Performance Modeling and Scalability Optimization of Distributed Deep Learning Systems

Feng Yan
College of William and Mary Williamsburg, VA, USA

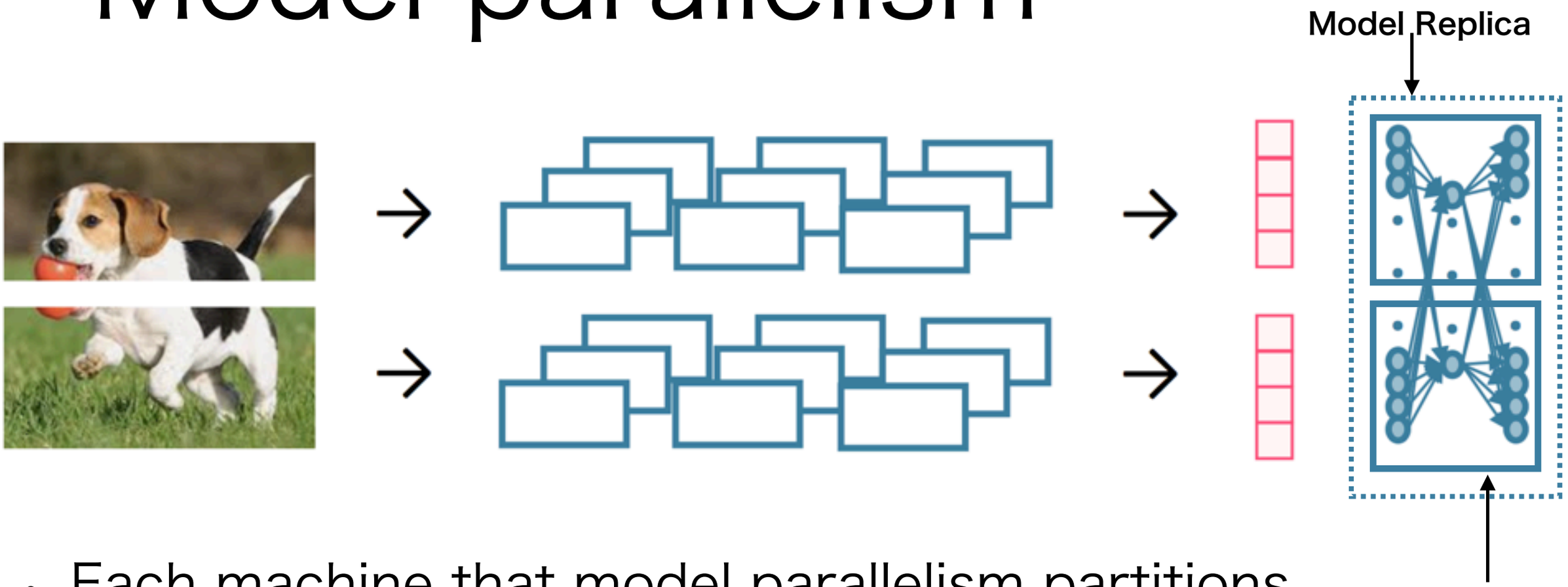
Olatunji Ruwase, Yuxiong He, Trishul Chilimbi
Microsoft Research Redmond,WA, USA

KDD '15 Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

Background

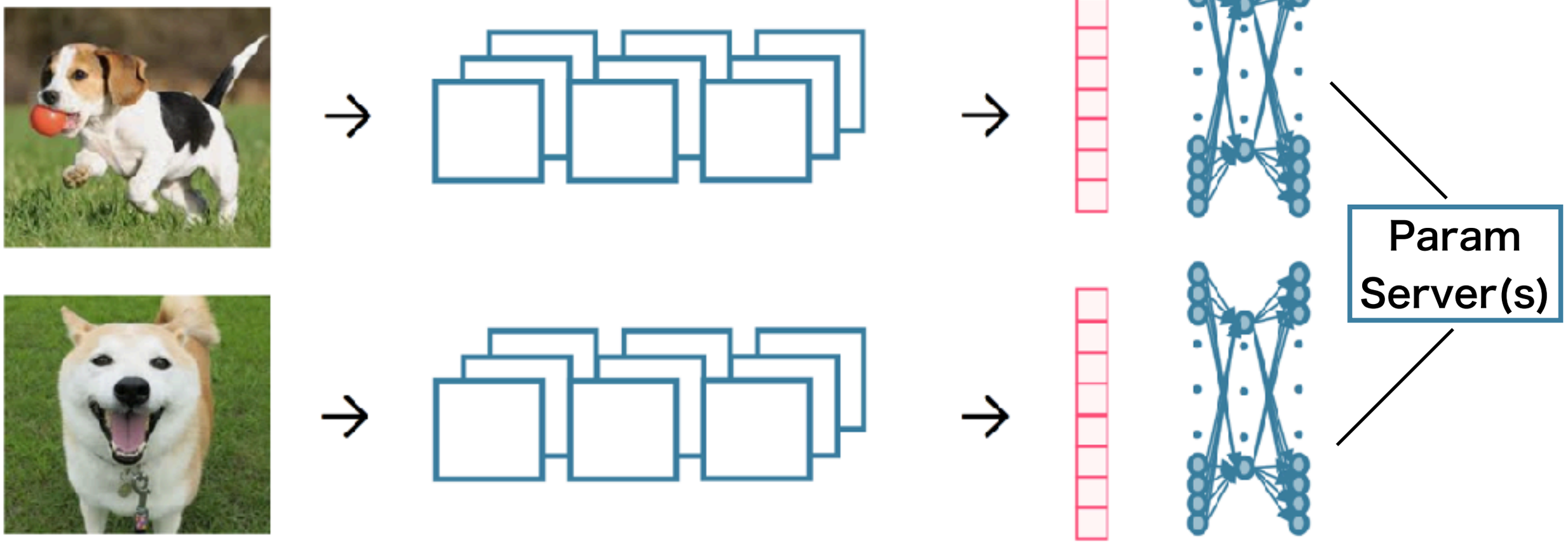
- In order to improving the efficiency of training large DNNs, researchers exploited distributed deep learning systems over clusters of machines in several ways:

Model parallelism



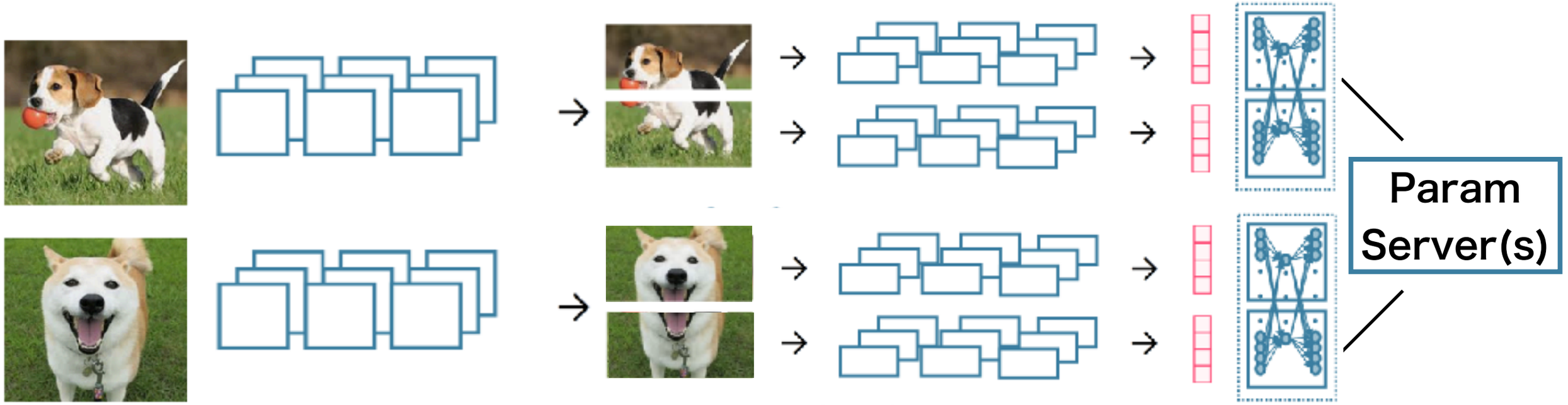
- Each machine that model parallelism partitions called a worker.
- A collection of workers that make up a complete DNN model called a model replica.

Data parallelism



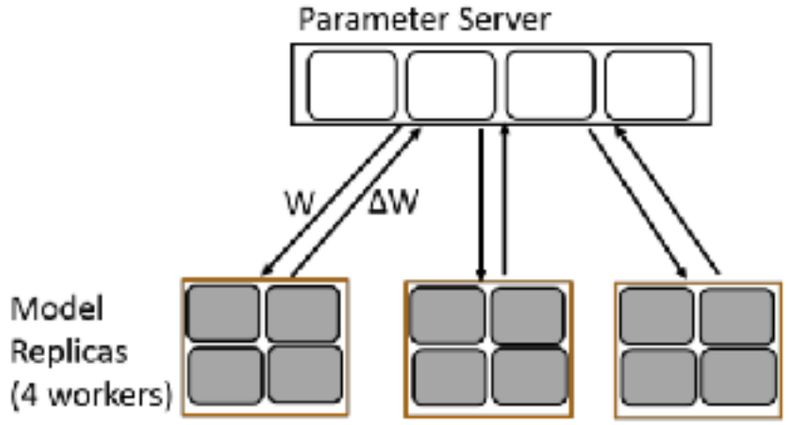
- Every Machine share parameter server to ensure convergence.
- Parameter server can also be partitioned if needed

Combination



- This framework enables to train a large amount of data into reasonable accuracy in a matter of time.

Problem

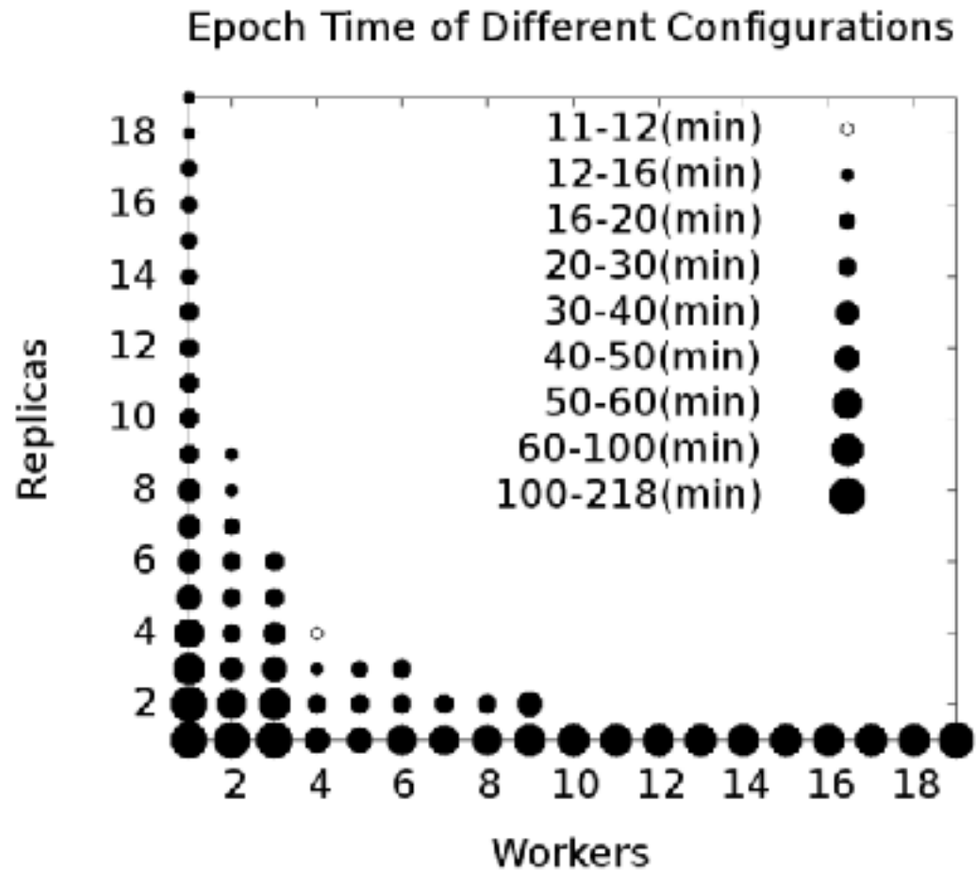


- Since the cluster of machines becomes larger, it becomes a challenging problem to find a proper way to configure the distributed system.
- System configuration includes the number of model replicas, number of works each replica has, number of the parameter servers, and so on…

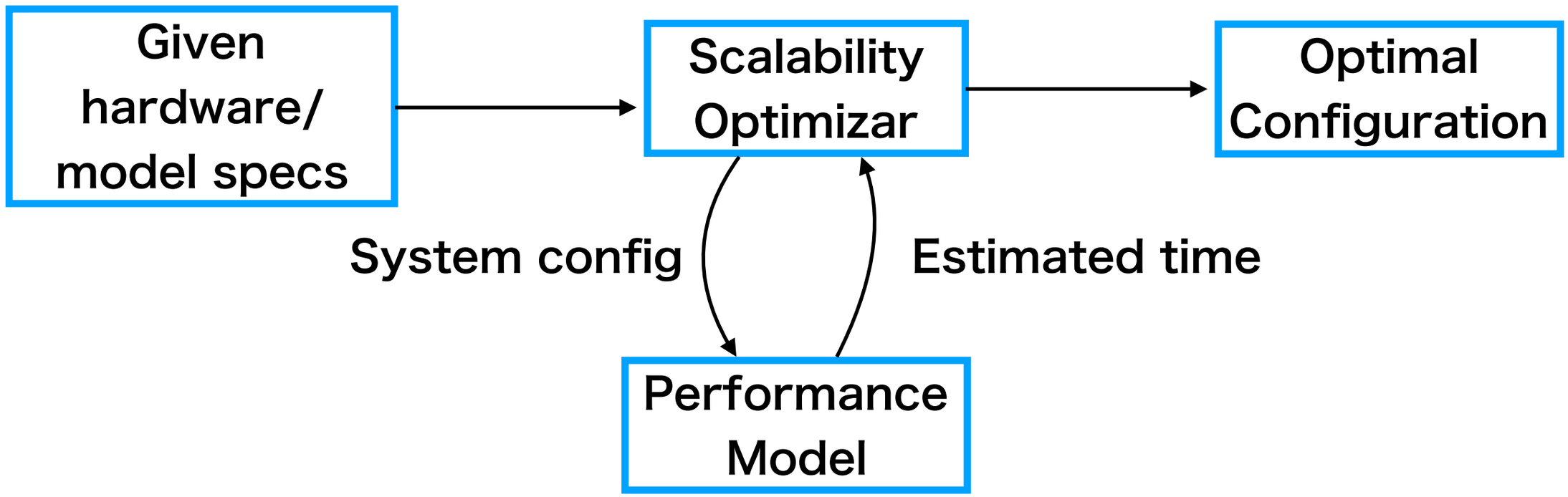
Benefits and challenges of finding an optimal configuration

- Benefits
  - 10x faster than the worst one
  - Noticeably faster than median one
- Challenges
  - A large configuration space that makes enumeration very time and resource consuming
  - One optimal configuration may be specific to only one kinds of model/hardware combination

ImageNet-22k
Processing on 20 machines



The way to solve



- Performance Model: estimates training epoch time for a given system configuration
- Scalability Optimizar: explores the configuration space efficiently and figure out the optimal one according to the estimation given by performance model

Performance Model — Model Parallelism

- The input of DNN is typically processed in three steps:
  - Feedforward evaluation

$$a_i = f\left(\left(\sum_{j=1}^{J} w_{ij} \times a_j\right) + b_i\right)$$

  - Back-propagation
    - Error terms of the output layer $\delta_i = (true_i - a_i) \times f'(a_i)$
    - Error terms $\delta_i = \left(\sum_{s=1}^{S} \delta_s \times w_{si}\right) \times f'(a_i)$
- Weight update

$$\Delta w_{ij} = \alpha \times \delta_i \times a_j \;\; for\; j = 1...J$$

Performance Model — Model Parallelism

- Local computation time
  - Depends on number of connections, numbers of neurons, and the latency of basic operations (such as MulAdd operations)
  - U(l, p) represents the computation of the layer l in partition p.
- Remote communication time
  - Depends on the size of cross-machine activations, network latency and bandwidth
  - M(l, p) represents the communication time of the layer l in partition p
- So, T(l, p) = U(l, p) + M(l, p)

Feedforward — Local Computation time

- Feedforward evaluation

$$a_i = f\left(\left(\sum_{j=1}^{J} w_{ij} \times a_j\right) + b_i\right),$$

$$U_f(l,p) = C_{muladd} * W(l,p) + C_{act} * N_{neuron}(l,p)$$

  - Compute the output activation of neurons
  - C_{muladd} donates the time of one multiply-add operation
  - W(l, p) donates the number of weight connected from layer l - 1 to the neurons in partition p of layer l
- $N_{neuron}(l,p)$ donates the number of neurons in partition p of layer l
- C_{act} donates the time of computing the function of each neuron
- The estimation of C_{muladd} and C_{act} can be obtained by a micro benchmark which emulates feedforward evaluation only using basic operations on a worker machine.

for $(i = 0; i < N_{neuron}(l,p); i++)$ {
  foreach $(j \in \mathcal{S}_i)$ {
    $y_i += w_{ij} * a_j$ // **cost:** C_{muladd}
  }
  $a_i = f(y_i)$ // **cost:** C_{act}
}

Feedforward — Remote communication time

- Feedforward evaluation

$$a_i = f\left(\left(\sum_{j=1}^{J} w_{ij} \times a_j\right) + b_i\right),$$

$$M_f(l,p) = C_{ncost} + \frac{A(l,p) * C_{bits}}{C_{nbw}}$$

  - C_{ncost} donates the network latency of sending one bit of data between two workers
  - A(l, p) donates the number of remote activations that partition p received from layer l - 1
  - C_{bits} donates the size of each activation
  - C_{nbw} donates the bandwidth of machine's NIC.
  - Since activations can be sent to the next layer asynchronously, so the total communication time is dominated by the delay in receiving activations from previous layer.

Back propagation

- Back propagation

$$a_i = f\left(\left(\sum_{j=1}^{J} w_{ij} \times a_j\right) + b_i\right) ,$$

$$U_b(l,p) = C_{muladd} * W'(l,p) + N_{neuron}(l,p) * C_{err}$$

- Local computation time
  - Compute the error terms of neuron in layer l +1
  - W'(l, p) donates the number of weight connected from layer l + 1 to the neurons in partition p of layer l
  - C_{err} donates the time of computing error function
  - The estimation of Muladd and error function is also made through basic canonical computations, similar to the one in feedforward.
- Remote communication time

$$M_b(l,p) = C_{ncost} + \frac{E(l,p) * C_{bits}}{C_{nbw}}$$

  - E(l, p) replace A(l, p) in the one of feedforward, which is the number of remote error terms that partition p received from layer l+1

Weight update

$$\Delta w_{ij} = \alpha \times \delta_i \times a_j \;\; for\; j = 1...J$$

- Since weights are not communicated in model parallelism, the remote communication time is zero, so:

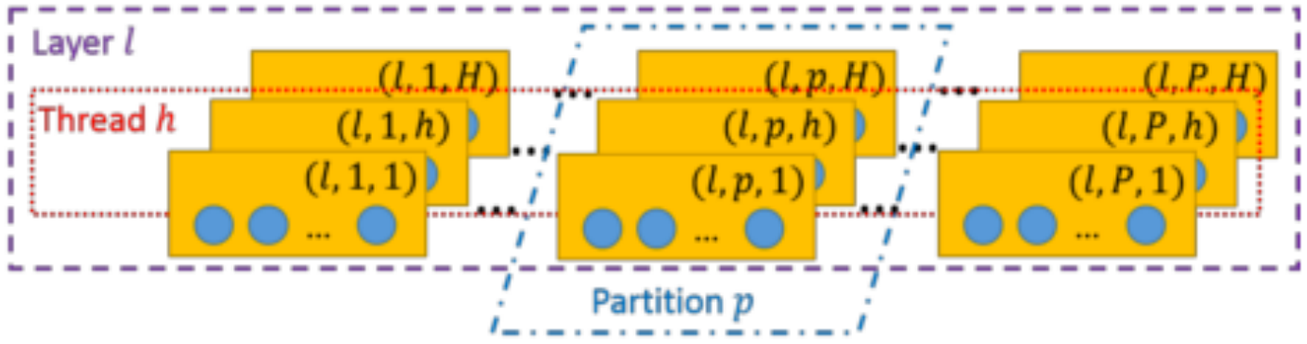
$$T_w(l,p) = U_w(l,p) = C_{muladd} * W(l,p)$$

- From the estimated time for feedforward, back-propagation and weight updates, we can finally get the train time for each layer and thus the total time to train on an example with model parallelism.

$$T_{epoch} = \sum_{l=1}^{L} [U_f(l,p) + M_f(l,p) + U_b(l,p) + M_b(l,p) + U_w(l,p)]$$

Data Parallelism — Chip-level Multiprocessing

- Data parallelism can be also applied on modern multi-core processors. The cores of a CMP system can process different samples concurrently. So the number of concurrent thread may be a configuration choice.
- A new dimension h(h ∈ [1,H(l)]) is introduced to extend the model to support data parallelism using CMP, where H(l) represent the number of threads training in parallel in layer l.



Data Parallelism — Chip-level Multiprocessing

- Local computation time

$$U_{i=\{f,b,w\}}(l,p,h) = C_{interf}(H(l)) * U_i(l,p)$$

- $C_{interf}(H(l))$ is a performance interference factor to model the interference among threads. It's estimated as the ratio of the H(l)-thread execution time and the single-thread by running a multi-threaded version of basic operation such that each thread processes the same code using different cores.
- It's possible for $C_{interf}(H(l))$ to be larger than one, which cause computation time of training one example increases. But running multiple samples concurrently can still reduce the epoch time of training the entire sample set.
- Q(l) is defined as data parallelism degree which is to represent the concurrency degree of training samples in parallel in layer l in order to define the epoch time.
- In this case, Q(l) = H(l)

Data Parallelism — Chip-level Multiprocessing

- Remote communication time

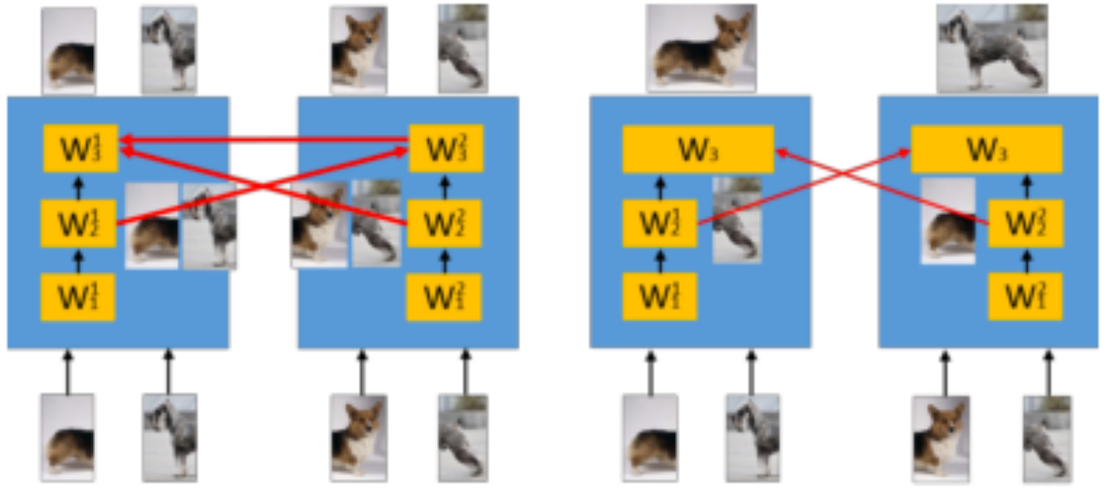
$$M_f(l,p,h) = C_{ncost}(H(l))) + \frac{A(l,p,h) * C_{bits}}{C_{nbw}(H(l))}$$

$$M_b(l,p,h) = C_{ncost}(H(l))) + \frac{E(l,p,h) * C_{bits}}{C_{nbw}(H(l))}$$

- When training multiple samples with multiple threads, the network bandwidth is shared among the threads, so each threads can only gets 1/H(l) of the bandwidth, we define which is $C_{nbw}(H(l))$
- $C_{ncost}(H(l)))$ denotes the network latency which is effected by H(l) because the latency may increase when establishing H(l) concurrent connection sending and receiving activations/error terms.

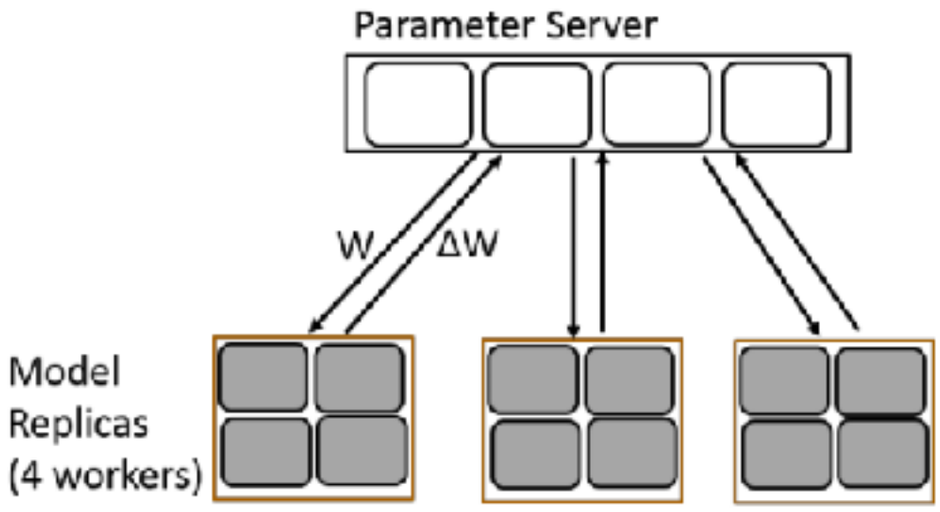
Data Parallelism — Layer Replication

- Since partitioning a fully connected layer may produce a large amount of data communication, a better alternative is to replicate the whole layer among several machines where each machine processes a subset of training data.
- A new dimension r is introduced to extend the model to support data parallelism using layer replication.
- The computation time is the same as that of using CMP, and the communication time is very similar.
- The data parallelism degree Q(l) = H(l) * R(l), R(l) denote to the number of layer replications.



Data Parallelism — Model Replicas

- The number of model replicas and parameter servers are the critical configuration choice to balance computation and communication.
- Computation time doesn't change as before. The data parallelism degree is extended to Q(l) = H(l) * R(l) * N_{mr}, where N_{mr} donates the number of replicas.



Data Parallelism — Model Replicas

- Communication time
  - The communication of weight update between replicas and parameter servers needs to be considered (only consider reading weights). However, the time it takes depends on the communication pattern. The worst case is when all replicas read weights from the same server simultaneously, the time will be:

$$M_w^{max} = \sum_{l=1}^{L}(C_{ncost}(h(l)) + \frac{N_{mr} * W(l)}{C_{nbw}(H(l))})$$

- where $W(l) = \Sigma_{1 \le p \le P(l)} W(l,p)$
  - The best case occurs when a single replica uses all of its workers and reads from all parameter servers in parallel, and there is no overlapped among multiple replicas while reading.

$$M_w^{min} = \sum_{l=1}^{L}(C_{ncost}(h(l)) + \frac{W(l))}{C_{nbw}(H(l)) * min(N_{ps}, N_{wr})})$$

Performance Model

- We get the estimation of the epoch time of the complete model.

$$T_{epoch} = \frac{M_w * N_{sample}}{N_{read}} + \sum_{l=1}^{L} \{[U_f(l,p,h,r) + M_f(l,p,h,r) + U_b(l,p,h,r) + M_b(l,p,h,r) + U_w(l,p,h,r)] * N_{sample}/Q(l)\}$$

- where $U_{i=\{f,b,w\}}(l,p,h,r) = C_{interf}(H(l)) * U_i(l,p)$

$$M_f(l,p,h,r) = C_{ncost}(H(l)) + \frac{A(l,p,h,r) * C_{bits}}{C_{nbw}(H(l))}$$

$$M_b(l,p,h,r) = C_{ncost}(H(l)) + \frac{E(l,p,h,r) * C_{bits}}{C_{nbw}(H(l))}$$

$$Q(l) = H(l) * R(l) * N_{mr}$$

- Then we can use the performance model to do scalability optimization.

Scalability optimizer

- The purpose of a scalability optimizer is to enumerates different system configurations and use to proposed performance model to estimate the training time, and finally finds out the optimal configuration.
- Here is the problem formulation for the optimizer:

Variables to define a configuration Φ

- N_{ps}: number of parameter servers
- N_{mr}: number of model replicas
- N_{wr}: number of workers per model replica; $\mathcal{SW}$=$\{1, ..., N_{wr}\}$

For each layer $1 \leq l \leq L$,

- P_l: number of partitions; $\mathcal{SP}_l$=$\{1, ..., P_l\}$
- H_l: number of threads; $\mathcal{SH}_l$=$\{1, ..., H_l\}$
- R_l: number of replicas; $\mathcal{SR}_l$=$\{1, ..., R_l\}$
- mapping function f_l: $\mathcal{SP}_l \times \mathcal{SH}_l \times \mathcal{SR}_l \rightarrow \mathcal{SW}$

Objective:

$minimize_{\{\Phi\}} T_{epoch}(\Phi)$

Subject to:

C1: $N_{ps} + N_{mr} \times N_{wr} \leq N$

C2: $H_l \leq K$, for all $1 \leq l \leq L$ (K is total number of cores of a machine)

Brute-force search

- The simplest way to search is to traverse the configuration space to find the best performed one.
- To find the proper of resource parameters, there are N^3 combinations.
- To find an assignment that minimizes the remote communications, mapping of segments and worker is a permutation that results in O(n!) complexity.
- At each layer, the selection on the number of partitions and replicas will also be N^2
- There are K choices of the number of threads
- So the complexity of the brute-force search is

$$O(N^3[K * N^2 * (N!)]^L)$$

Efficient Search Algorithm

- Optimizing segment-worker mapping
    - Using a greedy approach, for each segment, just finding a worker which can minimize the remote connection time from previous. The time complexity is O(n). So for all segments, the complexity is O(N^2)
- Optimizing multi-layer composition
    - An optimal solution for up to l layers can be constructed using the optimal of sub-problems for up to layer l - 1. The key observation is that an optimal solution for up to l layers can be constructed using the optimal of sub-problems for up to layer l-1.

Optimizing multi-layer composition

- Donate $T_{epoch}([1,l],p_l)$ as the accumulated training time from layer 1 to layer l.

$$T_{epoch}([1,l],p_l) = \min_{1\le p_l-1\le N}[T_{epoch}([1,l-1],p_{l-1}) + U(l,p_l) + M(l,l-1,p_l,p_{l-1})]$$

- Where $U(l,p_l)$ is the computation time of layer l, and $M(l,l-1,p_l,p_{l-1})$ is the communication time between layer l-1 and layer l

- Dynamic programing is applied to prevent the cost being exponential. To get each T_{epoch} , the cost is N times segment-worker mapping cost, i.e., N^3. And if we consider replica, the complexity of per layer becomes O(N^4), and total complexity is O(L * N^4)
- To sum up, the entire complexity can be $O(L*K*N^7)$, reducing from exponential time to polynomial time.
    - And complexity can be further reduced to $O(L*K*N^5log^2N)$

Evaluation

- Methodology
  - Distributed deep learning system
    - Adam http://web.eecs.umich.edu/~mosharaf/Readings/Project-Adam.pdf
  - Benchmarks
    - MINST
      - DNN contains about 2.5 million connections in 5 layers: 2 convolutional layers, 2 linear layers and an output layer
    - ImageNet-22K
      - DNN contains over 2 billion connections in 8 layers: 5 convolutional layers, 2 linear layers and an output layer
- Computer Cluster
  - A cluster of 20 identically configured commodity servers connected by Ethernet
  - For each server:
    - Xeon-E2450 with 16 cores running at 2.1GHz
    - 64 GB of memory
    - 268.8 GFLOP/s SIMD FPU
    - A single 10Gbps NIC

Performance Model Validation

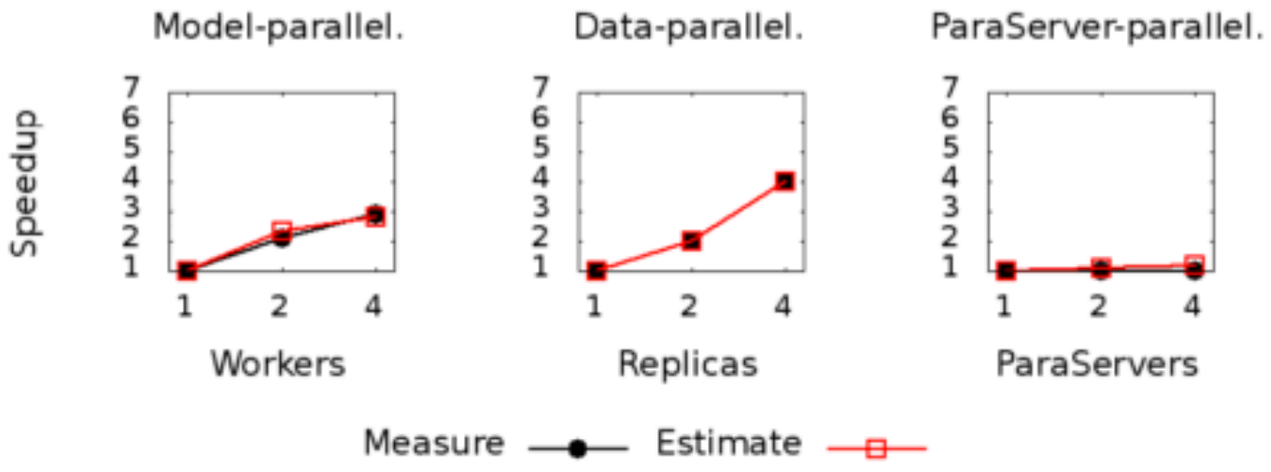


Figure 8: Scalability results of MNIST.

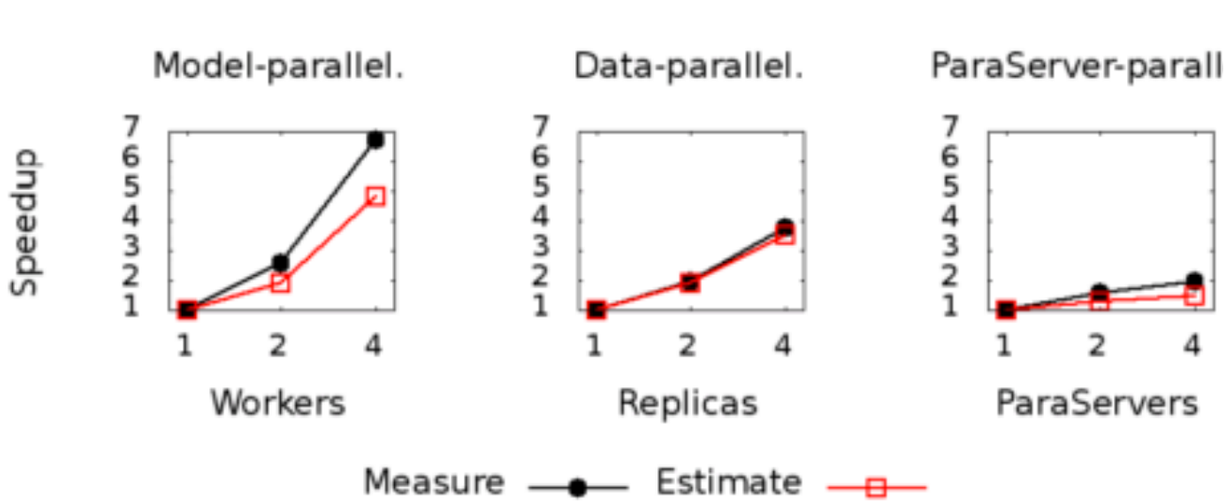


Figure 9: Scalability results of ImageNet-22K.

MNIST		
Configuration #W/#R/#PS	Rank	
	Measure	Estimate
2/9/2	1	1
2/8/4	2	2
2/6/8	3	3
4/4/4	4	4
1/19/1	5	5
1/18/2	6	6
1/16/4	7	7
1/12/8	8	8

(a)

ImageNet-22K		
Configuration #W/#R/#PS	Rank	
	Measure	Estimate
4/4/4	1	1
4/4/2	2	2
2/9/2	3	3
4/3/6	4	4
4/3/4	5	5
2/8/4	6	6
2/6/6	7	7
2/6/4	8	8

(b)

Table 1: Training speed ranking (1 is the fastest). W, R, and P stand for worker, replica, and parameter server respectively.

Performance Model Validation

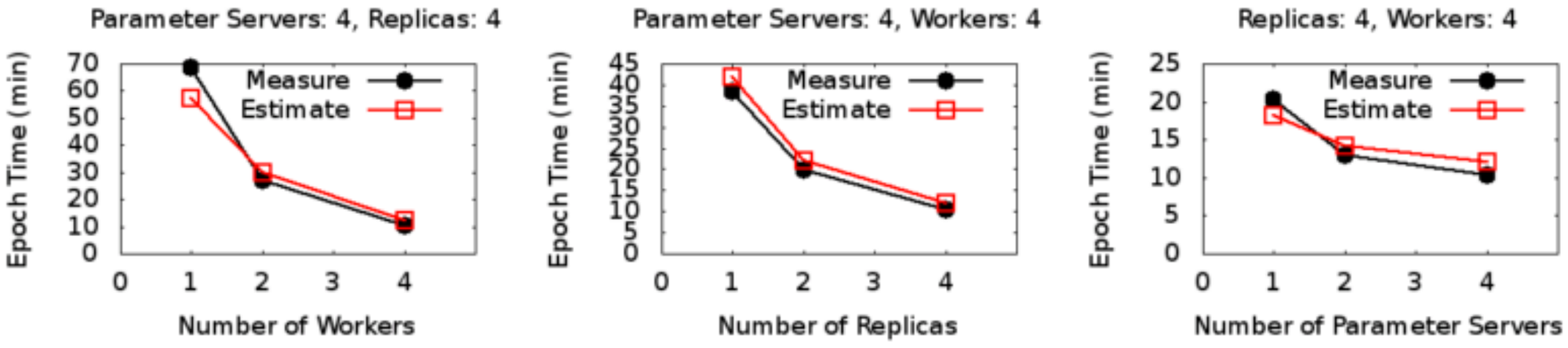


Figure 10: Estimated and measured epoch training time in different scenarios of ImageNet-22K.

- Estimation error <25%

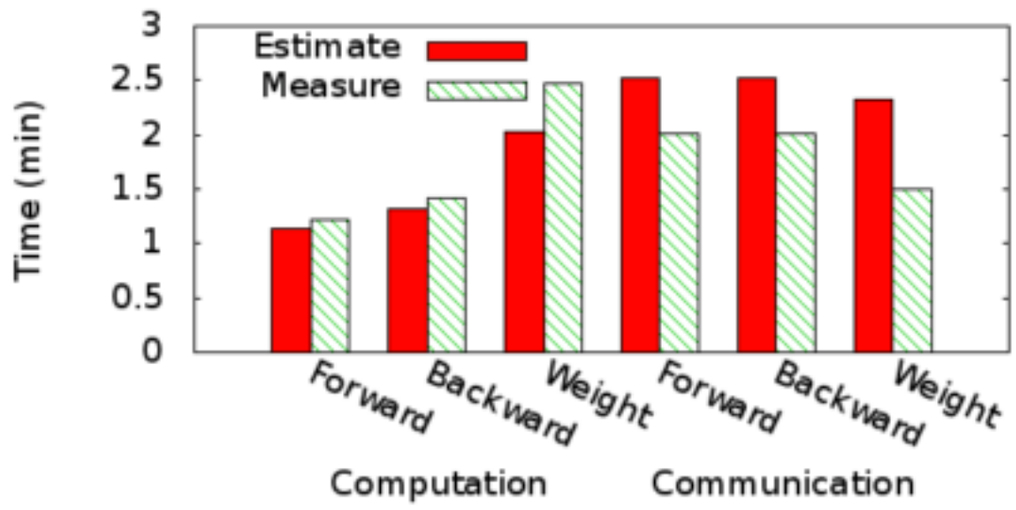


Figure 11: Epoch training time breakdown of ImageNet-22K.

Problems

- The estimation of communication time is still less accurate than that of computation time.
- Can this performance model estimate art-of-the-state DNN models which the number of layers is over 100?

The end

Thank you for listening!

Zhang Chenwu
17M38153