PROJECT ADAM:
BUILDING AN EFFICIENT AND SCALABLE DEEP LEARNING TRAINING SYSTEM

Trishul Chilimbi Yutaka Suzue Johnson Apacible
Karthik Kalyanaraman *Microsoft Research*

Presented by:
LU YI 17R50002

OUTLINE

1 Introduction

2 Background

3 ADAM system architecture

4 Evaluation

5 Related work

INTRODUCTION

INTRODUCTION

Traditional statistical machine learning operates with a table of data and a prediction goal.

row ––>independent observation

columns–>hand crafted features of underlying data set

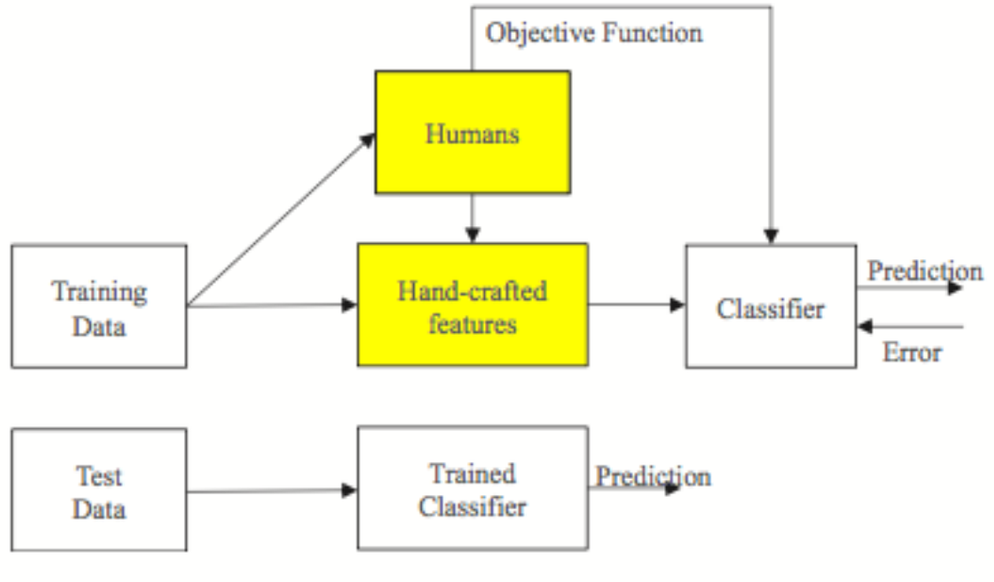


Figure 1. Machine Learning.

Unfortunately

hard AI tasks :speech recognition or visual object classification

Deep learning

additionally learning hierarchical features from the raw input data

using these features to make predictions

deep neural networks(DNN)

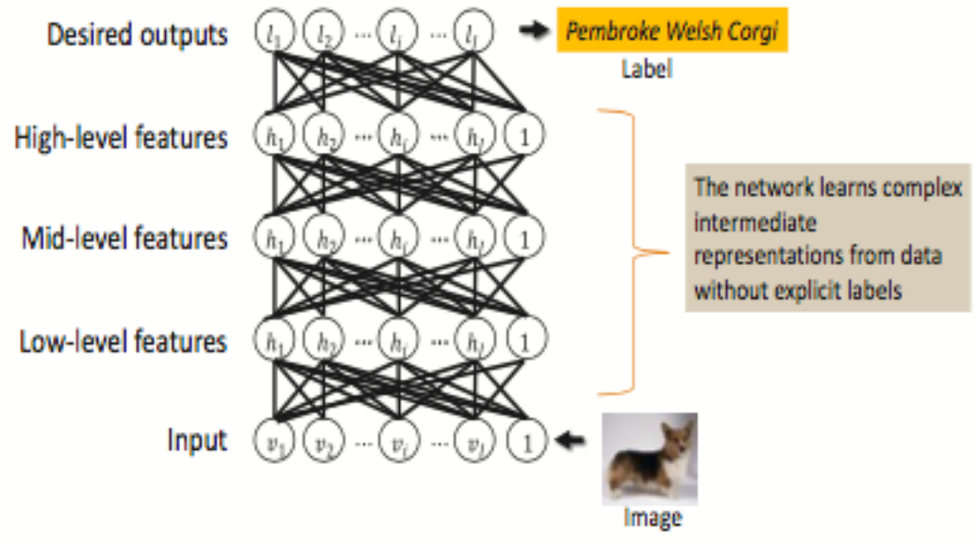


Figure 2. Deep networks learn complex representations.

INTRODUCTION

Deep learning's recent success :

- advances in computing capability for training these models
- the core algorithms and models are mostly unchanged from the eighties and nineties

learning hierarchical features requires significantly more training data and computing power to be successful.

prevent over-fitting ⟶ poor generalization performance on unseen test data

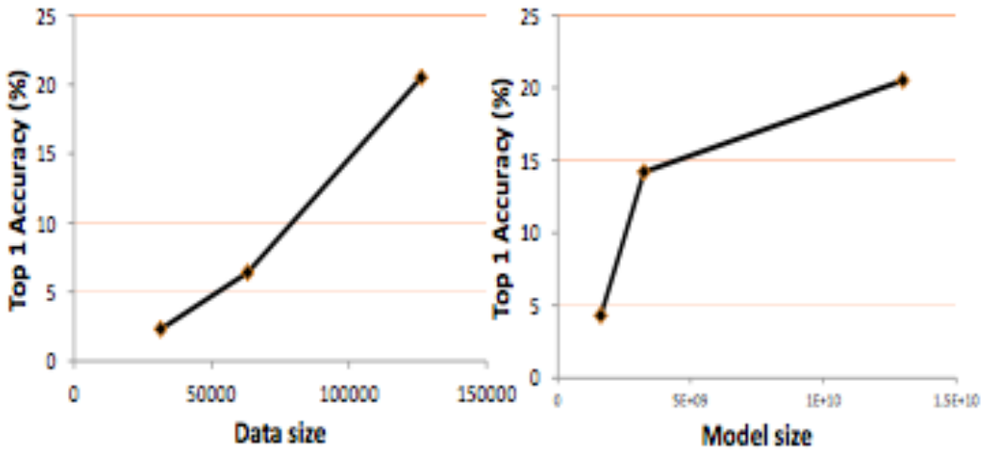


Figure 3. Accuracy improvement with larger models and more data.

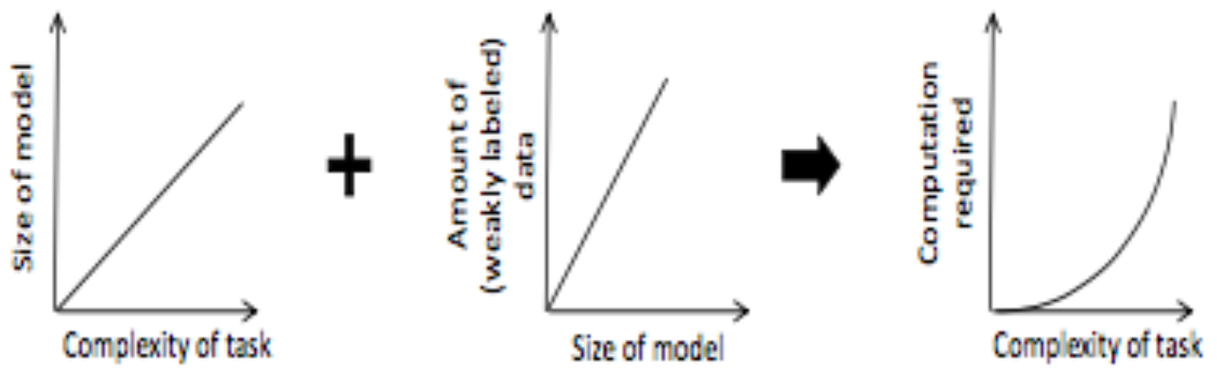


Figure 4. Deep Learning Computational Requirements.

INTRODUCTION

While this works well when the model fits within 2-4 GPU cards attached to a single server, it limits the size of models that can be trained.

researchers recently built

a scalable distributed deep learning training system called Adam comprised of commodity servers

> Commodity computing (also known as commodity cluster computing) involves the use of large numbers of already-available computing components for parallel computing, to get the greatest amount of useful computation at low cost.

INTRODUCTION

main contributions

- Optimizing and balancing both computation and communication for this application through whole system co-design.
  - they partition large models across machines so as to minimize memory bandwidth and cross- machine communication requirements.
  - they restructure the computation across machines to reduce communication requirements.
- Achieving high performance and scalability by exploiting the ability of machine learning training to tolerate inconsistencies well.
  - multi- threaded model parameter updates without locks
  - asynchronous batched parameter updates
- Demonstrating that system efficiency, scaling, and asynchrony all contribute to improvements in trained model accuracy.

BACKGROUND

BACKGROUND

2.1 Deep Neural Networks for Vision

Artificial neural networks —consist of→ neurons

neurons: multiple inputs and a single output

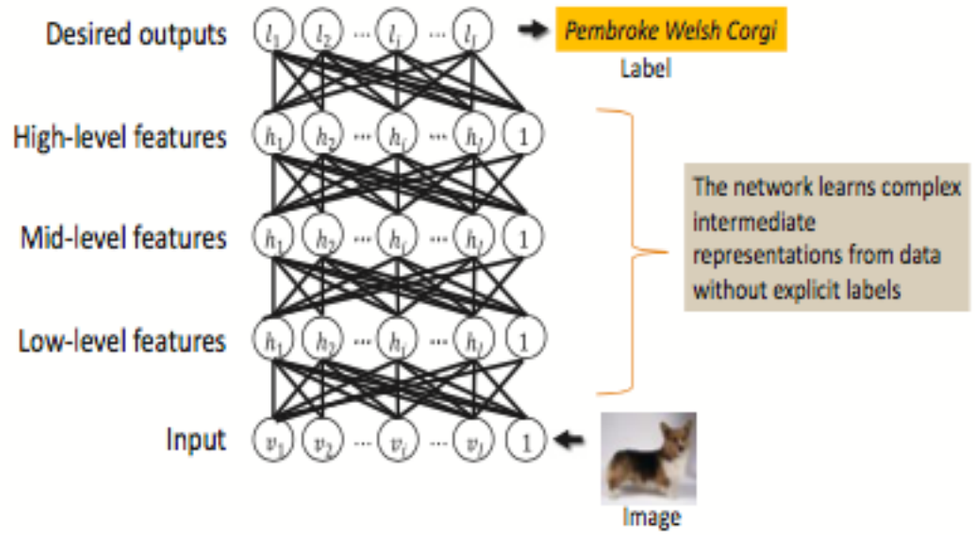


Figure 2. Deep networks learn complex representations.

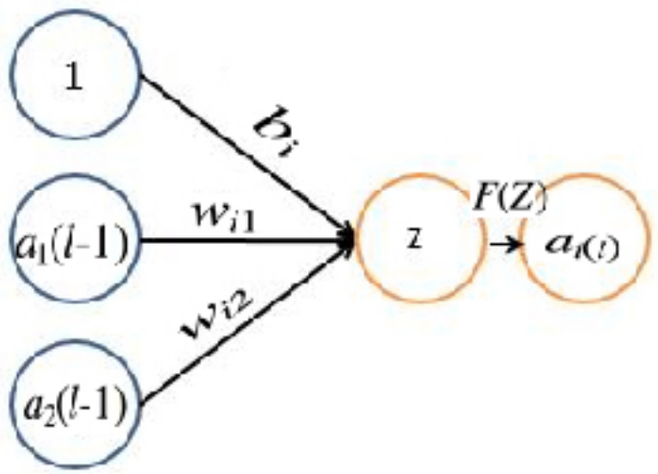
activation : The output of a neuron i in layer l

$$a_i(l) = \mathrm{F}((\Sigma_{j=1..k}\, w_{ij}(l\text{-}1,l)*a_j(l\text{-}1)) + b_i)$$

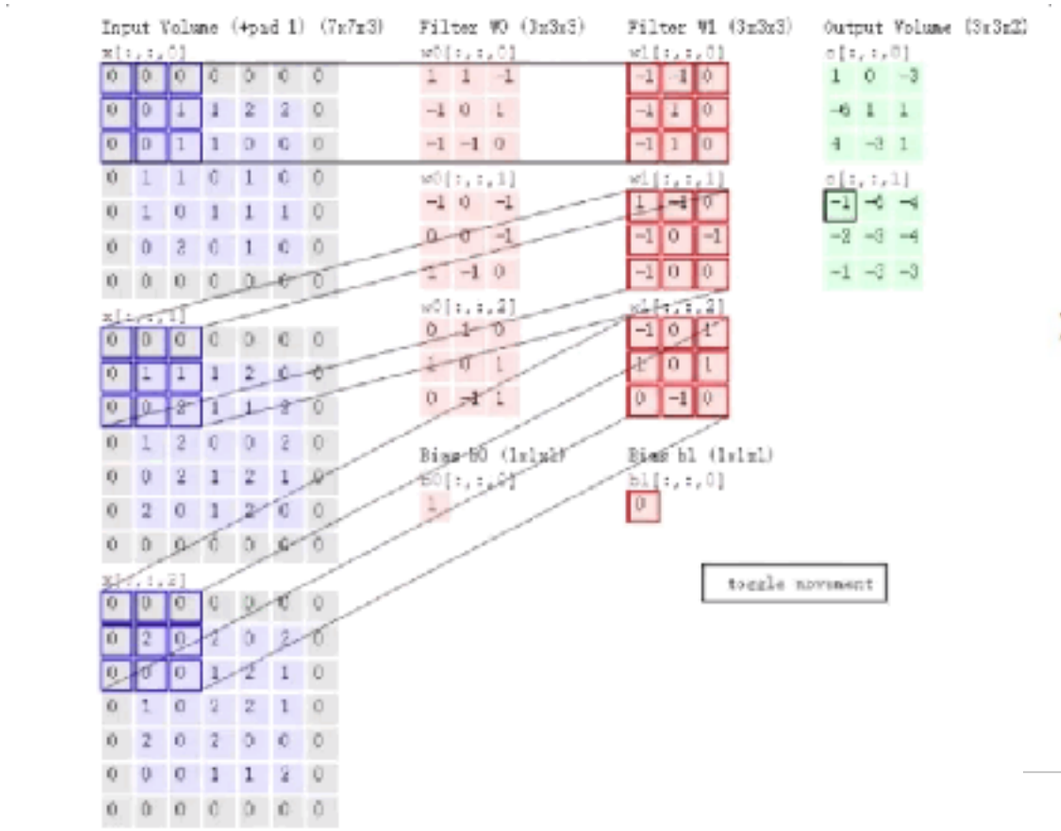
w_{ij} : the weight associated with the connection between neurons i and j

b_i : a bias term associated with neuron i

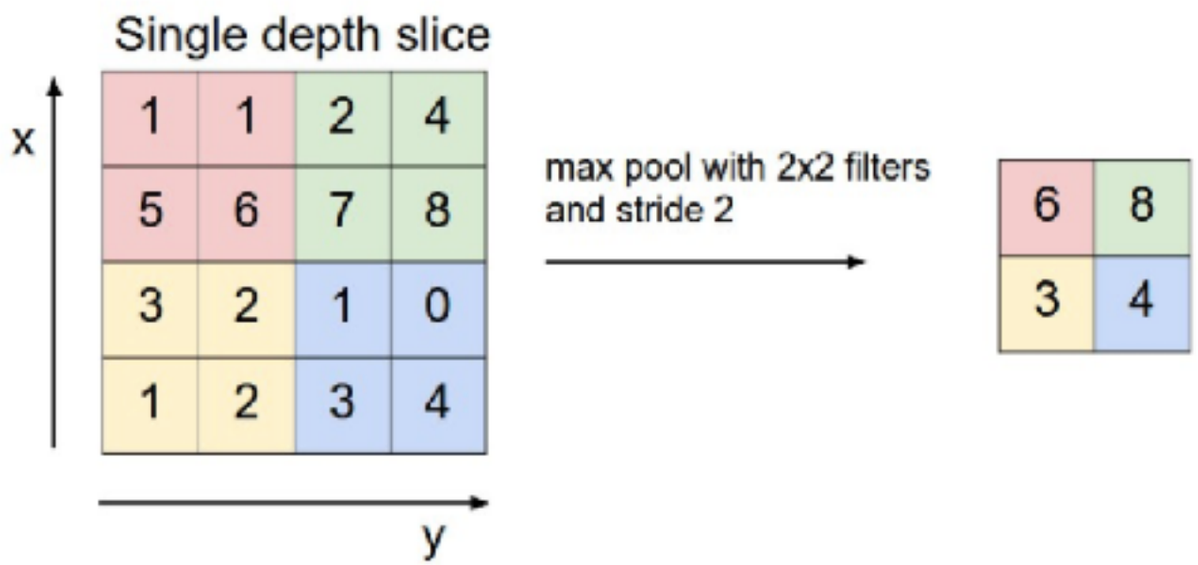
F: associated with all neurons in the network is a pre- defined non-linear function, typically sigmoid or hyperbolic tangent.



Convolutional neural netswork



max-pooling layer



- only connected to spatially local neurons in the next layer
- share weights ⟶ reduces the number of free parameters
- a type of nonlinear down-sampling by outputting the maximum value from non- overlapping sub-regions ⟶ provides the network with robustness to small translations in the input

BACKGROUND

softmax function

The last layer of a neural network

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

the logistic function that "squashes" a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range [0, 1] that add up to 1.

2.1 Neural Network Training

Neural networks are typically trained by back- propagation using gradient descent.

In stochastic gradient descent the training inputs are processed in a random order

The inputs are processed one at a time with the following steps performed for each input to update the model weights.

Feed-forward evaluation:

Back-propagation:

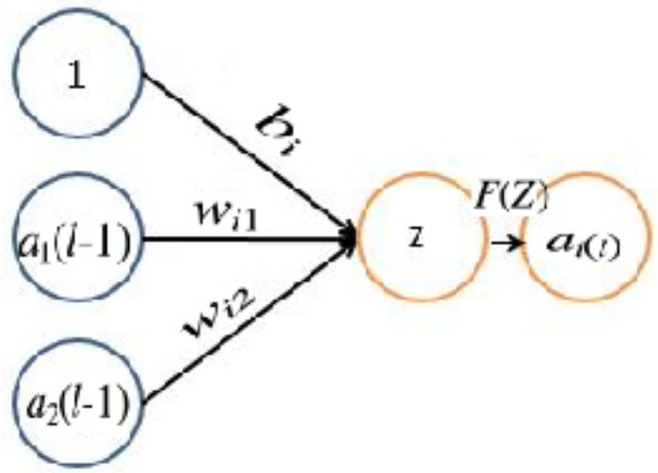
Weight updates:

BACKGROUND

Feed-forward evaluation:

activation : The output of a neuron *i* in layer *l*

$$a_i(l) = \mathrm{F}((\Sigma_{j=1..k}\, w_{ij}(l\text{-}1,l)*a_j(l\text{-}1)) + b_i)$$



Back-propagation:

Error terms, δ, are computed for each neuron, i, in the output layer, l_n, first as follows:

$$\delta_i(l_n) = (t_i(l_n) - \mathrm{a}_i(l_n))*\mathrm{F}'(a_i(l_n))$$

where $t(x)$ is the true value of the output and $F'(x)$ is the derivative of $F(x)$.

These error terms are then back-propagated for each neuron i in layer l connected to m neurons in layer $l+1$ as follows:

$$\delta_i(l) = (\Sigma_{j=1..m}\, \delta_j(l+1)*\mathrm{w}_{ji}(l,l+1))*\mathrm{F}'(\mathrm{a}_i(l))$$

BACKGROUND

Weight updates:

These error terms are used to update the weights (and biases similarly) as follows:

$$\Delta w_{ij}(l\text{-}1,l) = \alpha * \delta_i(l) * a_j(l\text{-}1) \text{ for j} = 1 \text{ .. k}$$

α is the learning rate parameter

repeated for each input until the entire training datasethas been processed, which constitutes a training epoch.

At the end of a training epoch, the model prediction error is computed on a held out validation set. Typically, training continues for multiple epochs, reprocessing the training data set each time, until the validation set error converges to a desired (low) value.

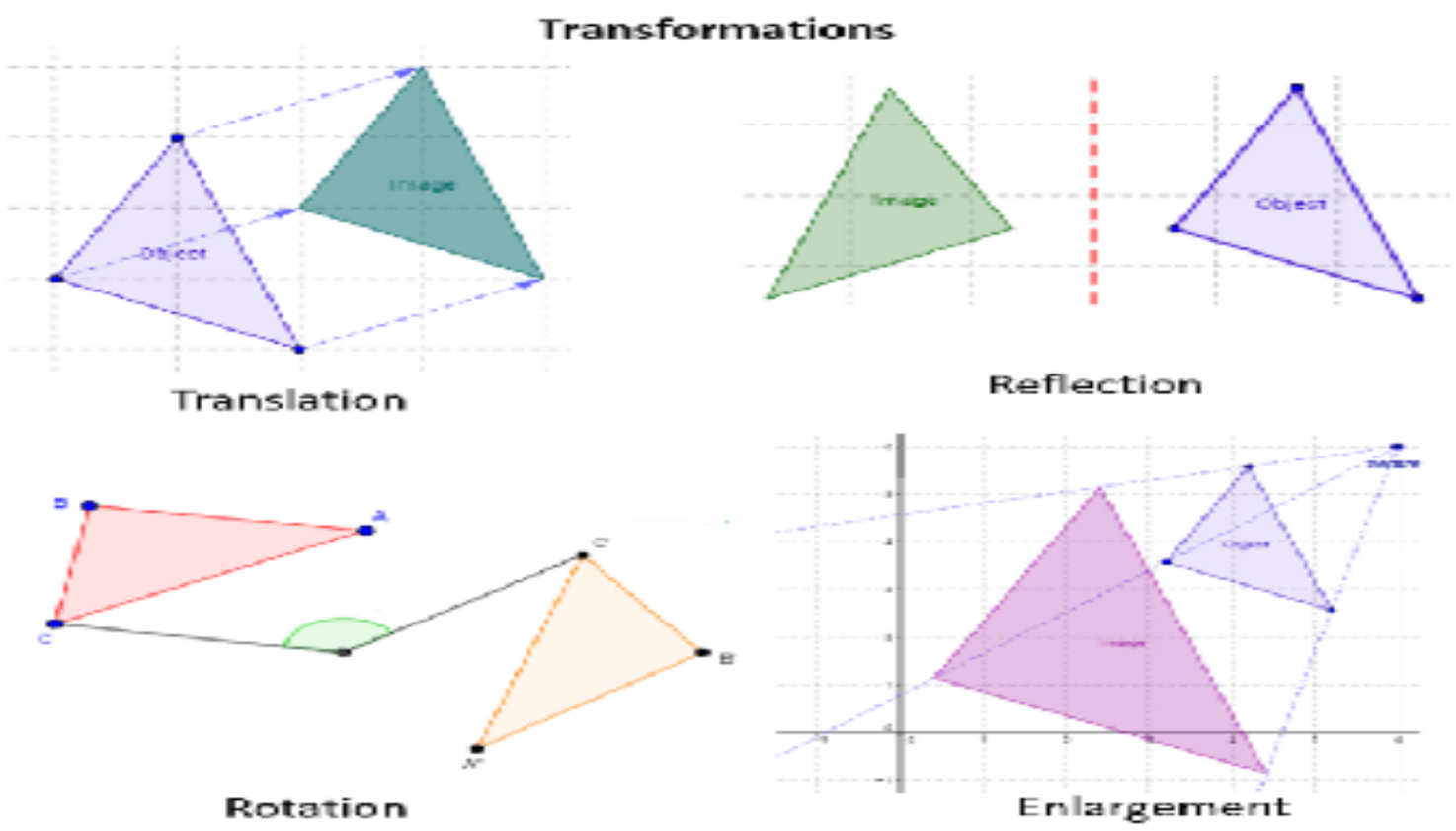
ADAM SYSTEM ARCHITECTURE

3.1 Fast Data Serving

Training large DNNs requires vast quantities of training data (10-100 TBs).

data transformations ⟶ avoid over-fitting

data serving machines ⟶ offload the computational requirements ⟶ ensure high throughput data delivery



- augmented by randomly
- in advance
- utilizing the entire system memory as a image cache
- use asynchronous IO
- request images in advance in batches using a background thread

3.2 Model Training

3.2.1 Multi-Threaded Training

3.2.2 Fast Weight Updates

3.2.3 Reducing Memory Copies

3.2.4 Memory System Optimizations

3.2.5 Mitigating the Impact of Slow Machines

3.2.6 Parameter Server Communication

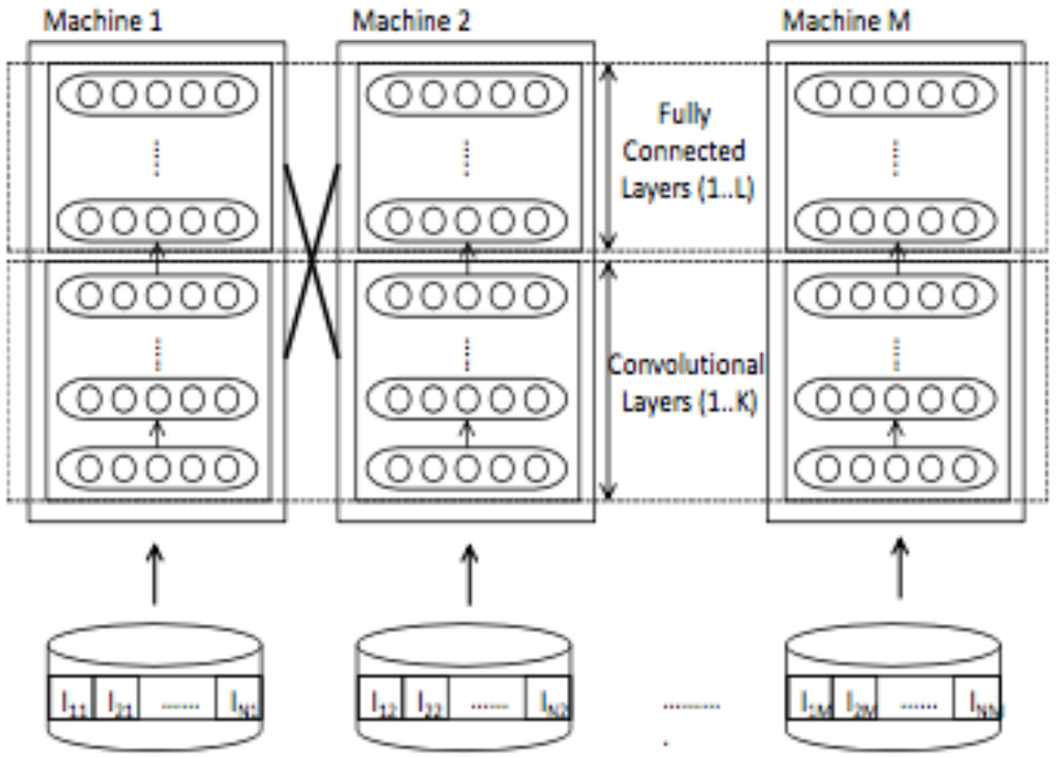


Figure 6. Model partitioning across training machines.

3.2.1 Multi-Threaded Training

- different images $\xrightarrow{\text{assigned to}}$ threads that share the model weights
- each thread $\xrightarrow{\text{allocate}}$ training context

context: the activations and weight update values computed during back-propagation for each layer

- The context is pre-allocated to avoid heap locks while training

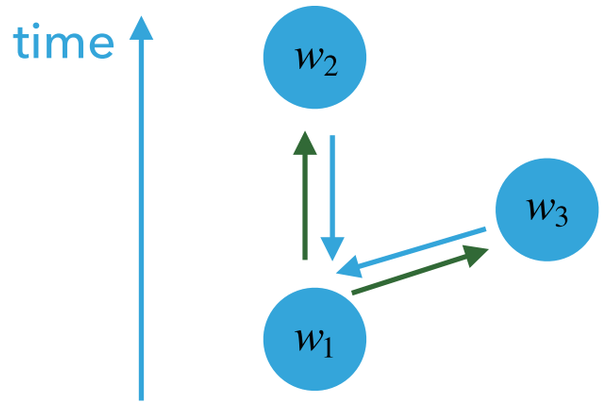
- Both the context and per-thread scratch buffer for intermediate results use NUMA- aware allocations to reduce cross-memory bus traffic as these structures are frequently accessed.

3.2.2 Fast Weight Updates

access and update the shared model weights locally without using locks

↓

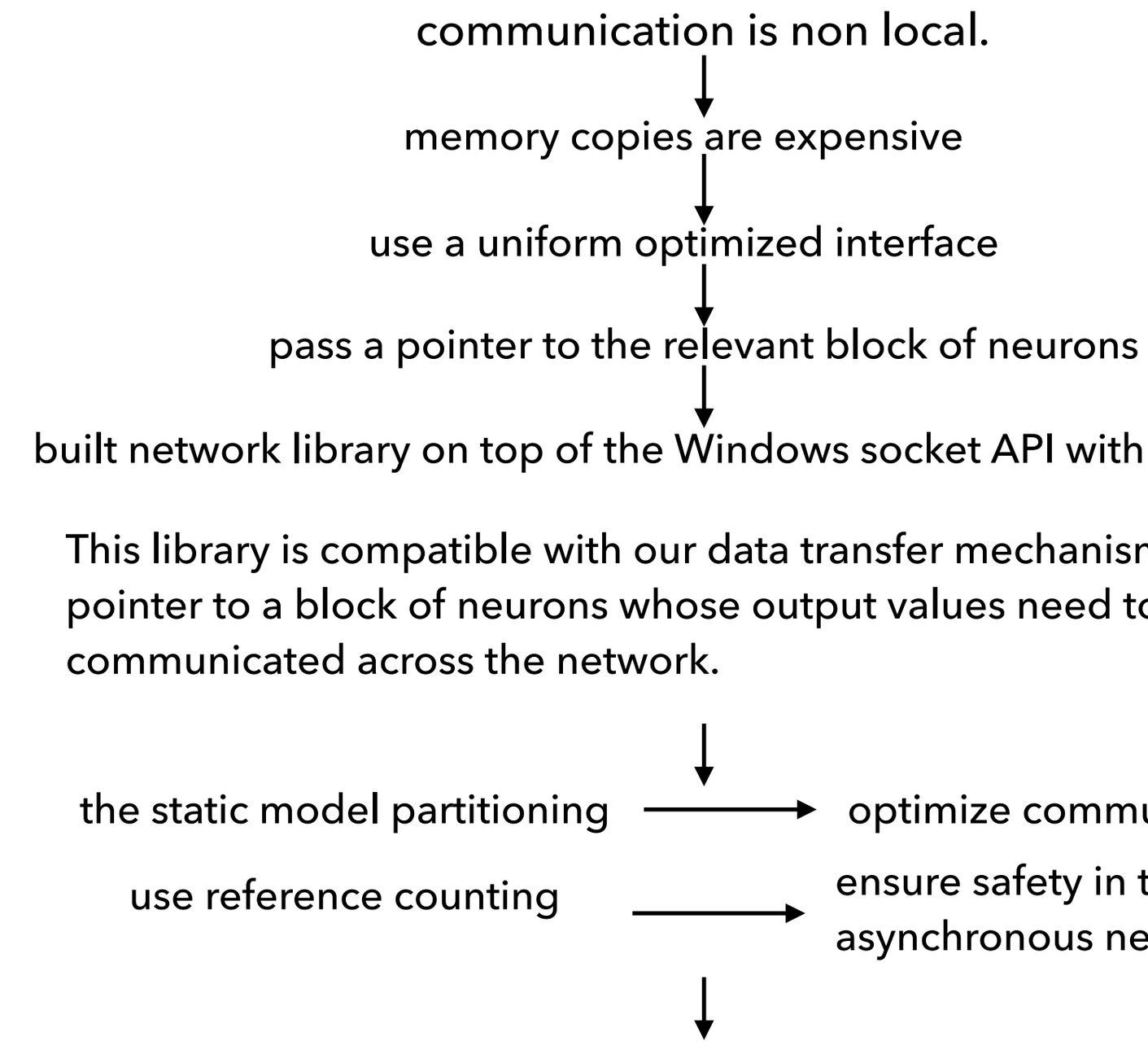
some races and modifying weights based on stale weight values



↓

the weight updates are associative and commutative and because neural networks are resilient and can overcome the small amount of noise that this introduces

3.2.3 Reducing Memory Copies



3.2.4 Memory System Optimizations

- partition models across multiple machines such that the working sets for the model layers fit in the L3 cache.

- optimize computation for cache locality

  The forward evaluation and back-propagation computation have competing locality requirements in terms of preferring a row major or column major layout for the layer weight matrix

  created two custom hand-tuned assembly kernels that appropriately pack and block the data such that the vector units are fully utilized for the matrix multiply operations

3.2.5 Mitigating the Impact of Slow Machines

- To avoid stalling threads on faster machines that are waiting for data values to arrive from slower machines, we allow threads to process multiple images in parallel

  We use a dataflow framework to trigger progress on individual images based on arrival of data from remote machines.

- because we need to wait for all training images to be processed to compute the model prediction error on the validation data set and determine whether an additional training epoch is necessary

  We have empirically determined that waiting for 75% of the model replicas to complete processing all their images before declaring the training epoch complete can speed training by up to 20% with no impact on the trained model's prediction accuracy.

3.2.6 Parameter Server Communication

two different communication protocols for updating parameter weights

For the convolutional layers

locally computes and accumulates the weight updates in a buffer that is periodically sent to the parameter server machines when k (which is typically in the hundreds) images have been processed. The parameter server machines then directly apply these accumulated updates to the stored weights.

For the fully connected layers

Rather than directly send the weight updates we send the activation and error gradient vectors to the parameter server machines where the matrix multiply can be performed locally to compute and apply the weight updates.

3.3 Global Parameter Server

The parameter server is in constant communication with the model training machines receiving updates to model parameters and sending the current weight values. The rate of updates is far too high for the parameter server to be modeled as a conventional distributed key value store.

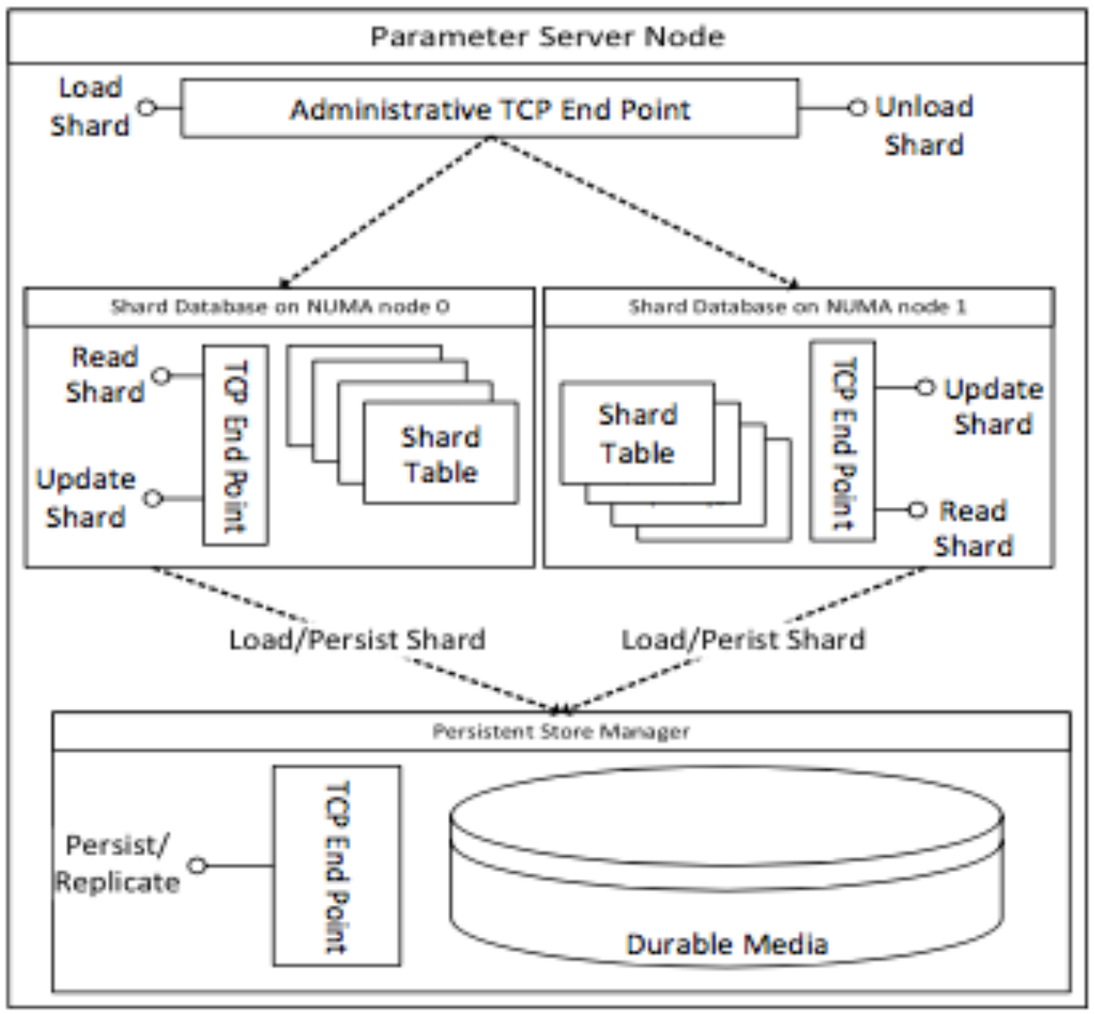


Figure 7. Parameter Server Node Architecture.

3.3.1 Throughput Optimizations

The model parameters

divided into → 1 MB sized shards → hashed into → storage buckets

divided into → 1 MB sized shards → hashed into → storage buckets

(storage buckets are distributed equally among the parameter server machines)

This improves temporal locality and relieves pressure on the L3 cache by applying all updates in a batch to a block of parameters before moving to next block in the shard.

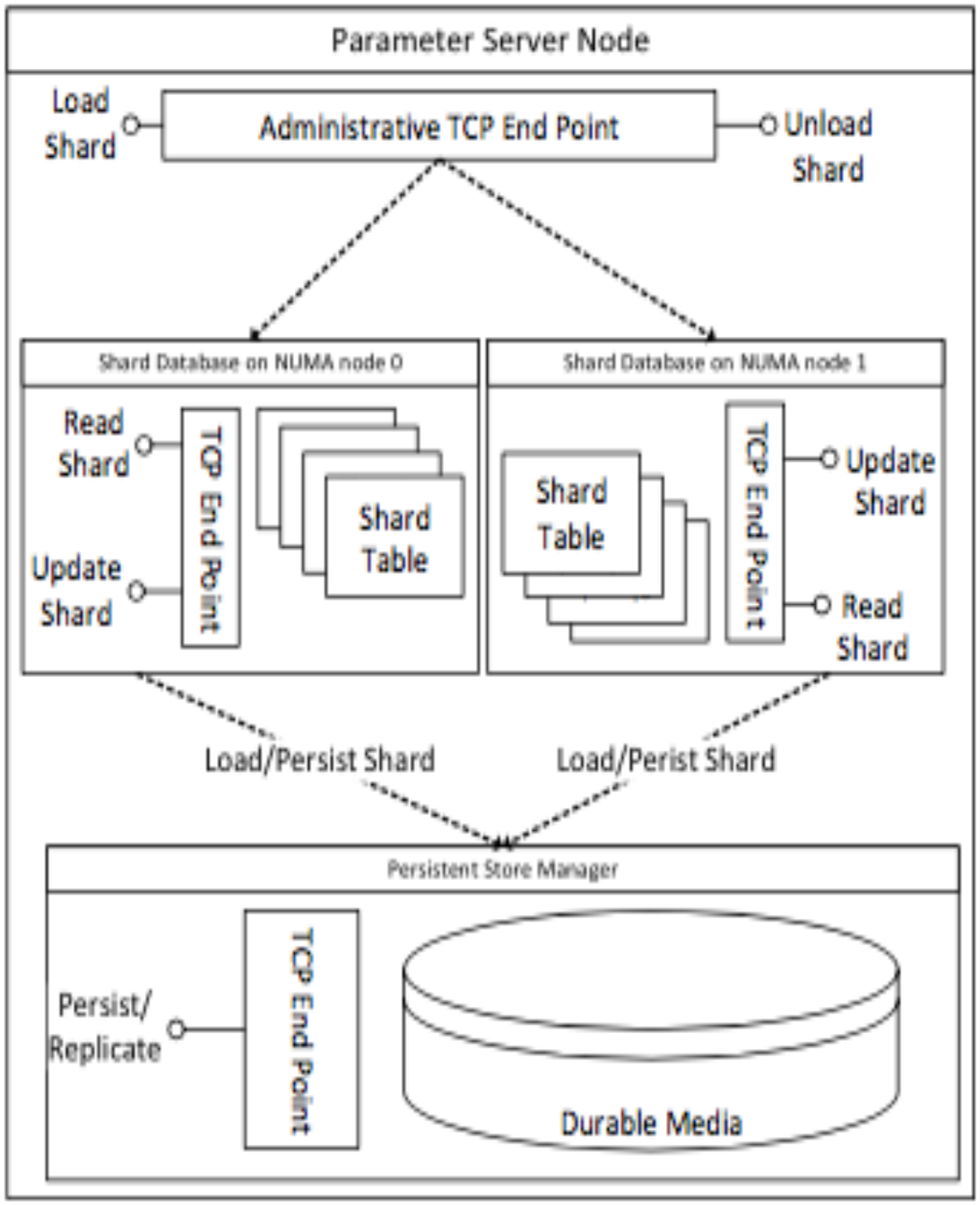


Figure 7. Parameter Server Node Architecture.

ADAM SYSTEM ARCHITECTURE

- Shards are allocated on a specific NUMA node and all update processing for the shard is localized to that NUMA node by assigning tasks to threads bound to the processors for the NUMA node by setting the appropriate processor masks.

- **lock free data structures**

  queues and hash tables in high traffic execution paths to speed up network, update, and disk IO processing

- **lock free memory allocation** where buffers are allocated from pools of specified size that vary in powers of 2 from 4KB all the way to 32MB. Small object allocations are satisfied by our global lock free pool for the object.

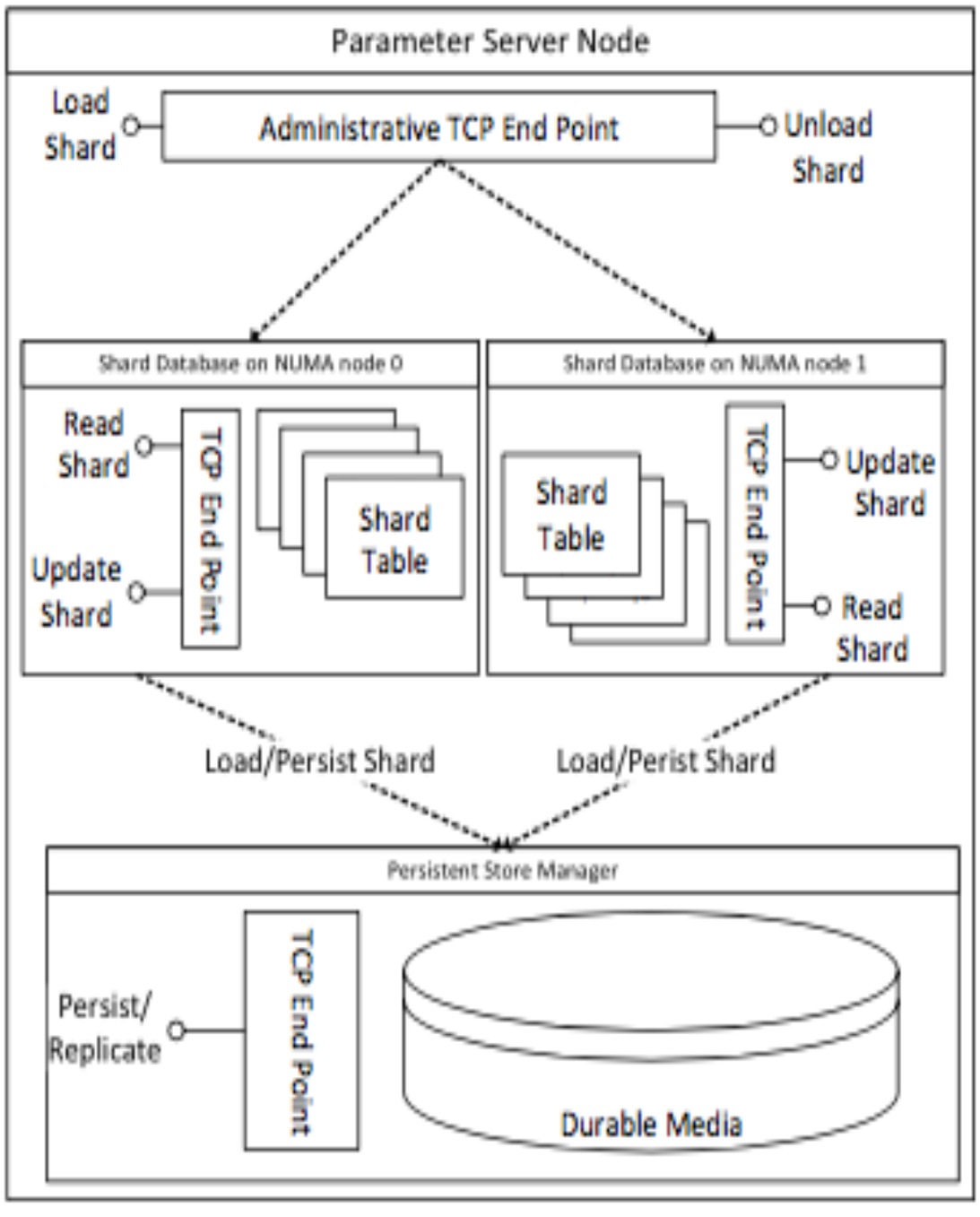


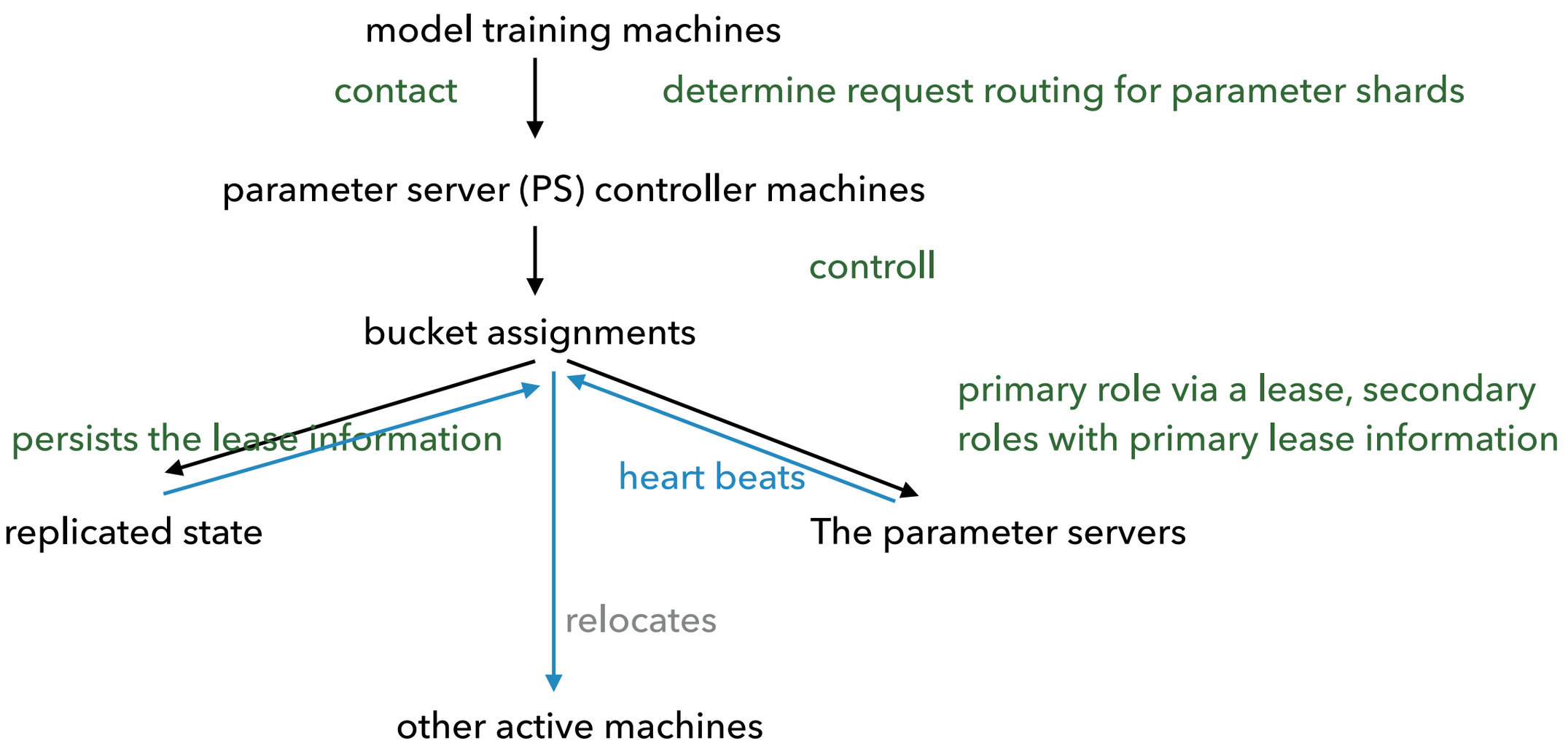
Figure 7. Parameter Server Node Architecture.

3.3.3 Fault Tolerant Operation

three copies of each parameter shard in the system

primary version ⟶ actively served

two other copies ⟶ fault tolerance



3.3.4 Communication Isolation

Parameter server machines have two 10Gb NICs. Since parameter update processing from a client (training) perspective is decoupled from persistence, the 2 paths are isolated into their own NICs to maximize network bandwidth and minimize interference as shown in Figure 7. In addition, we isolate administrative traffic from the controller to the 1Gb NIC.

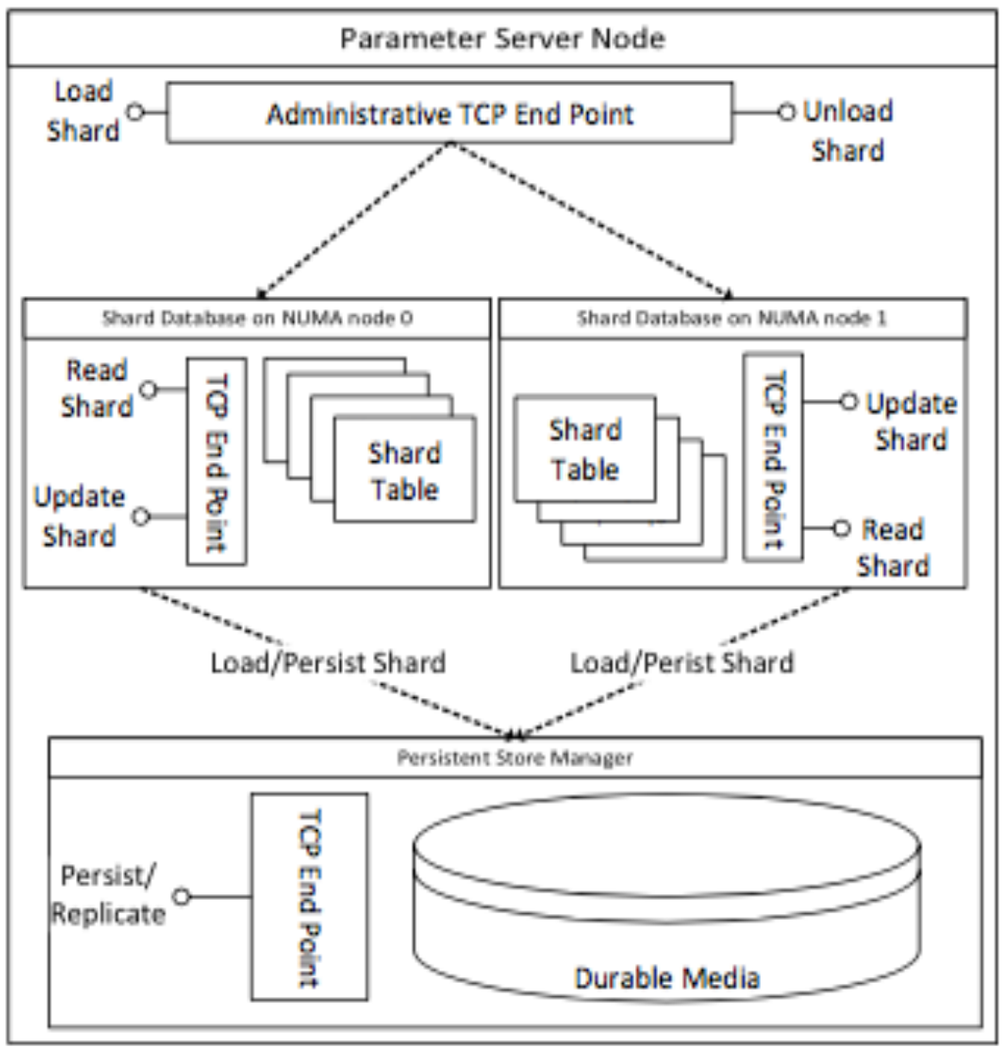


Figure 7. Parameter Server Node Architecture.

EVALUATION

EVALUATION

4.1 Visual Object Recognition Tasks

two popular benchmarks for image recognition tasks

MNIST :

a digit classification task where the input data is composed of 28x28 images of the 10 handwritten digits. This is a very small benchmark with 60,000 training images and 10,000 test images that we use to characterize the baseline system performance and accuracy of trained models.

ImageNet :

a large dataset that contains over 15 million labeled high-resolution images belonging to around 22,000 different categories. The images were gathered from a variety of sources on the web and labeled by humans using Mechanical Turk.

We use this benchmark to characterize Adam's performance and scaling, and the accuracy of trained models.

4.2 SystemHardware

Adam is currently comprised of a cluster of 120 identical machines organized as three equally sized racks connected by IBM G8264 switches. Each machine is a HP Proliant server with dual Intel Xeon E5-2450L processors for a total of 16 cores running at 1.8Ghz with 98GB of main memory, two 10 Gb NICs and one 1 Gb NIC. All machines have four 7200 rpm HDDs. A 1TB drive hosts the operating system (Windows 2012 server) and the other three HDDs are 3TB each and are configured as a RAID array. This set of machines can be configured slightly differently based on the experiment but model training machines are selected from a pool of 90 machines, parameter servers from a pool of 20 machines and image servers from a pool of 10 machines. These pools include standby machines for fault tolerance in case of machine failure.

4.3 Baseline Performance and Accuracy

We first evaluate Adam's baseline performance by focusing on single model training and parameter server machines. In addition, we evaluate baseline training accuracy by training a small model on the MNIST digit classification task.

4.3.1 Model Training System

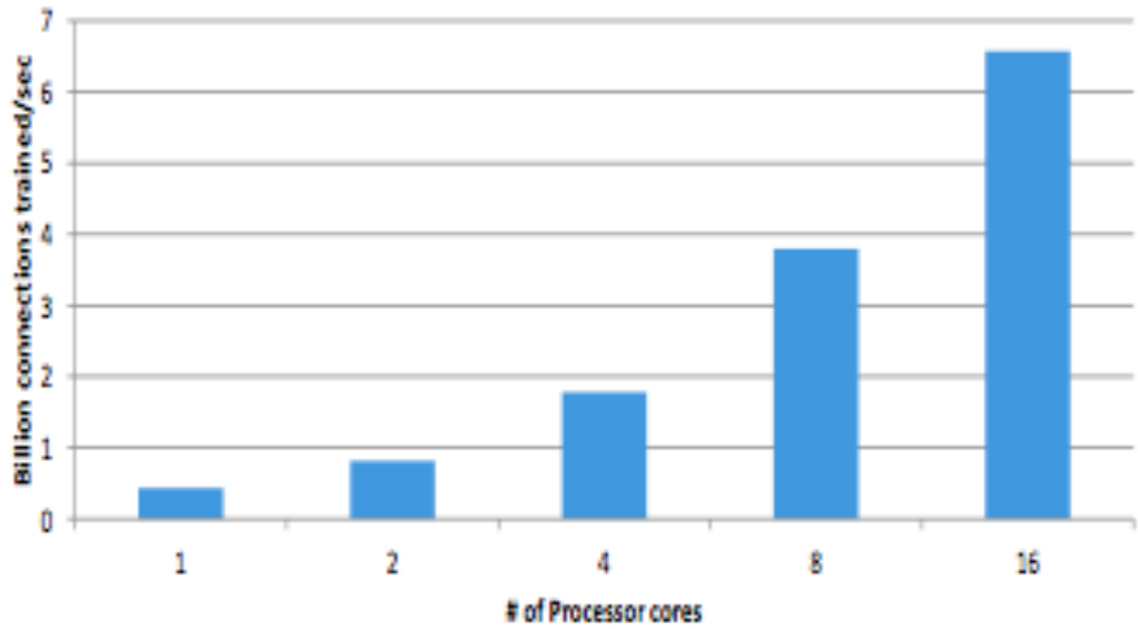


Figure 8. Model Training Node Performance.

Adam shows excellent scaling as we increase the number of cores since we allow parameters to be updated without locking. The scaling is super-linear up to 4 cores due to caching effects and linear afterwards.

4.3.2 Parameter Server

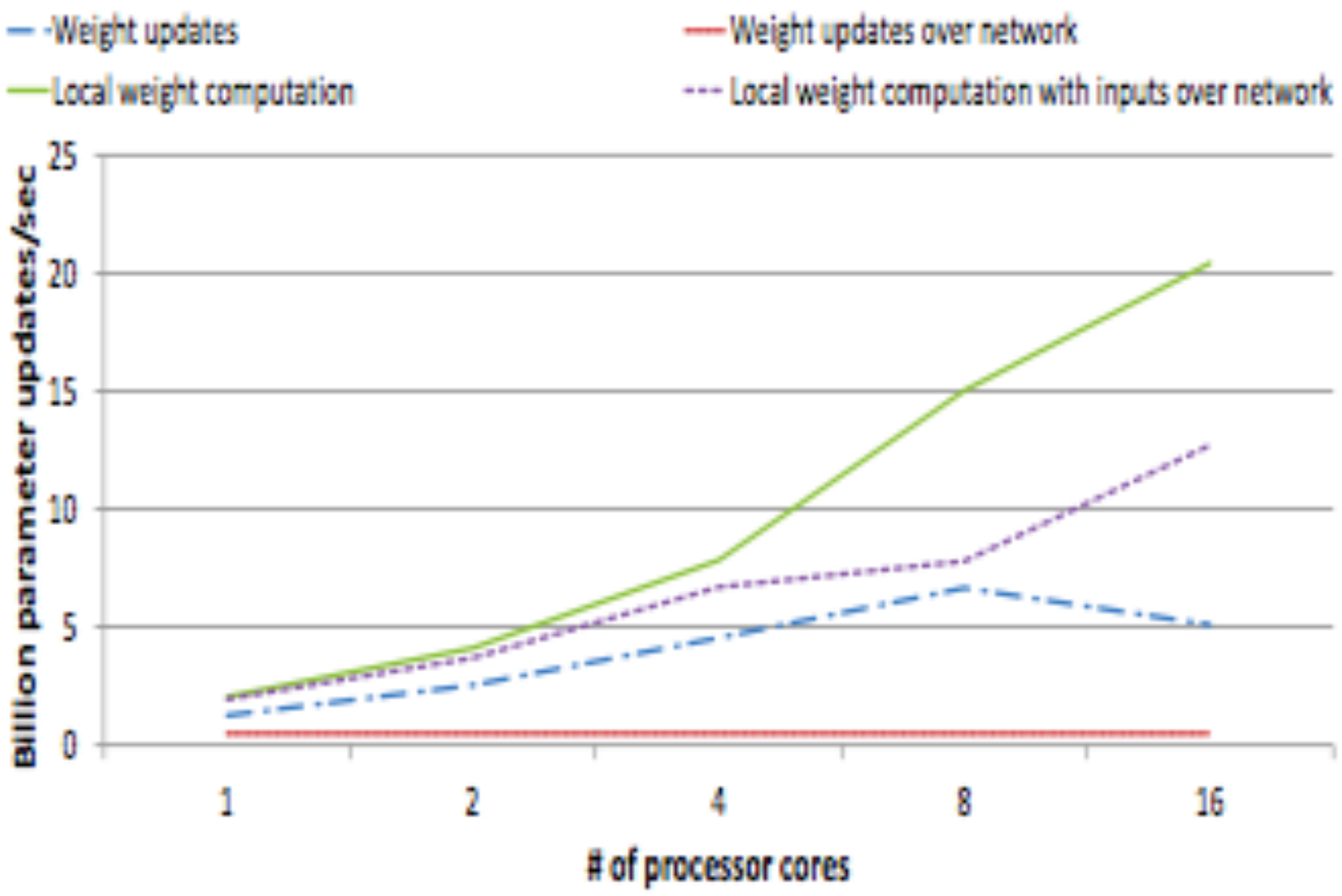


Figure 9. Parameter Server Node Performance.

4.3.3 Trained Model Accuracy

a fairly standard model

2 convolutional layers (5x5 kernels and each is followed by a 2x2 max-pooling layer.The first convolutional layer has 10 feature maps and the second has 20.)

2 fully connected layers (Both fully connected layers use 400 hidden units.)

a final ten class softmax output layer.

The resulting model is small and has around 2.5 million connections.

Table 1. MNIST Top-1 Accuracy

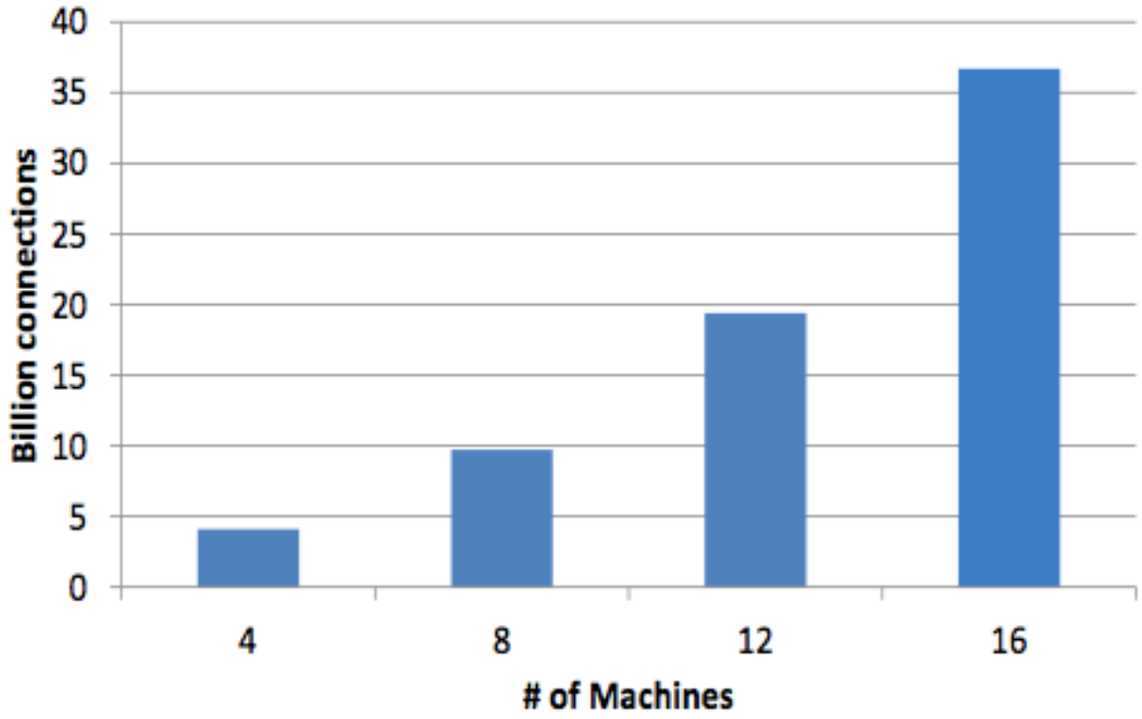
Systems	MNIST Top-1 Accuracy
Goodfellow et al [12]	99.55%
Adam	**99.63%**
Adam (synchronous)	99.39%

We believe that our accuracy improvement arises from the **asynchrony** in Adam which adds a form of stochastic noise while training that helps the models generalize better when presented with unseen data. In addition, it is possible that the asynchrony helps the model escape from unstable local minima to potentially find a better local minimum.

4.4 System Scaling and Accuracy

We evaluate our system performance and scalability across multiple dimensions and evaluate its ability to train large DNNs for the ImageNet 22K classification task.

4.4.1 Scaling with Model Workers



Our 16 machine configuration is capable of training a 36 Bn connection model. More importantly, the size of models we can train efficiently increases super-linearly as we partition the model across more machines.

4.4.2 Scaling with Model Replicas

we evaluated configurations comprising 4, 10, 12, 16, and 22 replicas. All experiments used the same parameter server configuration comprised of 20 machines.

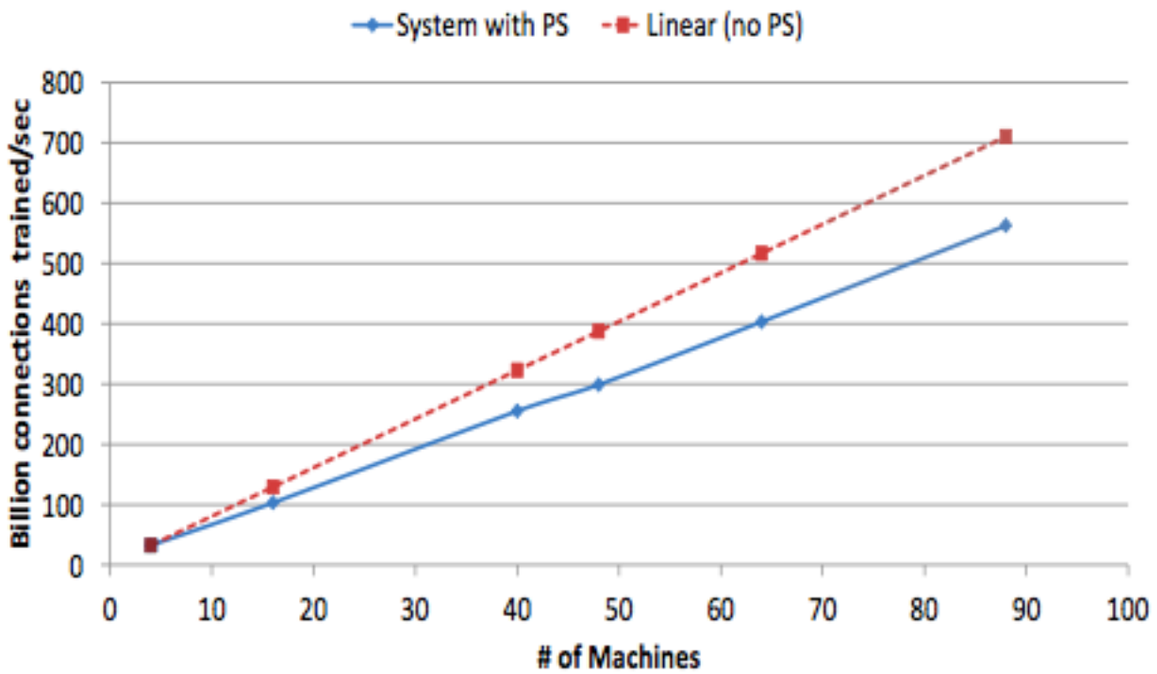


Figure 11. System scaling with more Replicas.

The results indicate that Adam scales well with additional replicas.

4.4.3 Trained Model Accuracy

Table 2. ImageNet 22K Top-1 Accuracy.

Systems	ImageNet 22K Top-1 Accuracy
Le et al. [18]	13.6%
Le et al. (with pre-training) [18]	15.8%
Adam	**29.8%**

previous best top-1 result

supplemented the ImageNet training data with 10 million unlabeled images sampled from Youtube videos

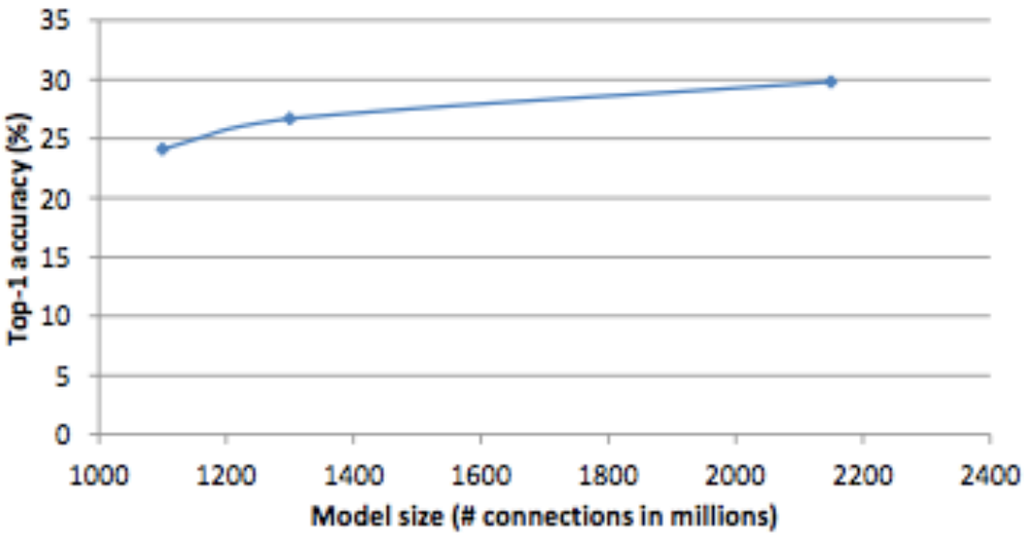


Figure 12. Model accuracy with larger models.

To better understand the reasons for this accuracy improvement, we used Adam to train a couple of smaller models to convergence for this task.

training larger models increases task accuracy

4.4.4 Discussion

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 that take advantage of these updates being associative and commutative. Finally, Adam enables training models to high accuracy by exploiting its efficiency to train very large models and leveraging asynchrony to further improve accuracy.

CONCLUSION

CONCLUSION

We show that large-scale commodity distributed systems can be used to efficiently train very large DNNs to world-record accuracy on hard vision tasks using current training algorithms by using Adam to train a large DNN model that achieves world-record classification performance on the ImageNet 22K category task. While we have implemented and evaluated Adam using a 120 machine cluster, the scaling results indicate that much larger systems can likely be effectively utilized for training large DNNs.

Thank you!

Presented by:
LU YI 17R50002