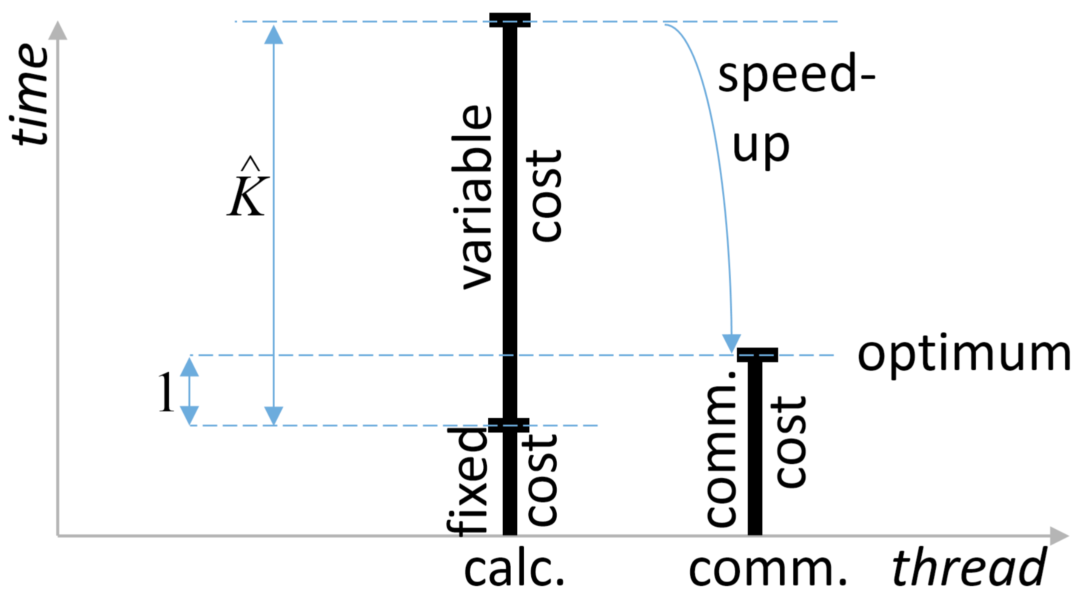
Data Parallel Training

- Two approaches
  - Focusing on communication than computation
  - Communicate less
  - Communicate less often



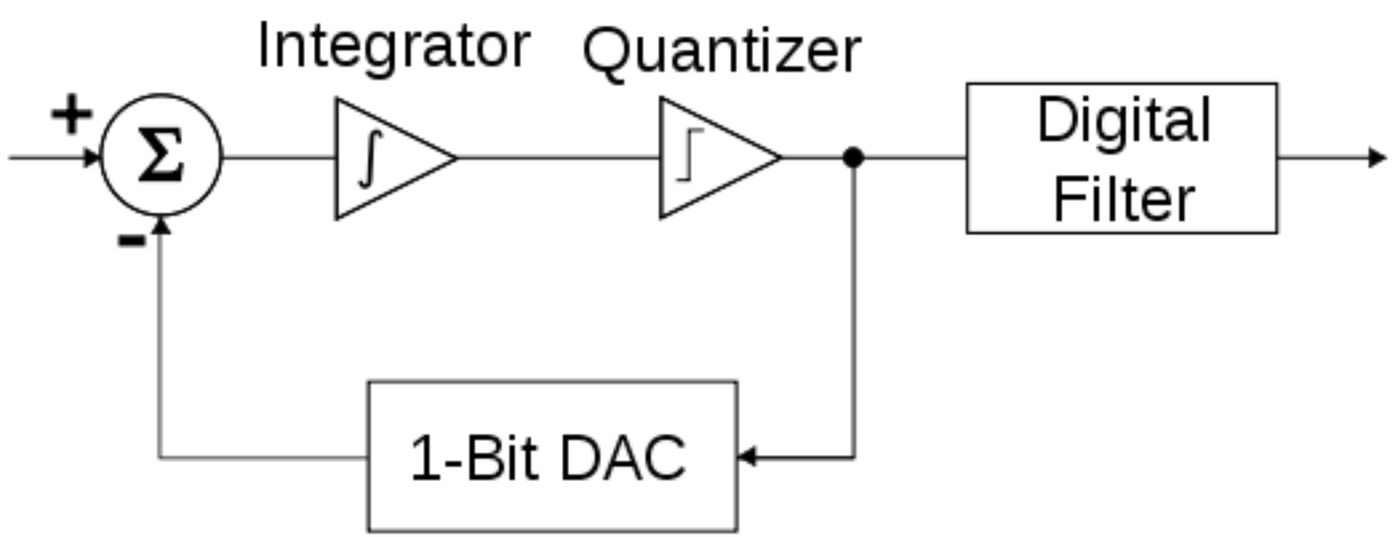
Data Parallel Training

- Two approaches
  - Focusing on communication than computation
  - Communicate less → 1-bit SGD
  - Communicate less often



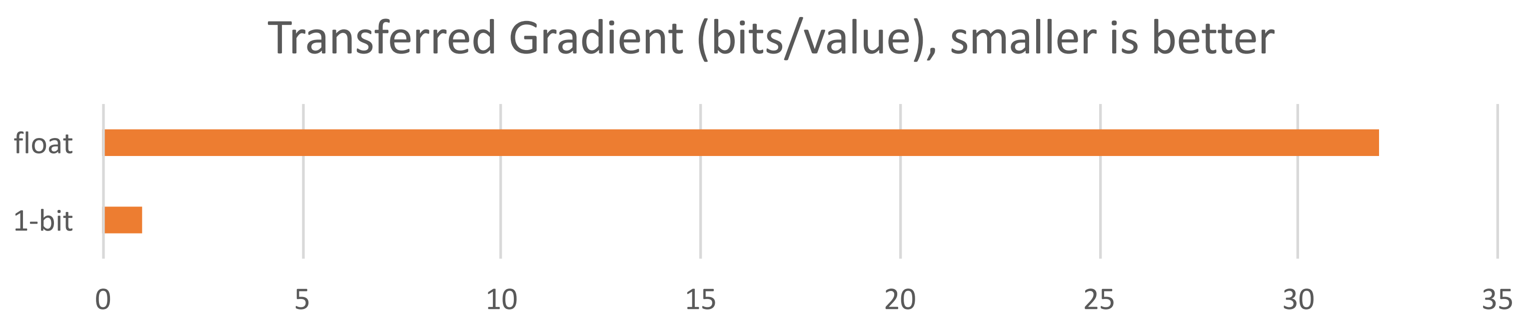
A Key Idea of 1-Bit SGD

- Inspired by Sigma-Delta modulation
  - A method for encoding analog signals into digital signals using only a single 1-bit DAC
  - http://www.analog.com/en/design-center/interactive-design-tools/sigma-delta-adc-tutorial.html



A Key Idea of 1-Bit SGD

- Transmit a single-bit update for each subgradient dimension
  - e.g. Instead of $G = \{-.1, 0.3, \dots, 0.2\}$, use $G = \{-\tau, \tau, \dots, \tau\}$



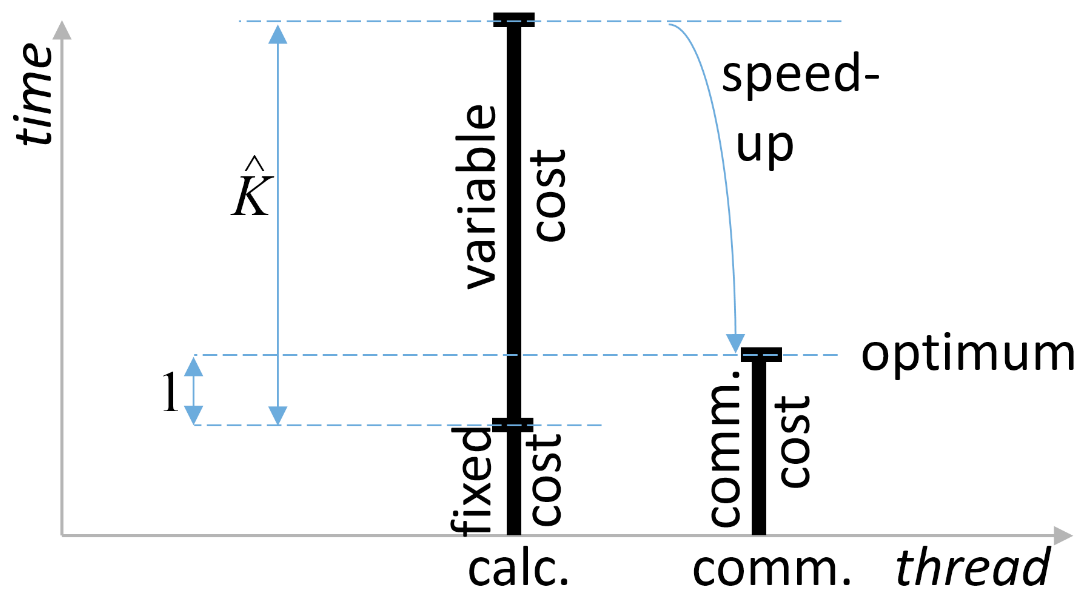
A Pseudo Code of a Mini-batch cycle in a Single Node in Distributed SGD

1. Receive and uncompress any weight update messages from other compute nodes and apply them to the local replica of the DNN
2. Load feature vectors and supervision targets for a mini-batch
3. Compute a sub-gradient $G^{(s)}$ by Back-Propagation
4. Aggregate the sub-gradient in the gradient residual $G^{(r)} = G^{(r)} + G^{(s)}$
5. Reset the message map M
6. For each element ${g_i}^{(r)}$ of $G^{(r)}$:
    - If ${g_i}^{(r)} > \tau$ then
        - push the pair $\{i, +\tau\}$ to the message M
        - Subtract τ from residual: ${g_i}^{(r)} = {g_i}^{(r)} - \tau$
    - Else if ${g_i}^{(r)} < \tau$ then
        - push the pair $\{i, -\tau\}$ to the message M
        - Add τ to the residual: ${g_i}^{(r)} = {g_i}^{(r)} + \tau$
7. Compress M and send to all other compute nodes
8. Apply M to the local replica of the DNN

Strom, Nikko. "Scalable distributed dnn training using commodity gpu cloud computing." INTERSPEECH 2015.

Data Parallel Training

- Two approaches
  - Communicate less → **1-bit SGD**
  - Communicate less often → **Block Momentum**

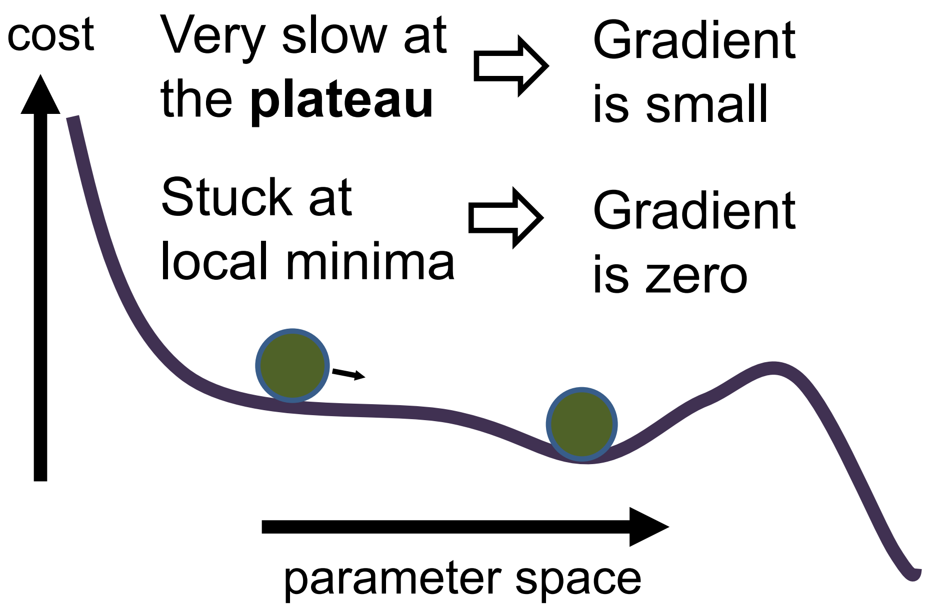


Block Momemtum

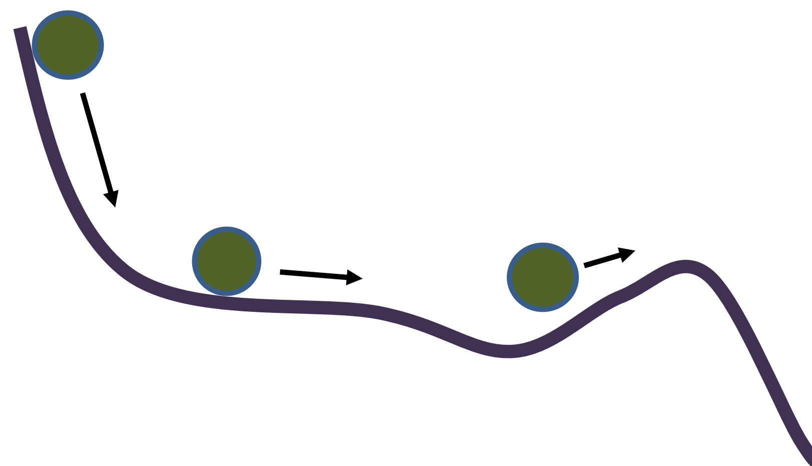
- A recent, effective parallelization method
- Goal: avoid to communicate after every mini-batch
  - Run a block of many mini-batches without synchronization
  - Then exchange and update with “block gradient”
- Problem: taking such a large step causes divergence

Gradient Descent with Momentum

Without momentum

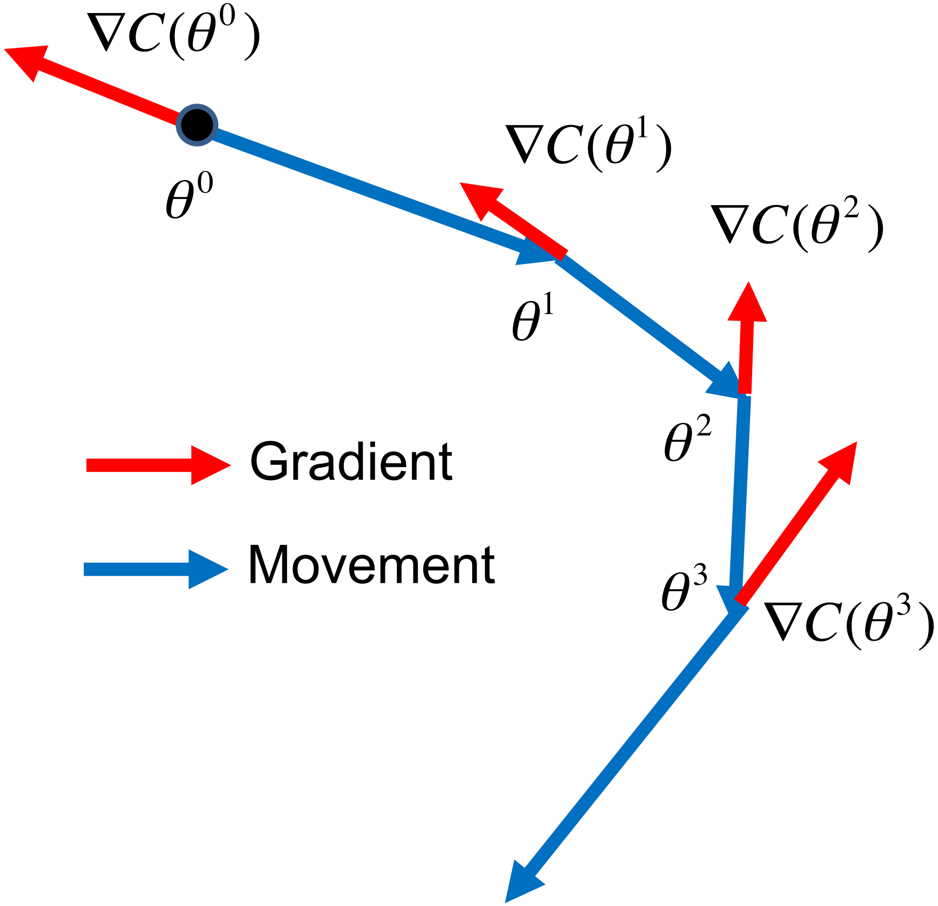


With momentum



(Credits: Hung-yi Lee’s More Tips for Training Neural Network)

Original Gradient Descent



Start at position θ^0

Compute gradient at θ^0

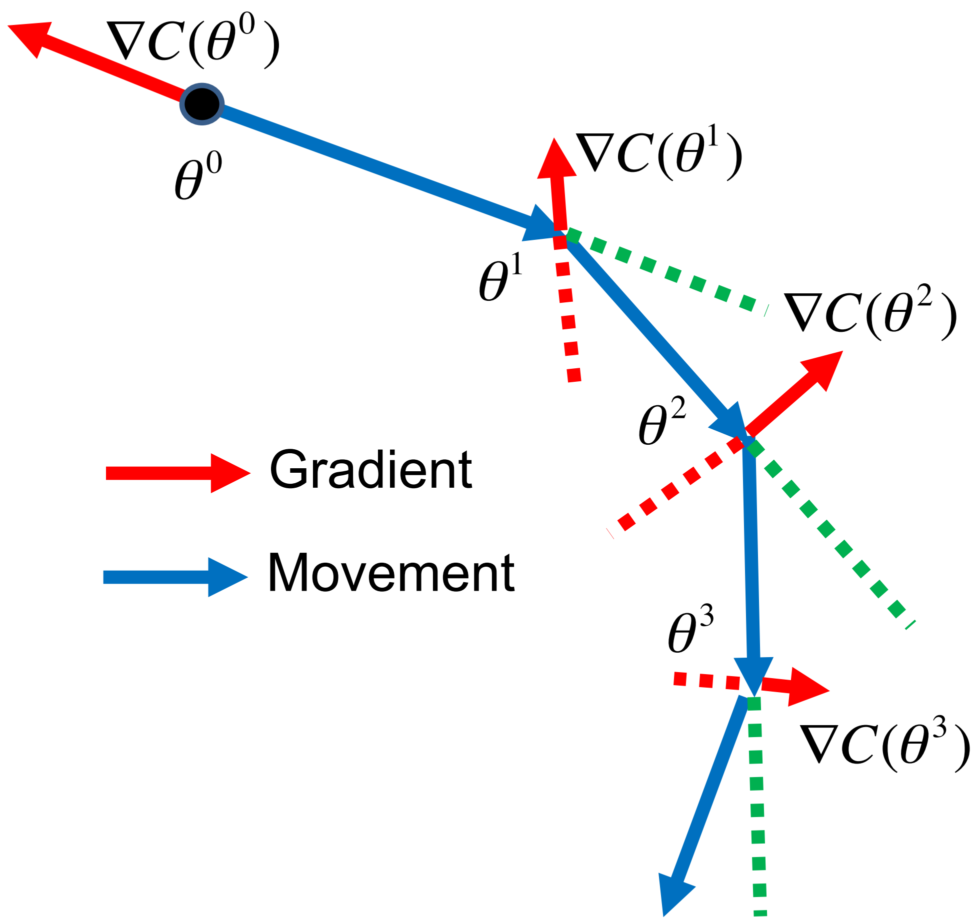
Move to $\theta^1 = \theta^0 - \eta \nabla C(\theta^0)$

Compute gradient at θ^1

Move to $\theta^2 = \theta^1 - \eta \nabla C(\theta^1)$

⋮

Gradient Descent with Momentum



Start at position θ^0

Momentum $v^0 = 0$

Compute gradient at θ^0

Momentum $v^1 = \lambda v^0 - \eta \nabla C(\theta^0)$

Move to $\theta^1 = \theta^0 + v^1$

Compute gradient at θ^1

Momentum $v^2 = \lambda v^1 - \eta \nabla C(\theta^1)$

Move to $\theta^2 = \theta^1 + v^2$

$\vdots$

- v^i is the weighted sum of all the previous gradient $(\nabla C(\theta^0), \nabla C(\theta^1), \cdots, \nabla C(\theta^{i-1}))$

Gradient Descent with Momentum

- A form of accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients

---

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

---

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

Data Partition

- Full training set D is partitioned into M non-overlapping blocks
- Each block is partitioned into N non-overlapping splits

$$\mathbf{D} = \{\mathbf{D}_j | j = 1, 2, \cdots, M\}$$
$$\mathbf{D}_j = \{\mathbf{D}_{jk} | k = 1, 2, \cdots, N\}$$
$$\text{for} \quad \forall j, k, l, m \quad \mathbf{D}_{jk} \cap \mathbf{D}_{lm} = \emptyset$$

Blockwise Model-Update Filtering (BMUF)

- Broadcast a global model $\boldsymbol{W}_g(t-1)$ to each worker
- Each worker computes a gradient for a split. If we simply aggregate the parameters $\overline{\boldsymbol{W}}(t)$ by N-averaging
- However, in the parameter server, instead of directly using $\overline{\boldsymbol{W}}(t)$, the global model is updated as follows.

Blockwise Model-Update Filtering (BMUF)

- Compute $\boldsymbol{G}(t)$ to denote the model-update resulting from block D_t

$$\boldsymbol{G}(t) = \overline{\boldsymbol{W}}(t) - \boldsymbol{W}_g(t-1)$$

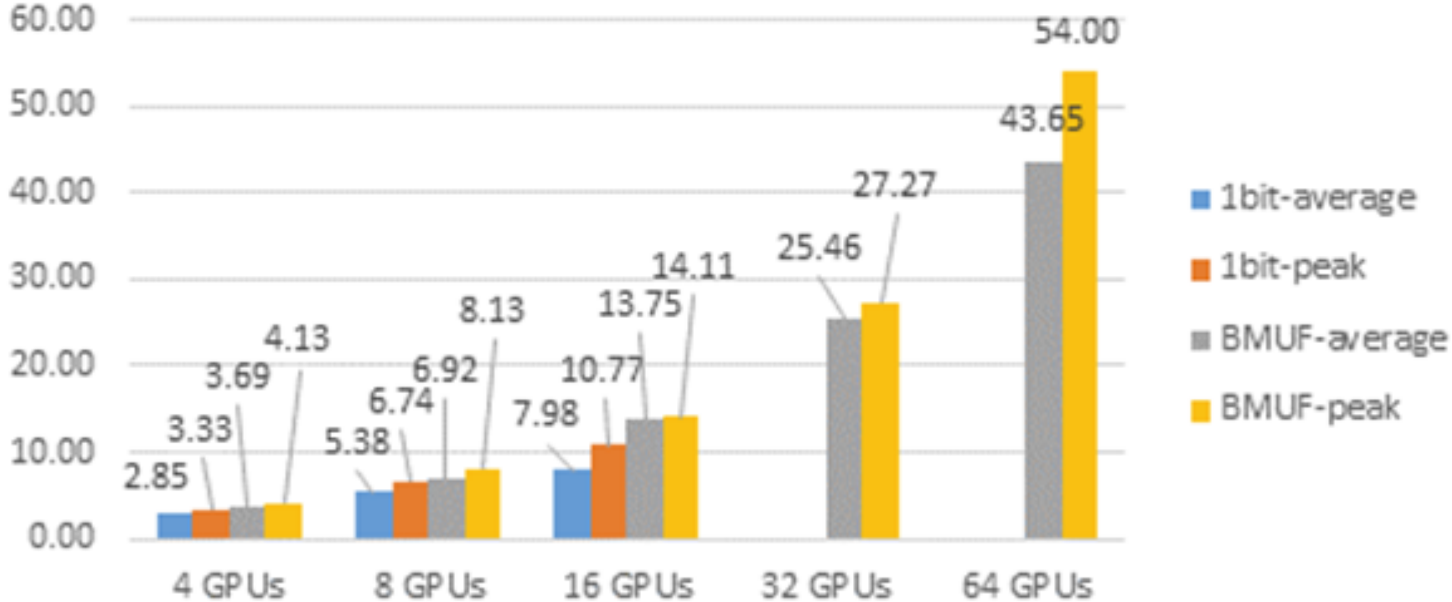
- Then calculate the global-model update $\Delta(t)$

$$\Delta(t) = \eta_t \Delta(t) + \xi_t \boldsymbol{G}(t), \qquad 0 \le \eta_t < 1, \xi_t > 0$$

- Finally, the global model update is

$$\boldsymbol{W}(t) = \boldsymbol{W}(t-1) + \Delta(t)$$

Results



LSTM SGD baseline	11.08				
Parallel Algorithms	4-GPU	8-GPU	16-GPU	32-GPU	64-GPU
1bit	10.79	10.59	11.02		
BMUF	10.82	10.82	10.85	10.92	11.08

Table 2: WERs (%) of parallel training for LSTMs

Frank Seide, "CNTK: Microsoft's Open-Source Deep-Learning Toolkit", Microsoft Research Faculty Summit 2016

Reference

- 1-big SGD
  - F. Seide et al. "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs." INTERSPEECH 2014.
  - N. Strom "Scalable distributed DNN training using commodity GPU cloud computing." INTERSPEECH 2015.
- Block momentum
  - K. Chen and Q. Huo. "Scalable training of deep learning machines by incremental block training with intra-block parallel optimization and blockwise model-update filtering." ICASSP 2016.