Research Article

DDLS: Distributed Deep Learning Systems: A Review

Najdavan Abduljawad Kako[1], Subhi R. M. Zeebaree[2], Mohammed A. M.Sadeeq[3]
Ahmed Alkhayyat[4] Hanan M. Shukur[5]

[1]Duhok Polytechnic University,
Duhok, Kurdistan Region, Iraq.
najdavan.kako@dpu.edu.krd
[2]Duhok Polytechnic University,
Duhok, Kurdistan Region, Iraq.
subhi.rafeeq@dpu.edu.krd
[3]Duhok Polytechnic University,
Duhok, Kurdistan Region, Iraq.
mohammed.abdulrazaq@dpu.edu.krd
[4]Islamic University,
Najaf, Iraq.
ahmed.hussien795@gmail.com
[5]Al-Kitab University,
Kirkuk, Iraq.
hanan.m.shukur@uoalkitab.edu.iq

Article History: Received: 10 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 28 April 2021

Abstract: The clustered deep learning systems practice deep neural model networks with a cluster pooled resources aid. Distributed profound learning systems engineers should make multiple choices to process their diverse workloads successfully in their selected environment. Combined with the cluster bandwidth constraints, the abundance of GPU-based deep learning, the ever-greater size of data sets, and deep neural network models would entail developing high-quality models by distributed, profound learning systems designers. Because of their extensive lists of features and architectural deviations, it is not easy to compare distributed deep learning systems side by side. By examining the overall properties of deep learning models and how these workloads can be expanded into a cluster to carry out collective algorithm testing, the fundamental principles at work are shed when training a deep neural network in an isolated machinery cluster. Different techniques been addressed which are used by today's distributed deep learning systems and discuss their consequences. In order to conceptualize and compare deep-level structures, different methods have been developed by previous works to deep-level systems spread DDLS. Indeed, this paper addressed them to be more clearance for the readers.

Keyword: Machine learning, deep learning, distributed systems, gradient descent, big data.

1. Introduction

In recent years, Deep Learning (DL) has made significant progress. In the different fields, DL tools have been used by scientists and engineers to solve their problems, including computer vision (He et al., 2016), natural language processing (Vaswani et al., 2017), voice recognition (Amodei et al., 2016), etc. In DL, an increase in training data sizes usually improves model efficiency (e.g., classification accuracy) (Brownlee, 2019). However, the training method of DL is very computational and thus time-consuming, given the increased data size and model complexity. They are training a ResNet-50 model (in 90 epochs) with the new Nvidia Tesla V100 GPU on the ImageNet data sets, for example (Russakovsky et al., 2015; You et al., 2019). Takes about two days. Generally, one must adjust certain hyper-parameters to reach satisfactory efficiency, which takes much more time. The computer power of a single accelerator can be used entirely utilizing highly tailored program installation to reduce the training time(Wang et al., 2019).

In addition, the most impressive effect is the miniaturization of computer systems with the maybe smartphones alongside the creation of increasingly efficient and connected machine (P. Y. Abdullah et al., 2020; Alzakholi et al., 2020). This machines are all full-service computers packed with sensors, a lot of memory and a powerful CPU.

Naturally, they are often networked (Haji, Ahmad, et al., 2020; Shukur et al., 2020). In the same line plug machines are making their way to the market as well as other so-called microcomputers (Salih et al., 2020). These small computers are often connected directly to a desktop and provide near-desktop output, often in the size of a power adaptor (Z. S. Ageed et al., 2020). As a result, it is now not only possible, but also easy to build a computer infrastructure that consists of a large or small networked computer (Z. Ageed et al., 2020). In general, these machines are globally scattered and thus usually a distributed structure is said to be forming (Yazdeen et al., 2021). The size of a distributed system can range from a few devices to millions. The network may be wired, wireless or a hybrid of the two (Mohammed et al., 2021; M. M. Sadeeq et al., 2021). In addition, distributed networks are also very complex in that computers can enter and leave with almost continually evolving topology and output of the underlying network (Ageed, Zeebaree, Sadeeq, Kak, Yahia, et al., 2021).

Different descriptions for distributed systems, none of them acceptable or in accordance with any other systems have been given in literature (Haji, Zeebaree, et al., 2020; Ibrahim, 2021). It is enough to give us a loose characterization for our purposes: A distributed system is a set of self-contained computational components that are a cohesive system for its users (Abdulrahman et al., 2021). Two characteristics are described in this description of distributed systems. The first is that a distributed system is a set of computer components, each of which may act separately (Kareem et al., 2021; Subhi RM Zeebaree, 2020). A processing element that we usually call a node could be a hardware system or a process of the program (Ageed, Zeebaree, Sadeeq, Kak, Rashid, et al., 2021). A second thing is that consumers (whether they are humans or applications) feel that they have a single device to work with (Ageed, Zeebaree, Sadeeq, Abdulrazzaq, et al., 2021). This implies that the autonomous nodes must work together in one direction or another (Abdulqadir et al., 2021). The development of distributed networks is central to how this relationship will be established (Jijo et al., 2021). Please note that we make no claims about the node form. They can, in theory, range from high-performance mainframe computers to small devices in sensor networks even within a single system (Rashid et al., 2018; Rashid et al., 2019).

Disseminated training (Brown, 2015) is becoming very common, however, to accelerate the training process with many processors such as CPUs (You et al., 2018), GPUs, and Google TPUs (Lin et al., 2017). Many processors working with each task intuitively will minimize the overall training time, but coordination costs between processors usually limit the scalability of systems (Ioffe et al., 2015).

Much worse, multiple transformers could attain lower efficiency compared to a single processor by installing high-speed calculators (for instance, Nvidia V100 GPUs) with low-speed interconnections (for example, a 1/10 Gbps Ethernet) to train a deep model (for example, the calculated to communicate ratio is low) (S. M. S. A. Abdullah et al., 2021). Therefore, data communication in the distributed training of deep models should be designed to maximize the computational resources of distributed clusters (M. A. Sadeeq et al., 2021). The examination parts can be as follows:

Different distributed deep learning techniques are extensively studied, providing readers with motivations and principles underlying different design decisions and their effect on model training.

Some distributed deep learning systems vary markedly; others just slightly. In this area, we review a selection of major works (both science and commercial) and organize them depending on this review.

We separate our review-based methodology from related works in Section 2. The main components at work in a DDLS are addressed, rather than addressing different current DDLS in Section 3, clarifying how they connect, and including some advice on selecting an effective technique. We summarize our review for future studies in Section 4.

2. Literature Review

Several separate current works have been compared to distributed deep learning systems. A sequential review of recent advancements in deep learning, with several improvements in DDLS, is given (Saeed et al., 2021). This review, opposed to this approach, focuses exclusively on the distributed issue domain and is arranged by subject rather than chronologically (Obaid et al., 2020; K. Zhang et al., 2017). Demonstrate how complex deployment choices impact the preparation and bandwidth of networks in several distributed networks(Dino et al., 2020). More honest is our review. The principles that support the multiple training approaches been understood widely used in distributed deep learning and inevitably contribute to variations in application choices. We are therefore looking for a wider variety of explaining parallel and distributed deep learning algorithms that have recently been released (Ben-Nun & Hoefler, 2019), which begins with a tutorial on general concepts such as backpropagation, model architectures, and supervised learning algorithms, beforehand turning to parallel and distributed training related to architecture search and hyper-parameter. We believe that the researcher accepts the basics of deep learning and its best approaches in a cluster framework to train DL models (Subhi R Zeebaree et al., 2020). Therefore, it provides additional evidence on approaches for distributed training rather than relying on the scope (D. Q. Zeebaree et al., 2017). The review-based approach attempts to organize and separate the problem domain offers an in-depth overview of the multiple implementation options for distributed deep learning systems training and discusses the intuitions that underpin them (here, the differentiation in the workings of centralized optimization and decentralized optimization under different scheduling administrations) is compromised. In addition, we also present subjects and highlight recent studies that have not been adequately discussed but are of high interest to practitioners (asynchronous systems and their impact of staleness and reduction). Also, for the distributed and reproducible benchmarking system for deep learning, a software infrastructure was introduced to utilize the most powerful supercomputers for extreme-scale workloads (Ben-Nun, Besta, et al., 2019; Gao et al., 2019; Mattson et al., 2019).

There are similar works to distributed deep learning in another field of study. However, it was illustrated differently, such as neurocomputing (Najdavan A Kako, 2013), security (Najdavan A Kako, 2018; Najdavan Abduljawad Kako et al., 2020), parallel distributed processing in cloud computing (Subhi RM Zeebaree, 2020), and other artificial intelligence (Abdulazeez et al., 2021; Sadeeq et al., 2017; Subhi R Zeebaree et al., 2020).

Our goal is to provide an overview of a variety of distributed deep learning systems concepts and methods. To do this, an analysis been built by integrating the fundamental features that have a significant effect on how DDLS works. Real DDLS may be viewed as specializations of more general terms through recognizing the intuitions and values underpinning these characteristics. These ideas will be discussed with the limits of distributed atmospheres in mind. The implementation of our review gives an unhindered view of current works in this area and requires observations to be made based on basic design choices. In addition, quite recently, distributed deep learning has appeared. Several terms and divisions of the problem domain are used in current literature in this area. The review attempts to unify the definitions so that such phenomena can be precisely related to particular features or design choices. the review is divided into four parts:

1) Data-model parallelism.

2) Centralized versus decentralized optimization.

3) Synchronous versus asynchronous scheduling.

4) The exchanging parameters communication pattern.

A. Data-model parallelism

In both data parallelism and model parallelism, some constraints apply. For data parallelism, if there are so many processing nodes, there is a need to reduce the learning rate to ensure a smooth training phase. For model parallelism, if there are so many nodes, the network efficiency would be significantly reduced for the sake of connectivity cost. With many neuron operations, model parallelism could achieve good efficiency, and data-parallel is successful with many weights. In CNNs, the convolution layer includes around 90 percent of the computation and 5 percent of the parameters.

In comparison, 95 percent of the parameters and five percent-10 percent of the computation are found in the complete linked layer. Therefore, by utilizing data parallelism for the convolution layer and model parallelism for a completely linked layer, it can be parallelized the CNNs in data model mode (Buyya et al., 2016). The two parallelisms are:

Model Parallelism (MP): Parallelism of the model means the responsibility of a computer node by training the same data sample for parts of the model. The model is split into many components, and each computer node, like GPU, has a single part of it. The correspondence takes place between computer nodes where a neuron input is from the other computer node output. Parallel model performance is often smaller than parallel data since the parallel model's connectivity costs are much greater than parallel data.

Data Parallelism (DP): It is possible to apply data parallelism easily and is thus the most frequently used multi-GPU architectural technique.

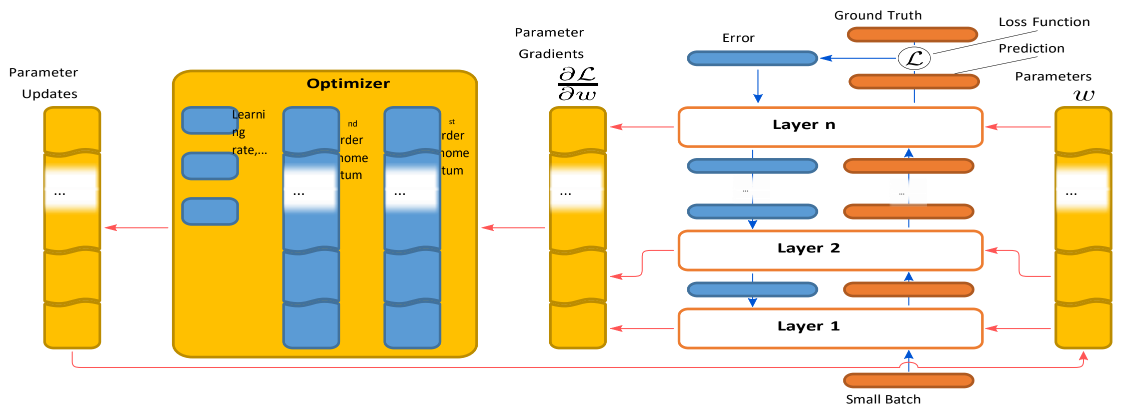

Figure 1: Deep learning models and their data flow cycles during training

Parallel data means that all GPUs use the same model for traveling on different data sub-sets. There is no synchronization between GPUs in parallel data transmission as each GPU has a complete model reproduction, including its deep network configuration and parameters. Nevertheless, the parameter gradients computed from various GPUs in BP must be synchronized.

B. Centralized versus Decentralized Optimization

Deep learning models and their data flow cycles during training illustrated in figure 1. It is possible to break the training procedure into two separate cycles. By applying the BP algorithm on small batches taken from outcomes, the method calculates per-parameter grounded on the existing model. To evaluate model parameter changes, the optimization period absorbs these gradients. Although this can sound like a bidirectional reliance, it is necessary to remember that the model cycle may turn out several predictions to improve gradient outputs provided a collection of parameters. At the same time, the optimizer requires modified gradients to advance. There are two significant forms to map this implementation model onto a cluster of independent machines:

1) Centralized optimization: On a central machine, the optimization step is carried out, whereas the gradient processing code is repeated on the residual cluster nodes.

2) Decentralized optimization: In each cluster node, all cycles are repeated, and some synchronization is understood that enables the different optimizers to work helpfully.

C. Synchronous versus Asynchronous Scheduling

It is even necessary to classify DDLS through synchronous, asynchronous, and restricted asynchronous structures. Computations for all workers arise concurrently in bulk synchronous (or merely synchronous) structures. Global obstacles to convergence mean that specific worker nodes cannot proceed before the remaining workers enter a similar state. Asynchronous systems yield a more passive attitude to coordinating collective preparation and stop sidestepping workers' execution to please other workers. In other terms, synchronous systems accomplish successful collective training by eliminating variation in development between workers at the cost of a probable under-usage of resources. In contrast, asynchronous systems encourage a high use of hardware for more training and variation between workers as a controllable side effect that can also be beneficial in some circumstances.

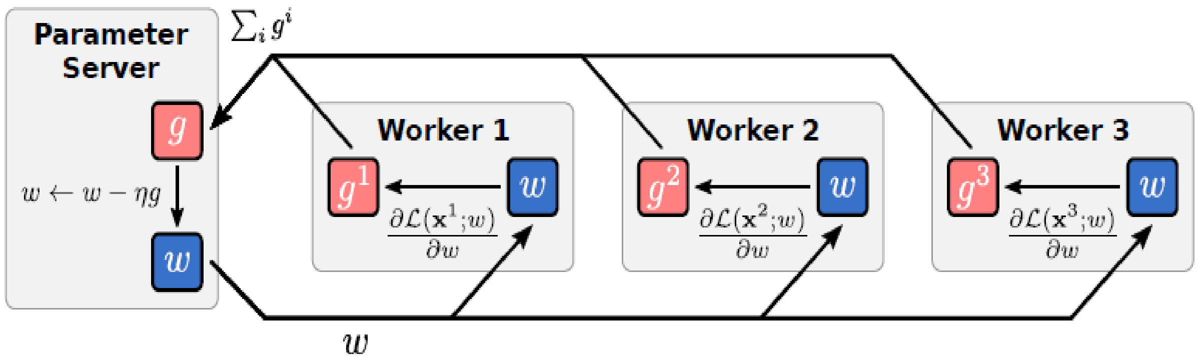
Parameter Server
$\sum_i g^i$
g
$w \leftarrow w - \eta g$
w
Worker 1
g^1
w
$\frac{\partial \mathcal{L}(\mathbf{x}^1;w)}{\partial w}$
Worker 2
g^2
w
$\frac{\partial \mathcal{L}(\mathbf{x}^2;w)}{\partial w}$
Worker 3
g^3
w
$\frac{\partial \mathcal{L}(\mathbf{x}^3;w)}{\partial w}$
w

Figure 1: The top is the flow of data in a synchronous DDLS, and the bottom path is minimizing the application of the parameter server and worker programs.

Bounded asynchronous structures define a blended approach of these two archetypes. They run similar to unified asynchronous networks but implement laws to satisfy worker advancing at different rates. Therefore, workers work asynchronously concerning each other, but only under those boundaries (W. Dai et al., 2015). Their conversation is separated accordingly because of the various motivations and consequences of creating a hyper propagation model of synchronous or asynchronous modes of service with centralized and decentralized.

Centralized Synchronous Systems CSS: The training of those systems is divided amongst the gradient computation and the parameter servers. Suppose such a method shows the asynchronous mode of process. In that case, training cannot continue even a complete exchange of parameters among the server of parameters and its worker because the server of parameters relies on the gradient feedback to the new version of the model (J. J. Dai et al., 2019; Feng et al., 2016; Iandola et al., 2016). In exchange, the workers depend on the revised model to further examine the loss function. Thus, the cluster as an entire transition cyclically among phases of unified synchronous DDLS, during which all workers execute the same operation (Tandon et al., 2017).

Decentralized Synchronous Systems DSS: in synchronous distributed deep learning systems, which focus on clustered optimization, independently perform model training in individual worker and thus do not swap parameters for more model training, but rather propagate with the rest of the cluster the independent results of each worker to be able to rate descent trajectories with reasonable accuracy. They thus work in stages separated by global barriers to synchronization.

Centralized Asynchronous Systems CAS: Distributed deep learning systems of those systems, each worker, once a small batch has been processed, a parameter server operates alone, while still sharing its gradients with the server (Abadi et al., 2016; Keskar et al., 2016). The parameter server enthusiastically inserts obtained gradients into the optimization algorithm to train the model instead of waiting for other workers to enter the same condition. Thus, each modification of the global model is only based on a single worker's gradient data (Chen et al., 2015). This is close to the processes of willing aggregation. However, instead of removing all residual workers' outcomes and wasting the computational capital expended, each worker can merely continue to use its nearby cached old version of the model parameters (Dean et al., 2012; Zhao et al., 2019).

Bounded Asynchronous Systems BAS: In practice, the equal scheduling inferred in our study is impractical since faster computers will be pushed back by the slowest computer, which is precisely the condition that asynchronous systems strive and prevent. However, when sharing criteria, not implementing any order transfers some harm. Gradients from severely eclipsed workers will confuse the optimizer of the parameter server, which can set back testing or even ruin model training.

Decentralized Asynchronous Systems DAS: Dissimilar in DAS, where model training of all workers sometimes ceases to re-parameterize the local models at a global synchronization barrier, workers behave independently in DAS and begin to discover the loss function of this model that is disconnected from the current state of the server (Feyzmahdavian et al., 2016; S. Zhang, 2016). Consequently, upon completing a parameter swap with the server node, the worker cannot substitute their model parameters. Alternatively, they need to combine the respective asynchronously collected results. The Preference form for combining server $(\widetilde{w})$ and worker models (w^i) in such a setting is to apply linear interpolation (Kim et al., 2016; Lian et al., 2015).

D. Parameters Communication Pattern

Different machine performs each function in the cluster. However, computing programs running on one computer, different machines, or distributed over several machines are worker, parameter servers, and server nodes. Distributing the worker function is model parallelism. The concentration focused on communication patterns used by parallel data distributed deep learning systems to coordinate the transfers of clusters and accelerate parameters.

Communication patterns in centralized systems: Having the parameter server task based on a single computer, irrespective of the training process, vastly simplifies the system design since whole model training is organized in one program. Besides, it is simple to configure, monitor, and debug such systems. This might, however, render the parameter server a bottleneck.

Communication patterns in decentralized systems: Discrete training phases of loops assume the coordination demand of each clustered worker per local measurement stage. In this way, decentralized networks usually maintain better use of computer hardware, even with minimal network capacity, making it easier to train massive models despite bandwidth restrictions. Scaling to greater cluster sizes, however, will also cause the server node to become a bottleneck. In decentralized systems, it is possible to break the server's position to decrease coordination costs, as in centralized systems. However, autonomous DDLS provide several choices for arranging parameter transfers since each system is a self-contained autonomous trainer.

3. Evaluation

Many of the distributed deep learning practices are focused on the promising results obtained by Google's large-scale image classification (Y. Zhang et al., 2020) utilizing clusters of huge nodes of CPU using Disbelief. A centralized asynchronous distributed deep learning system with multiparameter server support (Dean et al., 2012). Several distributed deep learning systems have been suggested for expanding the Disbelief approach. For example, the imposition of delay limits to monitor the staleness of asynchronous workers has increased the convergence pace (Xing et al., 2015). In order to create multiple parallelisms and integrate with a general ML model, the emphasis was on formalizing the processing of DL workloads. A related approach was taken by transferring gradient processing measures for certain neural network layers into the parameter servers (Chilimbi et al., 2014).

Furthermore, in a PAXOS cluster, they arrange the parameter servers to achieve high availability. The current descendants of Disbelief implement new ideas such as specifying stand-by staff, improving model partitioning utilizing auto-tuning models, and reducing expansion by enabling parameter servers to be organized hierarchically (Abadi et al., 2016; Chen et al., 2015). Systems in deep learning with COTS HPC (Goyal et al., 2017) and Fire Caffe on compute clusters are reduced distributed deep learning systems operations that obtain convinced concepts from Disbelief. However, it has been used in supercomputer environments, particularly HPC and GPU, where they have proven to be an ideal tool for many specific applications in terms of performance. By integrating unified synchronous parallel data in model training on top of Apache Spark, a simple convergence of data analysis platforms and the hardware environment was used. They use specialized communication fashions to incorporate a parameters distribution registry to meet the everyday communication needs of such systems. The deep learning with Python data-parallel optimizer follows a similar method but introduces a traditional interface to use collective connectivity primitives to perform synchronous model training. Either one or more worker’s function as the server parameter server. The majority of distributed deep learning systems that depend on decentralized optimization stem from training deep networks in Spark (You et al., 2019). This decentralized synchronous distributed deep learning system replicates Caffe-solvers to conduct training in commodity cluster environments using the map-reduce API of Apache Spark. This technique has achieved widespread acceptance as part of the popular Java distributed deep learning systems. This technique also sees the constraint on synchronous execution as a significant downside. A worker that attempts to improve upon training DNN in Spark (distributed deep learning training models) (Lin et al., 2017). Extending the simple Spark-based approach to quasi-asynchronous training by overlapping computation and connectivity and extrapolating the recently found descent route to deal with staleness, and DL with elastic averaging SGD, that preserves the principle of small isolated phases of training but implements entirely asynchronous scheduling. If the cluster gets bigger, having one server node will reach deadlock (Lian et al., 2017). Also, many approaches are used to scale out the decentralized optimization (Blot et al., 2016; Huang et al., 2018; S. Zhang, 2016). Table 1 displays all systems for DDL, and Table 2 can support researchers to find the choice when applying or selecting a distributed deep learning system.

TABLE 1: Comparison and overview for different distributed deep learning systems.

DDLS Name (a-z)	**Parallelism** (Model / Data)	**Optimization**	**Scheduling**	**Parameter Exchange**	**Topology**	**Achieved Objective and Significant Results**
BigDL (J. J. Dai et al., 2019)	DP	central	synchronous	scatter-red.	DPS (always $k = n$)	For 1/n of the model, each worker works as a parameter server. Via the Spark block manager, distributed parameter exchanges are carried out.
CaffeOnSpark (Feng et al., 2016)	DP	central	synchronous	scatter-red.	DPS (always $k = n$)	Exchange of parameters by RDMA using repetitive MPI function invocations. There are available equivalent implementations for CAFFE2 and CHAINER.
COTS HPC (Coates et al., 2013)	MP	central	synchronous	–	Dist. array abstraction	Model clusters divided along tensor dimensions & spread among the cluster. The access may be fine-grained with a low-level array definition.
D-PSGD (Lian et al., 2017)	DP	decentral	synchronous	2:1 reduce	closed ring	Each node only communicates parameters with its neighboring nodes on the ring.
DistBelief (Dean et al., 2012)	MP and DP	central	asynchronous	ad hoc	DPS	Dedicated parameter server nodes spread by model divisions
EASGD (S. Zhang et al., 2014)	DP	decentral	asynchronous	ad hoc	single server	Active variation of hyper-parameters can accelerate training (Decentralized asynchronous system)
FireCaffe (Iandola et al., 2016)	DP	central	synchronous	binomial tree	SPS	centralized synchronous system in a simple manner
GoSGD (Blot et al., 2016)	DP	decentral	soft-bounded asynchronous	ad hoc	P2P mesh	There is no dedicated server node, and when two workers realized via a sum-weighted randomized gossip protocol, the parameter will exchange
MPCA-SGD (Lin et al.,	DP	decentral	soft-bounded	binomial	dedicated server	Model and data may be modified separately.

2017)			asynchronous	tree	node	Punishment is a component of the loss function in the model. The extrapolation mechanism is used to reduce the staleness impact.
MXNet (Chen et al., 2015)	MP and DP	central	bounded asynchronous	scatter-reduce async.: ad hoc	DPS (default $k = n$)	Supports numerous technical parameter server configurations in comparison to hierarchical multi-stage proxy servers.
Parameter Server (W. Dai et al., 2015)	MP and DP	central	bounded asynchronous	reduce async.: ad hoc	DPS	Model partitions propagation through parameter groups is redundant. Model parallelism enables to organize workers in groups, and one worker can act as a proxy server per group.
Petuum (Xing et al., 2015)	MP and DP	central	bounded asynchronous	ad hoc with eager scatter	DPS	Uses pause bounds for the first time to regulate staleness. The prompt distribution of model parameters further decreases model variance.
Project Adam (Chilimbi et al., 2014)	MP and DP	central	asynchronous	ad hoc	DPS	It has a dedicated parameter server group that is operated as a PAXOS cluster. Hybrid parallelism takes advantage of gradient computing by transforming computation into GPUs.
PyTorch (Paszke et al., 2019)	MP and DP	central	synchronous	all-reduce	SPS or replicated PS	Capabilities of model parallelism in version 1.4 can just use either synchronous data-parallelism or model parallelism.
SparkNet (Moritz et al., 2015)	DP	decentral	synchronous	reduce	dedicated server node	Decentralized synchronous implementation is understood using Spark map-cut. Production grade reimplementation is existing in deeplearning4J.
TensorFlow (Abadi et al., 2016)	MP and DP	central	bounded asynchronous	scatter/all-red. async.: ad	DPS (default	Technologies supporting single and multiparameter servers and all-reduce-

				hoc	$k = n)$	related methods are studied.
TreeEASGD (Langer et al., 2020)	DP	decentral	bounded asynchronous	ad hoc	tree	The nodes of the tree are workers. Workers on immediate up and downstream neighbors can only exchanges parameters

Data Parallelism (DP); Model Parallelism (MP); Distributed Parameter Server (DSP)

TABLE 2: Criteria for choosing a distributed deep learning system approach.

	RLBD	RBM	opt LR	COCC	CAHP	RSOI	ERB
Model Parallelism		✓	higher	1+	3+	1+	trivial
Model Parallelism and mini-batch pipelining		✓	lower	2+	4+	2+	medium
Data Parallelism, central and sync.			higher	1+	1+	1+	easy
Data Parallelism, central and async.	✓		lower	3+	2+	3+	hard
Data Parallelism, decentral, and sync.	✓		likely lower	1+	2+	2+	easy

4. Conclusion & Future Work

In this review, we discussed different theoretical and functional aspects that may emerge in the training of DL models in a cluster and presented reasonable surmises that enable thinking about how different methods use the tools available to the association. The underlying concepts of architecture that have a guiding effect were structured. Our review defines a simple scheme that enables the main sources of distributed deep learning to be separated. In order to obtain a compact description, we then extended this analysis to classify a set of distributed deep learning systems irrespective of application-specific particularities. Some possible fields of future study related that may be an important part of the study for the IoT or vehicle applications to use P2P model sharing combined with decentralized optimization strategies, as suggested by (Blot et al., 2016).

The majority of work in distributed deep learning is restricted to optimal research scenarios. The condition is typically more complicated in real cluster configurations because of overlapping workloads. For certain clinicians, a systematic study of multiple distributed methods in real-life situations will be useful. Lin et al. (Lin et al., 2017) proved that the clusters are mostly not substituted but rather expanded as investment commodities. A largely un-tackled innovation challenge is the successful realization of distributed instruction in heterogeneous setups.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). *Tensorflow: A system for large-scale machine learning.* Paper presented at the 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16).
2. Abdulazeez, A. M., Hajy, D. M., Zeebaree, D. Q., & Zebari, D. A. (2021). Robust watermarking scheme based LWT and SVD using artificial bee colony optimization. *Indonesian Journal of Electrical Engineering and Computer Science*, *21*(2), 1218-1229.

3. Abdullah, P. Y., Zeebaree, S. R., Jacksi, K., & Zeabri, R. R. (2020). An hrm system for small and medium enterprises (sme) s based on cloud computing technology. *International Journal of Research-GRANTHAALAYAH, 8*(8), 56-64.
4. Abdullah, S. M. S. A., Ameen, S. Y. A., Sadeeq, M. A., & Zeebaree, S. (2021). Multimodal emotion recognition using deep learning. *Journal of Applied Science and Technology Trends, 2*(02), 52-58.
5. Abdulqadir, H. R., Zeebaree, S. R., Shukur, H. M., Sadeeq, M. M., Salim, B. W., Salih, A. A., et al. (2021). A Study of Moving from Cloud Computing to Fog Computing. *Qubahan Academic Journal, 1*(2), 60-70.
6. Abdulrahman, L. M., Zeebaree, S. R., Kak, S. F., Sadeeq, M. A., Adel, A.-Z., Salim, B. W., et al. (2021). A state of art for smart gateways issues and modification. *Asian Journal of Research in Computer Science*, 1-13.
7. Ageed, Z., Mahmood, M. R., Sadeeq, M., Abdulrazzaq, M. B., & Dino, H. (2020). Cloud computing resources impacts on heavy-load parallel processing approaches. *IOSR Journal of Computer Engineering (IOSR-JCE), 22*(3), 30-41.
8. Ageed, Z. S., Ibrahim, R. K., & Sadeeq, M. A. (2020). Unified Ontology Implementation of Cloud Computing for Distributed Systems. *Current Journal of Applied Science and Technology*, 82-97.
9. Ageed, Z. S., Zeebaree, S. R., Sadeeq, M. A., Abdulrazzaq, M. B., Salim, B. W., Salih, A. A., et al. (2021). A State of Art Survey for Intelligent Energy Monitoring Systems. *Asian Journal of Research in Computer Science*, 46-61.
10. Ageed, Z. S., Zeebaree, S. R., Sadeeq, M. M., Kak, S. F., Rashid, Z. N., Salih, A. A., et al. (2021). A Survey of Data Mining Implementation in Smart City Applications. *Qubahan Academic Journal*, *1*(2), 91-99.
11. Ageed, Z. S., Zeebaree, S. R., Sadeeq, M. M., Kak, S. F., Yahia, H. S., Mahmood, M. R., et al. (2021). Comprehensive survey of big data mining approaches in cloud systems. *Qubahan Academic Journal, 1*(2), 29-38.
12. Alzakholi, O., Shukur, H., Zebari, R., Abas, S., & Sadeeq, M. (2020). Comparison among cloud technologies and cloud performance. *Journal of Applied Science and Technology Trends, 1*(2), 40-47.
13. Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., et al. (2016). *Deep speech 2: End-to-end speech recognition in english and mandarin.* Paper presented at the International conference on machine learning.
14. Ben-Nun, T., Besta, M., Huber, S., Ziogas, A. N., Peter, D., & Hoefler, T. (2019). *A modular benchmarking infrastructure for high-performance and reproducible deep learning.* Paper presented at the 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS).
15. Ben-Nun, T., & Hoefler, T. (2019). Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR), 52*(4), 1-43.
16. Blot, M., Picard, D., Cord, M., & Thome, N. (2016). Gossip training for deep learning. *arXiv preprint arXiv:1611.09726*.
17. Brown, L. (2015). Gpu accelerated deep learning with cudnn. *GTC.*
18. Brownlee, J. (2019). Impact of dataset size on deep learning model skill and performance estimates. *Machine Learning Mastery, 6*.
19. Buyya, R., Calheiros, R. N., & Dastjerdi, A. V. (2016). *Big data: principles and paradigms*: Morgan Kaufmann.
20. Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., et al. (2015). Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*.
21. Chilimbi, T., Suzue, Y., Apacible, J., & Kalyanaraman, K. (2014). *Project adam: Building an efficient and scalable deep learning training system.* Paper presented at the 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14).
22. Coates, A., Huval, B., Wang, T., Wu, D., Catanzaro, B., & Andrew, N. (2013). *Deep learning with COTS HPC systems.* Paper presented at the International conference on machine learning.
23. Dai, J. J., Wang, Y., Qiu, X., Ding, D., Zhang, Y., Wang, Y., et al. (2019). *Bigdl: A distributed deep learning framework for big data.* Paper presented at the Proceedings of the ACM Symposium on Cloud Computing.
24. Dai, W., Kumar, A., Wei, J., Ho, Q., Gibson, G., & Xing, E. (2015). *High-performance distributed ML at scale through parameter server consistency models.* Paper presented at the Proceedings of the AAAI Conference on Artificial Intelligence.
25. Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., et al. (2012). Large scale distributed deep networks. *Google Research*.

26. Dino, H., Abdulrazzaq, M. B., Zeebaree, S. R., Sallow, A. B., Zebari, R. R., Shukur, H. M., et al. (2020). Facial Expression Recognition based on Hybrid Feature Extraction Techniques with Different Classifiers. *TEST Engineering & Management, 83*, 22319-22329.
27. Feng, A., Shi, J., & Jain, M. (2016). CaffeOnSpark open sourced for distributed deep learning on big data clusters: Feb.
28. Feyzmahdavian, H. R., Aytekin, A., & Johansson, M. (2016). An asynchronous mini-batch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control, 61*(12), 3740-3754.
29. Gao, W., Tang, F., Wang, L., Zhan, J., Lan, C., Luo, C., et al. (2019). AIBench: an industry standard internet service AI benchmark suite. *arXiv preprint arXiv:1908.08998*.
30. Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., et al. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.
31. Haji, L. M., Ahmad, O. M., Zeebaree, S. R., Dino, H. I., Zebari, R. R., & Shukur, H. M. (2020). Impact of cloud computing and internet of things on the future internet. *Technology Reports of Kansai University, 62*(5), 2179-2190.
32. Haji, L. M., Zeebaree, S., Ahmed, O. M., Sallow, A. B., Jacksi, K., & Zeabri, R. R. (2020). Dynamic resource allocation for distributed systems and cloud computing. *TEST Engineering & Management, 83*, 22417-22426.
33. He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition.* Paper presented at the Proceedings of the IEEE conference on computer vision and pattern recognition.
34. Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, M. X., Chen, D., et al. (2018). Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*.
35. Iandola, F. N., Moskewicz, M. W., Ashraf, K., & Keutzer, K. (2016). *Firecaffe: near-linear acceleration of deep neural network training on compute clusters.* Paper presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.
36. Ibrahim, I. M. (2021). Task scheduling algorithms in cloud computing: A review. *Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12*(4), 1041-1053.
37. Ioffe, S., & Szegedy, C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* Paper presented at the International conference on machine learning.
38. Jijo, B. T., Zeebaree, S. R., Zebari, R. R., Sadeeq, M. A., Sallow, A. B., Mohsin, S., et al. (2021). A Comprehensive Survey of 5G mm-Wave Technology Design Challenges. *Asian Journal of Research in Computer Science*, 1-20.
39. Kako, N. A. (2013). *An Investigation of the Coefficient of Variation Using the Colored Stochastic Hodgkin-Huxley Equations.* Eastern Mediterranean University (EMU)-Doğu Akdeniz Üniversitesi (DAÜ).
40. Kako, N. A. (2018). *Classical Cryptography for Kurdish Language.* Paper presented at the 4th International Engineering Conference on Developments in Civil & Computer Engineering Applications (IEC2018).
41. Kako, N. A., Sadeeq, H. T., & Abrahim, A. R. (2020). New symmetric key cipher capable of digraph to single letter conversion utilizing binary system. *Indonesian Journal of Electrical Engineering and Computer Science, 18*(2), 1028-1034.
42. Kareem, F. Q., Zeebaree, S. R., Dino, H. I., Sadeeq, M. A., Rashid, Z. N., Hasan, D. A., et al. (2021). A survey of optical fiber communications: challenges and processing time influences. *Asian Journal of Research in Computer Science*, 48-58.
43. Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.
44. Kim, H., Park, J., Jang, J., & Yoon, S. (2016). Deepspark: Spark-based deep learning supporting asynchronous updates and caffe compatibility. *arXiv preprint arXiv:1602.08191, 3*.
45. Langer, M., He, Z., Rahayu, W., & Xue, Y. (2020). Distributed training of deep learning models: A taxonomic perspective. *IEEE Transactions on Parallel and Distributed Systems, 31*(12), 2802-2818.
46. Lian, X., Huang, Y., Li, Y., & Liu, J. (2015). Asynchronous parallel stochastic gradient for nonconvex optimization. *arXiv preprint arXiv:1506.08272*.
47. Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., & Liu, J. (2017). Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *arXiv preprint arXiv:1705.09056*.
48. Lin, Y., Han, S., Mao, H., Wang, Y., & Dally, W. J. (2017). Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*.
49. Mattson, P., Cheng, C., Coleman, C., Diamos, G., Micikevicius, P., Patterson, D., et al. (2019). Mlperf training benchmark. *arXiv preprint arXiv:1910.01500*.

50. Mohammed, C. M., & Zebaree, S. R. (2021). Sufficient comparison among cloud computing services: IaaS, PaaS, and SaaS: A review. *International Journal of Science and Business, 5*(2), 17-30.
51. Moritz, P., Nishihara, R., Stoica, I., & Jordan, M. I. (2015). Sparknet: Training deep networks in spark. *arXiv preprint arXiv:1511.06051*.
52. Obaid, K. B., Zeebaree, S., & Ahmed, O. M. (2020). Deep Learning Models Based on Image Classification: A Review. *International Journal of Science and Business, 4*(11), 75-81.
53. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.
54. Rashid, Z. N., Zebari, S. R., Sharif, K. H., & Jacksi, K. (2018). *Distributed cloud computing and distributed parallel computing: A review.* Paper presented at the 2018 International Conference on Advanced Science and Engineering (ICOASE).
55. Rashid, Z. N., Zeebaree, S. R., & Shengul, A. (2019). *Design and analysis of proposed remote controlling distributed parallel computing system over the cloud.* Paper presented at the 2019 International Conference on Advanced Science and Engineering (ICOASE).
56. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision, 115*(3), 211-252.
57. Sadeeq, H., Abdulazeez, A., Kako, N., & Abrahim, A. (2017). *A Novel Hybrid Bird Mating Optimizer with Differential Evolution for Engineering Design Optimization Problems.* Paper presented at the International Conference of Reliable Information and Communication Technology.
58. Sadeeq, M. A., & Zeebaree, S. (2021). Energy management for internet of things via distributed systems. *Journal of Applied Science and Technology Trends, 2*(02), 59-71.
59. Sadeeq, M. M., Abdulkareem, N. M., Zeebaree, S. R., Ahmed, D. M., Sami, A. S., & Zebari, R. R. (2021). IoT and Cloud computing issues, challenges and opportunities: A review. *Qubahan Academic Journal, 1*(2), 1-7.
60. Saeed, J., & Zeebaree, S. (2021). Skin Lesion Classification Based on Deep Convolutional Neural Networks Architectures. *Journal of Applied Science and Technology Trends, 2*(01), 41-51.
61. Salih, A. A., Zeebaree, S. R., Abdulraheem, A. S., Zebari, R. R., Sadeeq, M. A., & Ahmed, O. M. (2020). Evolution of Mobile Wireless Communication to 5G Revolution. *Technology Reports of Kansai University, 62*(5), 2139-2151.
62. Shukur, H., Zeebaree, S. R., Ahmed, A. J., Zebari, R. R., Ahmed, O., Tahir, B. S. A., et al. (2020). A State of Art Survey for Concurrent Computation and Clustering of Parallel Computing for Distributed Systems. *Journal of Applied Science and Technology Trends, 1*(4), 148-154.
63. Tandon, R., Lei, Q., Dimakis, A. G., & Karampatziakis, N. (2017). *Gradient coding: Avoiding stragglers in distributed learning.* Paper presented at the International Conference on Machine Learning.
64. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.
65. Wang, Y., Wang, Q., Shi, S., He, X., Tang, Z., Zhao, K., et al. (2019). Benchmarking the performance and power of AI accelerators for AI training. *arXiv preprint arXiv:1909.06842*.
66. Xing, E. P., Ho, Q., Dai, W., Kim, J. K., Wei, J., Lee, S., et al. (2015). Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data, 1*(2), 49-67.
67. Yazdeen, A. A., Zeebaree, S. R., Sadeeq, M. M., Kak, S. F., Ahmed, O. M., & Zebari, R. R. (2021). FPGA implementations for data encryption and decryption via concurrent and parallel computation: A review. *Qubahan Academic Journal, 1*(2), 8-16.
68. You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., et al. (2019). Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*.
69. You, Y., Zhang, Z., Hsieh, C.-J., Demmel, J., & Keutzer, K. (2018). *Imagenet training in minutes.* Paper presented at the Proceedings of the 47th International Conference on Parallel Processing.
70. Zeebaree, D. Q., Haron, H., Abdulazeez, A. M., & Zeebaree, S. R. (2017). Combination of K-means clustering with Genetic Algorithm: A review. *International Journal of Applied Engineering Research, 12*(24), 14238-14245.
71. Zeebaree, S. R. (2020). *Remote Controlling Distributed Parallel Computing System over the Cloud (RCDPCSC).* Paper presented at the 2020 3rd International Conference on Engineering Technology and its Applications (IICETA).
72. Zeebaree, S. R., Ahmed, O., & Obid, K. (2020). *CSAERNet: An Efficient Deep Learning Architecture for Image Classification.* Paper presented at the 2020 3rd International Conference on Engineering Technology and its Applications (IICETA).

73. Zhang, K., Alqahtani, S., & Demirbas, M. (2017). *A comparison of distributed machine learning platforms.* Paper presented at the 2017 26th International Conference on Computer Communication and Networks (ICCCN).
74. Zhang, S. (2016). *Distributed stochastic optimization for deep learning.* Ph. D. thesis (New York University, New York).
75. Zhang, S., Choromanska, A., & LeCun, Y. (2014). Deep learning with elastic averaging SGD. *arXiv preprint arXiv:1412.6651*.
76. Zhang, Y., Wang, Y., Liu, X.-Y., Mi, S., & Zhang, M.-L. (2020). Large-scale multi-label classification using unknown streaming images. *Pattern Recognition, 99*, 107100.
77. Zhao, X., An, A., Liu, J., & Chen, B. X. (2019). *Dynamic stale synchronous parallel distributed training for deep learning.* Paper presented at the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS).