Distributed Training of Deep Neural Network Acoustic Models for Automatic Speech Recognition

A comparison of current training strategies

The past decade has witnessed great progress in automatic speech recognition (ASR) due to advances in deep learning. The improvements in performance can be attributed to both improved models and large-scale training data. The key to training such models is the employment of efficient distributed learning techniques. In this article, we provide an overview of distributed training techniques for deep neural network (DNN) acoustic models used for ASR. Starting with the fundamentals of data parallel stochastic gradient descent (SGD) and ASR acoustic modeling, we investigate various distributed training strategies and their realizations in high-performance computing (HPC) environments with an emphasis on striking a balance between communication and computation. Experiments are carried out on a popular public benchmark to study the convergence, speedup, and recognition performance of the investigated strategies.

Introduction

Deep learning has had great impact across a broad variety of areas such as computer vision, natural language processing, and ASR over the past several years. It is also the driving force behind the current artificial intelligence technologies that have achieved unprecedented success in a wide spectrum of applications. ASR is one of the first areas that has witnessed performance breakthroughs using deep-learning techniques [1]. Models that employ deep architectures are dominant in today’s ASR systems that are ubiquitous in our everyday life, for example, Google’s voice search, Amazon’s Alexa, Apple’s Siri, and IBM’s Watson speech-to-text service. On some data sets, the word error rates (WERs) of high-performance ASR can even achieve human parity [2], [3].

The high performance of deep learning relies heavily upon large amounts of training data and high computational power. For instance, today, the amount of training speech data for ASR can easily reach thousands of hours, is often tens of thousands of hours, with some tasks even utilizing as much as one million hours [4]. Moreover, DNNs can have tens of millions of parameters or more. Without an appropriate distributed training strategy, training such models with deep architectures using this order
of magnitude of data may take months to finish on a single GPU. It is therefore critical to investigate strategies that can speed up the training and reduce the turn-around time to an acceptable level. In this context, distributed learning has been demonstrated to be a very effective approach for training speedup.

Over the years, various distributed machine learning strategies have been proposed and applied to different tasks. Distributed training of acoustic models as a branch of distributed machine learning has much in common with other domains such as computer vision in terms of algorithmic and system design; however, it also has characteristics unique to the ASR domain. Therefore, directly borrowing techniques from other domains may not offer the best training performance for ASR. In the speech community, distributed acoustic modeling has been an active research topic. Notable work includes parameter server (PS)-based synchronous training [5], [6], PS-based asynchronous training [7], [8], and decentralized asynchronous training [9], [10].

This article reviews commonly used techniques for the distributed training of acoustic models. We first discuss some fundamentals of parallel SGD (PSGD), HPC architectures, and deep acoustic modeling, which are the foundation of current large-scale distributed training. We then present an in-depth investigation of several distributed strategies including synchronous/asynchronous and centralized/distributed schemes and discuss their pros and cons. Specifically, emphasis is placed on the interplay between computation and communication, the most important factor to consider when designing a successful distributed training strategy. Experiments are carried out on the 2,000-h Switchboard (SWB2000) data set [11], [12], one of the most widely used public benchmark data sets in the speech community [2], [3]. We conclude by analyzing the performance of various strategies. In particular, we show that the asynchronous decentralized (AD)-PSGD algorithm recently proposed by IBM has achieved one of the best speedup performances to date on this data set.

**Optimization using SGD**

**Algorithm**

Most of the machine learning problems that employ DNNs are optimized using SGD. Suppose $\mathbf{X} \subseteq \mathbb{R}^{d_{x}}$ and $\mathbf{Y} \subseteq \mathbb{R}^{d_{y}}$ are the input and output spaces, respectively, of a supervised learning problem. We want to estimate a function $h$ with parameters $w$ that maps the input to the output

$$h(w; x) : \mathbf{X} \rightarrow \mathbf{Y}. \quad (1)$$

A loss function $f(h(w; x), y)$ is used to measure the closeness between the prediction $h(w; x)$ and label $y$ where $x \in \mathbf{X}$ and $y \in \mathbf{Y}$. A risk function $F(w)$, given parameters $w$, is defined as the expected loss over the underlying joint distribution $p(x, y)$

$$F(w) = \mathbb{E}_{(x,y)}[f(h(w; x), y)] = \mathbb{E}_{\xi}[f(w, \xi)], \quad (2)$$

where $\xi \sim p(x, y)$ is a random variable on data $(x, y)$. We want to choose parameters $w$ that minimize $F(w)$

$$w^{*} = \arg\min_{w} F(w). \quad (3)$$

In practice, we only have access to a set of $n$ training samples $\{(x_i, y_i)\}_{i=1}^{n}$. Accordingly, we minimize the following empirical risk:

$$F(w) = \frac{1}{n} \sum_{i=1}^{n} f(w, (x_i, y_i)), \quad (4)$$

where $\xi$ assumes the empirical data distribution.

When it comes to the large-scale optimization of DNNs in (3), SGD is the dominant technique employed in deep learning due to its computational efficiency and competitive performance over other more complex optimization algorithms [13]. SGD solves the optimization problem of (3) iteratively. A basic SGD update formula is given by (5)

$$w_{k+1} = w_k - \alpha_k \left[ \frac{1}{M} \sum_{m=1}^{M} \nabla f(w_k; \xi_{k,m}) \right], \quad (5)$$

where $w_k$ are the parameters after iteration $k$, $\alpha_k$ is the learning rate, and $\nabla f(w_k; \xi_{k,m})$ is the gradient evaluated at $w_k$ using the data samples denoted by the random variable $\xi_{k,m}$. There are $M$ samples randomly drawn from the whole $n$ training samples to form a so-called minibatch. Their aggregated gradient is considered a “noisy” version of the true gradient $\nabla F(w)$ and hence a stochastic approximation [14] of the deterministic gradient descent method. Therefore, it is often referred to as the minibatch-based SGD and $M$ is the batch size. In addition to the basic SGD algorithm given in (5), a variety of variants have been proposed in the literature to improve its convergence properties, the most notable being Adagrad [15], Adam [16], and Nesterov acceleration [17]. These SGD variants have found varied degrees of success in different applications. In this article, for tutorial purposes and ease of discussion, we will focus on the basic SGD form in (5).

**Training strategies**

The strategies of distributed machine learning can be broadly categorized into the families discussed in the next sections.

**Data parallel versus model parallel**

Data parallelism distributes minibatches of data onto a number of learners (e.g., a GPU or CPU), with each learner having a copy of the model. The computation is carried out on each learner in parallel using the data before the system aggregates the statistics in some fashion. Sometimes, the model is too large to fit into the memory of one single learner; under this condition, model parallelism is used to split the model across the learners, with each learner having only a partial model. The output of one learner is used as the input of another learner to conduct the computation of the full model. Although model parallelism is used in some scenarios [5], data parallelism is the dominant distributed strategy in practice in today’s distributed learning community [6], [7], [18].
Single-node versus multiple nodes

A node refers to a physical device (e.g., a server) in a computer network. Distributed training on a single node is a straightforward setup where all learners (e.g., GPUs) stay within one machine with a shared memory. The communication among the learners is reliable and can give a moderate speedup; therefore, it is still a reasonable option for the distributed training of DNNs. Obviously, it has the limitation of the capability of scaling out, and the number of learners is limited to the hardware configuration of the machines. Multiple-node distributed training has a number of machines (nodes) that form a cloud or cluster where learners use shared memory for local communication within the node and message passing over the network for internode communication. It has become the norm for recent large-scale distributed training.

Centralized versus decentralized

Centralized distributed training relies on a central PS and all learners only communicate with the PS for model updates, whereas decentralized setting has no PS and all learners form a network of certain topology (e.g., a ring). All learners are in an equal position in terms of computation and communication. In practice, centralized distributed training is commonly used in many machine learning applications; however, it imposes a large communication bottleneck on the PS as the hub when the number of learners is large. Conversely, because decentralized distributed training has an advantage on the communication bandwidth, it is becoming more and more popular.

Synchronous versus asynchronous

Distributed training algorithms can be run under synchronous or asynchronous mode. Under synchronous mode, the gradient computation and model update are synchronized with each other. The model update is carried out after all learners finish local gradient computation. Under asynchronous mode, however, the synchronization between the gradient computation and model update is removed and learners receive their minibatches as needed depending on their computation and communication speed, which significantly reduces the idle time relative to the synchronous mode.

In the next section, we focus on data-parallel, multiple-node distributed acoustic modeling because it is the most popular distributed setting nowadays. We present an in-depth investigation of its behavior in centralized/decentralized configurations under synchronous/asynchronous modes.

HPC architecture

Distributed computing aims at parallelizing execution among independent computational resources (e.g., processors). A parallel program consists of three components: 1) the parallel part that can be carried out in parallel, for example, gradient computation; 2) the sequential part that cannot be parallelized. For example, the summation of gradients or models can only be accomplished after the gradients or models are in place; and 3) the communication portion that passes information between computational resources, for instance, gradient/weight transfer among learners. Assuming the communication cost is zero, Amdahl’s law states that the speedup of a parallel program is bounded by \(1/(1-p)\), where \(p\) is the portion of the parallelizable part of the program. For the SGD algorithm, the parallel part dominates the sequential part; hence, \(p\) is very close to 1 and very high speedups can be achieved in practice. As a result, the fundamental limiting factor to achieve linear speedup, in this case, is the communication cost.

A typical distributed training system, as depicted in Figure 1, has the following hardware components: data storage, memory, processing units (CPUs/GPUs), and the network. Data communication follows this path: Each learner loads input data from data storage under storage bandwidth constraint to main memory. It conducts data preprocessing using the CPU before sending it to the GPU via the CPU–GPU data bus for model training. When the gradient computation is finished, the gradients are sent to the PS or other learners under the constraint of network bandwidth. The normal bandwidth of storage systems ranges from 1 to 10 megabyte/s (network file system) to 100 megabyte/s (hard disk drives) to 300–500 megabyte/s [solid state drives (SSDs)] to several gigabytes/s [nonvolatile memory express (NVMe) SSDs]. The typical main memory bandwidth is of several tens of gigabytes/s. The normal CPU–GPU data bus bandwidth ranges from 16 gigabytes/s one way [e.g., Peripheral Component Interconnect Express (PCI-e) third generation] to 50 gigabytes/s one way (e.g., IBM Power 9 Nvlink). The typical network bandwidth ranges from 100 megabyte/s (e.g., 1-gigabit Ethernet) to 10 gigabytes/s (e.g., 100-gigabit Ethernet, remote direct memory access). A high-end HPC cluster (e.g., SUMMIT supercomputer) usually is equipped with NVMe SSDs, NVlinks, and 100-gigabit Ethernet (or higher) to enable fast communication.

On the computation side, the key deciding factor is floating point operations/s (flops). Typical high-end CPUs run at

![Figure 1. The components and data flow in an HPC environment used for distributed training. DRAM: dynamic random access memory; I/O: input/output.](image-url)
hundreds of gigaflops to 1 teraflops, and typical high-end GPUs run at 10 teraflops or higher. System programmers use concurrent programming, usually achieved by multithreading, to overlap communication with computation. Under synchronous mode, data loading and data processing can be overlapped with gradient computation, while under asynchronous mode, all of the communication paths (i.e., data loading, data preprocessing, CPU–GPU data transfer, and gradients/weights transfer) can be overlapped with gradient computation. Ideally, we wish to pursue perfect overlaps between communication and computation. In practice, the communication that cannot be overlapped by computation is the limiting factor in achieving linear speedup. One of the important tasks in designing distributed learning is to maximize the overlap between the two.

**Acoustic modeling in ASR**

Suppose $X = \{x_1, x_2, \ldots, x_m\}$ is a sequence of acoustic features and $W = \{w_1, w_2, \ldots, w_n\}$ is a sequence of words. ASR systems find the most likely word sequence $W$, given the observed acoustic feature sequence $X$ (Figure 2):

$$W^* = \arg\max_w P(W|X) = \arg\max_w \frac{P(X|W)P(W)}{P(X)}.$$  \hspace{1cm} (6)

There are four major components in an ASR system. A feature extractor converts a speech waveform into a sequence of acoustic feature vectors [e.g., log Mel-frequency (logMel) features]. An acoustic model computes the probability that a particular word sequence can produce the observed acoustic features, $P(X|W) = P(x_1, x_2, \ldots, x_m|w_1, w_2, \ldots, w_n)$. A language model computes the probability of a particular word sequence, $P(W) = P(w_1, w_2, \ldots, w_n)$. Finally, there is a decoder that searches for the best word sequence $W^*$, which maximizes (6).

**Figure 2.** The components of an ASR system.

### Table 1. A model comparison between speech and computer vision

| Model   | Layers | Output Size (Megabytes) | Computation/Batch (s) |
|---------|--------|-------------------------|-----------------------|
| Vision  | ResNet | 50                      | ~ 100                 | ~ 0.18               |
| Speech  | LSTM   | Eight                   | ~ 160                 | ~ 0.07               |

In this article, we consider DNN acoustic models based on a hidden Markov model (HMM) structure, often referred to as **DNN-HMM**. HMMs are probabilistic automata commonly used for modeling sequences of variable length, such as speech. Under this structure, words are represented as strings of speech sounds, called *phones*. Each phone is represented by an HMM with numerous states. Because a phone can be affected by its context (i.e., phones immediately before and after it), modern ASR systems use so-called context-dependent (CD) phones. The HMM states of each CD phone are clustered using a decision tree based on their acoustic similarity. This gives rise to a large number of CD HMM states as fine-grained phone classes used for acoustic modeling. In DNN-HMMs, the DNN output after the topmost softmax represents a set of posterior probabilities corresponding to each of the CD HMM states [19]. Modern acoustic models use either convolutional neural networks or recurrent networks, the most popular of which are long short-term memory (LSTM) networks [2], [3].

Training acoustic models involves optimizing the DNNs under an appropriate objective function. In ASR, frame-based cross entropy (CE) [1] and sequence-based loss functions such as state-level minimum Bayes error [20] are commonly used for optimization. In recent years, end-to-end ASR systems based on connectionist temporal classification [21] or encoder-decoder [22] structures are also drawing attention in the speech community. The discussion in this article is focused on the LSTM-based DNN–HMM acoustic models trained with the CE criterion to investigate the distributed training strategies in ASR because they are most representative. But, the techniques presented are applicable in principle to other models and training criteria as well.

### Distributed training of acoustic models

**Unique characteristics**

DNN acoustic models have distinct characteristics from other domains (e.g., computer vision) in terms of distributed training. A DNN-HMM acoustic model typically has a softmax layer with a large number of output CD phone classes [2], [3], [19], which are usually on the order of magnitude of 10,000. Moreover, the distribution of speech samples across phone classes is hugely uneven. In addition, the input feature space is relatively more structured for speech signals; therefore, the DNN acoustic models are usually shallower than those in vision. Table 1 shows the configurations of two representative DNN models used for speech and vision tasks, a 50-layer residual neural network (ResNet) model utilized for the ImageNet image recognition task, and a six-layer LSTM model used for the SWB ASR task. The ResNet model has a smaller model size with even more convolutional layers due to parameter sharing and local connectivity; however, the convolutional operation is computationally expensive, which results in a longer computing time for each batch. Conversely, the LSTM model for the speech task has a bigger model but the computation is faster. Overall, the optimization of an acoustic model has a lighter
computational load but a heavier communication load relative to the vision models. This high communication/computation ratio imposes a major challenge for the distributed training of DNN acoustic models.

**Centralized distributed training**

Centralized distributed training has a PS serving as a hub in the network, as shown in Figure 3(a). The PS has the global view of the model and all learners communicate only with the PS. Each learner pulls the model from the PS. The computation of gradients is carried out locally on each learner, after which the gradients are pushed back to the PS. The PS collects the statistics from the learners and updates the model accordingly depending on whether the implementation is synchronous or asynchronous.

Consider the SGD iteration given in (5). Suppose each mini-batch $M$ is evenly split onto $L$ learners with each learner $l$ having a batch size of $M_l = M/L$.

**Synchronous PSGD**

Under synchronous PSGD, each learner $l$ pulls model $w_l$ from the PS, computes the local gradient

$$g^l(w_l, \xi_l) = \frac{1}{M_l} \sum_{m=1}^{M_l} \nabla f(w_l^m; \xi_{l,m}), \quad (7)$$

and then pushes $g^l(w_l, \xi_l)$ back to the PS.

The PS will wait until all $L$ learners finish their local gradient computation to aggregate them up for model update

$$w_{k+1} = w_k - \alpha_k \left[ \frac{1}{L} \sum_{l=1}^L g^l(w_l, \xi_l) \right], \quad (8)$$

Note that all learners use the same copy of the model $w_l$ for the computation of $g^l(w_l, \xi_l)$, which is consistent with the global copy of the model residing on the PS. Equations (7) and (8) give the same update as (5). Synchronous PSGD is the most reliable implementation of SGD in a distributed setting and shares the same convergence property. Synchronous SGD may suffer from the well-known “straggler” problem because the PS has to wait for the slowest learner. This synchronization cost limits the overall speedup. Nevertheless, because of its simplicity, the centralized synchronous distributed training of acoustic models is still popular in the speech community. Representative work includes [5] and [6] from Microsoft, [4] from Amazon, and [23] from Baidu.

The global model updates can be conducted either through gradient aggregation [5], [23] or model averaging [6]. A notable work among them is a strategy based on blockwise model-update filtering (BMUF) [6] proposed by Microsoft, which is a variant of synchronous SGD. Under BMUF, data are partitioned into blocks. Each worker updates its local model in parallel using SGD. Instead of performing direct model averaging, the global model is updated using block-level stochastic optimization by synchronizing the local models from all learners based on the block momentum. Good performance has been reported by Microsoft on large-scale distributed acoustic modeling and has also been used by Amazon to train DNN acoustic models using one million hours of data [4].

**Asynchronous PSGD**

Under asynchronous PSGD, each learner $l$ pulls model $\hat{w}_l$ from the PS and computes the local gradient

$$g^l(\hat{w}_l, \xi_l) = \frac{1}{M_l} \sum_{m=1}^{M_l} \nabla f(\hat{w}_l^m; \xi_{l,m}), \quad (9)$$

and then pushes $g^l(\hat{w}_l, \xi_l)$ back to the PS.

The PS updates the model right after receiving the gradient from learner $l$:

$$w_{k+1} = w_k - \alpha_k \cdot g^l(\hat{w}_l, \xi_l). \quad (10)$$

In this case, there may exist inconsistency between the model $w_k$ on the PS and the model $\hat{w}_l$ pulled by learner $l$ for the computation of its local gradient

$$\hat{w}_k = w_{k-\tau}, \quad \tau \geq 0. \quad (11)$$

This is because the model on the PS may have been updated by other learners while learner $l$ is still computing its local gradients. This inconsistency is often referred to as staleness. Asynchronous PSGD, with the synchronization removed between the learner and server, can significantly reduce idle time and improve the speedup. It does not have the “straggler” problem and can automatically balance the workload among the fast and slow learners. Nevertheless, the incurred staleness may hurt convergence and, eventually, the learning performance.

In the speech community, the centralized asynchronous distributed training of acoustic models has also been used [7], [8]. Downpour SGD based on the DistBelief framework [24], proposed by Google, is a representative

**FIGURE 3.** (a) Centralized distributed training with a PS and (b) decentralized distributed training without a PS, where communication takes place among the learners.
application. Using this strategy, distributed training ranging from multilingual acoustic modeling [7] to sequence acoustic modeling [8] has been reported to deliver good performance. Downpour SGD is a variant of asynchronous SGD and divides the data into blocks, distributing them onto multiple learners. Each learner keeps a local copy of the model and uses it to carry out computation in parallel. They independently push the updates to the PS, which keeps the current state of model, and then pull the updated model from the PS. The PS itself is sharded across multiple machines and each shard is responsible only for updating part of the model. Downpour SGD introduces asynchrony to both local learners and PS shards and has demonstrated robustness to machine failure during training. But, centralized asynchronous distributed training may be bothered by the potential large staleness issue in general, and it is challenging to have both good parallelization efficiency and convergence behavior.

### Decentralized distributed training

Centralized distributed training relies on a PS to communicate with all of the learners. All the communication takes place between the server and the learners and there is no communication among the learners. This introduces a high communication cost at the PS, which is proportional to the number of learners, and will eventually hurt the scaling of the training. A PS does not necessarily need to be realized as a physical server or sharded servers [24]. It can also be conceptually realized via the allreduce operation [25] based on message passing. In HPC terminology, an operation is a "reduce" operation if it is commutative and associative (e.g., summation). Allreduce is the operation that reduces all of the elements and broadcasts the reduction results to each participant. Decentralized distributed training does not require a centralized server to support the communication. Figure 3(b) shows how decentralized SGD works. The learners form a network by connecting with each other following some topology (e.g., a ring). Each learner keeps a local copy of the model and carries out gradient computation and model updates locally. The updated local model is then propagated to other learners in the network typically via model averaging.

Consider the centralized data PSGD setting of (5) under synchronous mode. Suppose each learner pulls the model from the PS, evaluates the gradient using batch size $M_l$, and updates the model locally:

$$w_{k+1}^{(l)} = w_k - \alpha_l \frac{1}{M_l} \sum_{m=1}^{M_l} \nabla f(w_k; \xi_{k,m}) .$$

Then we average models across all of the learners:

$$w_{k+1} = \frac{1}{L} \sum_{l=1}^{L} w_{k+1}^{(l)} = w_k - \alpha \frac{1}{M} \sum_{m=1}^{M} \nabla f(w_k; \xi_{k,m}) .$$

which shows that, given the basic SGD in (5), one-step model averaging and gradient averaging are equivalent. Equation (13) also provides a way to create a decentralized implementation of centralized SGD where all of the learners form a ring and the PS is replaced by a global model averaging carried out by the allreduce operation using reduction (summation) followed by broadcast.

In general, the mathematical model of data-parallel decentralized SGD is given by (14):

$$W_{k+1} = W_k - \alpha_k \cdot g(\Phi_k, \xi_k) .$$

where, for $l = 1, \ldots, L$:

- $W_k = [w_k^{(1)}, \ldots, w_k^{(L)}]$ is a matrix, with each column containing model parameters in each learner $l$ at iteration $k$.
- $T$ is a mixing matrix for the model-averaging pattern among learners, given a network topology.
- $\Phi_k = [\xi_k^{(1)}, \ldots, \xi_k^{(L)}]$ is a matrix, with each column containing model parameters used for computing gradient in each learner $l$ at iteration $k$.
- $\xi_k = [\xi_k^{(1)}, \ldots, \xi_k^{(L)}]$ is a matrix, with each column containing indexing random variables for minibatch samples used for computing gradients in each learner $l$ at iteration $k$.
- $g(\Phi_k, \xi_k) = \{[1/M_1] \sum_{m=1}^{M_1} \nabla f(w_k^{(1)}; \xi_{k,m}), \ldots, [1/M_L] \sum_{m=1}^{M_L} \nabla f(w_k^{(L)}; \xi_{k,m})\}$ is a matrix, with each column containing gradients computed in each learner $l$ at iteration $k$.

The first term on the right in (14) describes the communication pattern among learners, while the second term depends on the gradient computation on each learner. The two terms can be evaluated concurrently. Each learner keeps a local model and computes the gradients locally. Meanwhile, the local model is also averaged with other learners in the network through the mixing matrix. If the computation takes a longer than the communication, the first term can be entirely overlapped by the second term. Equation (14) can also be carried out under synchronous [9], [26] or asynchronous [9], [27] modes. Under synchronous mode, the model update has to hold until the two terms are both in place, while under asynchronous mode, the model update takes place whenever a learner finishes its local computation. By introducing various synchronization mechanisms between the two terms, (14) can cover a broad variety of decentralized training strategies.

Note that, although it is suitable for the theoretical analysis of convergence behaviors of a training strategy, (14) does not reflect its communication cost, which mainly includes the amount of time needed for data transfer from storage and memory, the model/gradient transfer between the CPU and GPU, and model/gradient averaging among learners.

Model averaging in the decentralized SGD is indicated by the mixing matrix $T$, which is typically chosen as doubly stochastic matrices. A matrix $T = (t_{ij})$ is called a doubly stochastic matrix if $t_{ij} \in [0, 1]$ and $\Sigma_i t_{ij} = \Sigma_j t_{ij} = 1$. For instance,
where \( T_1 \) represents a model-averaging scheme in which each learner averages its local models with its immediate left and right neighbors in a ring. \( T_a \) represents the scheme where local models of all learners are averaged. Treating \( T \) as a transition matrix of a Markov chain, if its represented chain is irreducible and aperiodic, it has a stationary uniform distribution \( T^n \rightarrow T_a \) when \( n \rightarrow \infty \). It indicates that, with sufficient rounds of model averaging under \( T \), the local models of all learners will reach a consensus, which is the average of all the local models.

In recent years, decentralized PSGD has been theoretically shown to be equally good in terms of convergence rate as that of conventional SGD [26], [27]. Communication wise, decentralized PSGD is advantageous over the centralized strategies because it removes the communication banner on the PS. Recent work from IBM [9], [10] has demonstrated good performance in both speedup and WER using AD strategies in large-scale acoustic modeling. In particular, a hybrid distributed setting was proposed in [10] that combines synchronous and asynchronous modes under the same decentralized PSGD framework. AD strategies also tolerate larger batch sizes relative to the centralized strategies.

**Improving training efficiency**

When designing a distributed training strategy based on SGD, batch size and communication bandwidth are two critical factors to consider in practice for training efficiency.

For data PSGD, the speedup is roughly proportional to the batch size under the constraints of GPU memory and model size. The more data we can parallelize in each batch, the faster the training is. It is difficult to increase the number of learners while maintaining a high percentage of GPU usage if the batch size is not sufficiently large. However, it is often observed that a too-large batch size may hurt the convergence of SGD and, eventually, the performance of the model [10]. Therefore, effectively increasing the batch size without compromising performance has been actively investigated in the distributed training community [9], [18]. To address the batch size issue, a learning-rate warm up is often used. A common practice is that a large batch is learned using a large learning rate, which are roughly in proportion, but the large learning rate is achieved by gradually scaling up from a small learning rate. In practice, this strategy usually gives good performance. It is also observed that larger batch sizes are possible in decentralized asynchronous SGD using partial model averaging [10].

Bandwidth indicates how much data can be communicated per second. In PSGD, we always hope for a perfect overlap between communication and computation to minimize the training time. The communication takes place when data are copied from storage to memory, models/gradadients are transferred between the CPU and GPU, and models/gradadients are averaged among learners. The computation mainly involves gradient evaluation. We aim to push computation-heavy operations to GPUs while reducing the communication cost. DNN models with a large number of parameters require high communication bandwidth and may become the eventual bottleneck of the whole distributed training. In particular, for distributed acoustic modeling, the issue may become even more severe due to its high communication/computation ratio. First, loading features and labels from storage to memory may affect its training efficiency, given the low bandwidth between the two. A typical way to deal with this is to run data loaders in multiple processes in parallel to pipeline data loading and perform online feature expansion if necessary. In terms of the model/gradient transfer, a broad variety of communication-reduction techniques have been proposed. Gradient compression approaches such as gradient quantization [28], [29] and gradient sparsification [30] are used to reduce the required communication bandwidth. Partial model averaging instead of global model averaging is another way to reduce the communication cost [9], [26].

When training acoustic models under discriminative sequence criteria, additional care must be taken for issues concerning storage, communication, and computation in terms of training efficiency. The hypothesis space in the discriminative objective function is represented by lattices [20], which take up a significant amount of storage space. As a result, data loading is more time-consuming compared to the CE training. On the computation side, the gradient evaluation involves the forward and backward algorithm running on lattices, which typically takes place on CPUs as it is nontrivial to express it in an efficient form of matrix multiplication suitable for GPUs. For large-scale distributed training, shallow lattices of low density are usually used to reduce the required storage space and speed up communication and computation [4].

**Experiments**

In this section, we evaluate various distributed deep acoustic model training strategies on the SWB2000. The first set of experiments are designed to compare their performance in convergence, speedup, and WER for pedagogical purposes.

**Data set**

The SWB2000 is a well-established public benchmark used for ASR evaluation [1]–[3]. The data set consists of 1,975 h
of audio data among which 10 h of audio is used for the heldout set for training. WERs are evaluated on the Hub5 2000 evaluation set which is composed of two parts: One is the 2.1-h SWB data and the other is the 1.6-h call-home (CH) data.

**Model**

The acoustic model is a DNN-HMM with a bidirectional LSTM architecture. There are six LSTM layers and each LSTM layer contains 1,024 cells, with 512 in each direction. On top of the LSTM layers, there is a linear bottleneck layer with 256 hidden units followed by a softmax output layer with 32,000 units corresponding to the CD-HMM states. The LSTM is unrolled 21 frames and trained using nonoverlapping feature subsequences of that length. The input is a 260-dimensional vector consisting of a speaker-adapted acoustic feature based on perceptual linear prediction [31] (40 dimensional), a speaker-embedding vector [32] (100 dimensional), and a logMel feature with its delta and double delta [2] (3 × 40 dimensional). The language model (LM) is trained using publicly available text data and double delta of logMel features on the fly, which overlaps with input features and labels. The data loader generates 21 frame subsequences for unrolled LSTMs and expands the delta and double delta of logMel features on the fly, which overlaps with the gradient evaluation on GPUs. No gradient compression is used in communication.

**Baseline**

We establish the baseline by training the acoustic model using SGD on a single P100 GPU without parallelization. The batch size is 256. Following the input dimensionality and LSTM unroll length, a batch is a tensor of size 260 × 21 × 256. The initial learning rate is 0.1, which is annealed by 1/√2 every epoch after the 10th epoch. The training concludes after 16 epochs. The WERs were 7.5 and 13% on SWB and CH, respectively, one of the best results achieved under CE training on this data set in the speech community. This is a well-tuned training method, where the learning-rate scheduling and batch size are optimized toward the best WERs.

**HPC setting**

We use a cluster of four ×86 servers with two 7-core Intel Xeon E5-2680 2.4-GHz CPUs and 1-TB main memory per server. Each server has four P100 GPUs; therefore, there are 16 GPUs in total. Servers are connected using 100-gigabit/s Ethernet. On each server, the CPU and GPU communicate via a PCI-e Gen3 bus with a 16-gigabytes/s peak bandwidth in each direction. The audio data are first converted to input features and labels in the HDF5 format and stored locally on NVMe SSDs on each server. The connection among learners has a ring topology. In case of allreduce, it is implemented using Nvidia Collective Communications Library (NCCL) [33]. Each learner has two concurrent processes on CPUs to load the input features and labels. The data loader generates 21 frame subsequences for unrolled LSTMs and expands the delta and double delta of logMel features on the fly, which overlaps with the gradient evaluation on GPUs. No gradient compression is used in communication.

**Training strategies**

We implement and compare the following three distributed training strategies:

1. A synchronous centralized (SC)-PSGD realized in a decentralized fashion using allreduce for model averaging (mixing matrix \(T_u\)). This is equivalent to SC-PSGD where the PS operations are replaced with a reduction-then-broadcast allreduce operation.

2. A synchronous decentralized (SD)-PSGD in which each learner averages its model using its left and right neighbors (mixing matrix \(T_l\)). This model-averaging pattern can reduce the communication cost compared to allreduce.

3. An AD-PSGD, where local gradient computations and model updates run concurrently with model averaging using its left and right neighbors (mixing matrix \(T_l\)). Asynchronous centralized PSGD is not included in the experiments as it is known to be difficult to train and gradually loses popularity to other strategies.

Figure 4(a) shows the heldout loss of the distributed strategies using 16 GPUs. The total batch size is 2,560; therefore, each learner has a local batch size of 160. The batch size of 2,560 is determined by the WERs. The initial learning rate is 0.1, the same as that of the baseline. In the first 10 epochs, it linearly warmed up to 1, after which it was annealed by 1/√2 every epoch. All three distributed training strategies converge to a similar loss close to that of the baseline. Their WERs are also close to those of the baseline. Specifically, WERs are 7.6 and 13.1% on SWB and CH, respectively, under SC-PSGD; 7.6 and 13.3% on SWB and CH, respectively, under SD-PSGD; and 7.6 and 13.2% on SWB and CH, respectively, under AD-PSGD.

Conversely, their speedup performance differs significantly, which is presented in Figure 4(b) as a function of the number of GPUs. The synchronous SGD (SC-PSGD and SD-PSGD) has a smaller speedup than the asynchronous SGD (AD-PSGD) due to the learner’s idle time during the synchronization. We also compare the impact of two implementations of allreduce on SC-PSGD: one is the open source message passing interface (SC-PSGD-OpenMPI) allreduce and the other is the NCCL (SC-PSGD-NCCL) allreduce. The latter is a faster allreduce implementation than the former, which improves the speedup for synchronous SGD. Because SD-PSGD uses partial model averaging, it is implemented using OpenMPI. Partial model averaging reduces the communication cost and therefore offers better speedup than that of SC-PSGD-OpenMPI. Among these strategies, the best speedup performance is given by AD-PSGD, which achieves 11-times speedup over 16 GPUs.

One of the advantages of asynchronous SGD over synchronous SGD is its automatic load balancing, which allows a faster learner to consume more computation than a slow learner. To demonstrate this, we design a scenario in which eight of the 16 GPUs in the cluster share running jobs from other tasks, which results in slow learners. Figure 5 shows the distribution of the workload under AD-PSGD across 16 GPUs in terms of the processed minibatches in one epoch. It clearly shows that faster learners pick up higher workloads during training to create faster overall training. Furthermore, asynchronous SGD
can eliminate the “straggler” problem that limits synchronous SGD. To show this effect, we design another scenario in Table 2, where we purposely slow down one GPU learner by two, 10, and 100 times to make it a straggler. As listed in the table, this straggler causes significant prolonged training time in one epoch under synchronous SGD while not affecting asynchronous SGD. AD-PSGD, in this case, delivers consistent speedup.

In the second set of experiments, we attempt to show how the discussed training strategies can help to substantially shorten the training time on the SWB2000 without sacrificing recognition accuracy, especially their scaling-out capability when increasing the number of GPUs. To maximize its parallelization performance, it is critically important to increase the batch size while sustaining a good convergence. It was found that AD-PSGD can tolerate a much larger batch size than can its synchronous centralized counterpart [10].

**HPC setting**

We use an eight-server cluster equipped with 1-TB main memory and eight V100 GPUs on each server. Each server has two 9-core Intel Xeon E5-2697 2.3-GHz CPUs. Between the servers are 100-gigabit/s Ethernet connections. The GPUs and CPUs are connected via a PCI-e Gen3 bus, which has a 16-gigabytes/s peak bandwidth in each direction. To maximize the feasible batch size while effectively reducing the required communication bandwidth, we employ a hierarchical-ring configuration following [10]. In this configuration, GPU learners on the same computing node run SC-PSGD using a local ring by NCCL allreduce. They are called superlearners. All the superlearners then form another ring running AD-PSGD; therefore, it is a hierarchical implementation of synchronous and asynchronous SGD in one configuration.

Table 3 shows the speedup and recognition performance. To make the speedup comparable, we optimize the batch size against the WERS on 64 GPUs and then scale down to 32 and 16 GPUs using the same local batch size on each learner (128 per learner). Therefore, the batch sizes on 16 and 32 GPUs may not be optimal under these two conditions. Although training the LSTM acoustic model on the SWB2000 with a single V100 GPU takes 195 h, it takes 20 h on 16 V100 GPUs, 9.9 h on 32 V100 GPUs, and 5.2 h on 64 V100 GPUs. This is equivalent to roughly a 38-times speedup with similar WERS. To the best of our knowledge, this is the best speedup reported on the SWB2000 with this level of recognition accuracy by the time of submission of this article.
Table 3. The scaling-out performance on the SWB2000 using the hierarchical-ring configuration with various numbers of GPUs.

| GPU           | Batch Size | Training Time (h) | Speedup | WER |
|---------------|------------|-------------------|---------|-----|
| Single V100 GPU | 256        | 195               | −       | 7.5 | 13  |
| 16 V100 GPUs  | 2,048      | 20                | 9.8     | 7.5 | 13.2|
| 32 V100 GPUs  | 4,096      | 9.9               | 19.7    | 7.5 | 13.2|
| 64 V100 GPUs  | 8,192      | 5.2               | 37.5    | 7.6 | 13.2|

Summary
In this article, we discussed the distributed training of DNN acoustic models using minibatch-based data PSGD. We provided an overview of existing distributed training strategies (synchronous versus asynchronous and centralized versus decentralized) used in the speech community and analyzed their advantages and disadvantages. We also studied their convergence and speedup performance on the popular public benchmark SWB2000 data set. For distributed training using data PSGD, a batch size that is sufficiently large is a necessary condition for good speedup. It is dependent on optimization algorithms and a careful design of learning schedules. In addition, handling the interplay between communication and computation is crucial for high-performance distributed training. In practice, we strive for maximum overlap between communication and computation when designing and implementing a distributed training strategy from both algorithmic and HPC architectural perspectives.

Authors
Xiaodong Cui (cuix@us.ibm.com) received his B.S. degree from Shanghai Jiao Tong University, China, in 1996, his M.S. degree from Tsinghua University, Beijing, China, in 1999, and his Ph.D. degree from the University of California, Los Angeles, in 2005, all in electrical engineering. Currently, he is a research staff member at IBM T.J. Watson Research Center, Yorktown Heights, New York, and an adjunct professor in the Department of Electrical Engineering, Columbia University, New York. He joined IBM in 2006 and has contributed to numerous U.S. Government and IBM internal projects on large vocabulary continuous speech recognition, multilingual speech recognition, spoken term detection, and speech-to-speech translation. He has served as an associate editor of IEEE/ACM Transactions on Audio, Speech and Language Processing and is an elected member of the IEEE Speech and Language Technical Committee. His research interests include deep learning in acoustic modeling, multilingual learning, automatic speech recognition, and machine learning. He is a Senior Member of the IEEE.

Wei Zhang (weiz@us.ibm.com) received his B.S. degree from the Beijing University of Technology in 2005, his M.Sc. degree from the Technical University of Denmark, Kongens Lyngby, in 2008, and his Ph.D. degree from the University of Wisconsin–Madison in 2013, all in computer science. Currently, he is a research staff member at the IBM T.J. Watson Research Center, Yorktown Heights, New York. His research interests include parallel programming, large-scale machine learning, and concurrent software reliability. He is an adjunct assistant professor at New York University. His Programming Language Design and Implementation 2011 paper won the ACM Special Interest Group on Programming Languages Research Highlights Award, and his IEEE International Conference on Data Mining 2016 (ICDM 2016) paper won ICDM best research paper runner-up award.

Ulrich Finkler (ufinkler@us.ibm.com) received his undergraduate degree in physics in 1991 and his M.S. degree in physics in 1994 from the University of Saarbrücken, Germany. After two years at the Fraunhofer Institute for Nondestructive Testing, he received his Ph.D. degree in computer science from the Max-Planck-Institute for Informatics, Saarbrücken, Germany, with a scholarship from the German Research Fund in 1997. Since then, he has been with the IBM T.J. Watson Research Center, Yorktown Heights, New York, where he works on very-large-scale integration tools, memory management and parallelization, the acceleration and performance analysis of software applications, and deep learning. He is also an adjunct associate professor at New York University. He is a Senior Member of the IEEE.

George Saon (gsaon@us.ibm.com) received his undergraduate degree from Polytechnic University of Bucharest in 1993, his M.Sc. and Ph.D. degrees in computer science from Henri Poincare University in Nancy, France, in 1994 and 1997, respectively. In 1995, he received his engineering diploma from the Polytechnic University of Bucharest, Romania. Since joining IBM T.J. Watson Research Center, Yorktown Heights, New York, in 1998, he has worked on a variety of problems spanning several areas of large vocabulary continuous speech recognition, such as discriminative feature processing, acoustic modeling, speaker adaptation, and large vocabulary decoding algorithms. He has published more than 100 conference and journal papers and holds several patents in the field of speech recognition. He received a best paper award at both INTERSPEECH 2010 and the 2011 IEEE Automatic Speech Recognition and Understanding Workshop. He has served as an elected member of the IEEE Speech and Language Technical Committee.

Michael Picheny (map22@nyu.com) received his B.S., M.S., and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge. He recently joined New York University in the Courant-Computer Science and the Center for Data Science as a research professor after many years as the senior manager of speech technologies in IBM Research AI at the IBM T.J. Watson Research Center, Yorktown Heights, New York. He has worked in the speech recognition area since 1981, when he joined IBM. He served as an associate editor of IEEE Transactions on Acoustics, Speech, and Signal Processing from 1986 to 1989 and as the chair of the Speech Technical Committee of the IEEE Signal Processing Society from 2002 to 2004. He was a member of the board of the International Speech Communication Association from...
