Distributed analytics for big data: A survey

Distributed Analytics for Big Data: A Survey

Francesco Berloco[a], Vitoantonio Bevilacqua[a,b,*], Simona Colucci[a]

[a] *Department of Electrical and Information Engineering (DEI), Politecnico di Bari, Via Orabona, 4, Bari, 70125, Italy,*
[b] *Apulian Bioengineering, Via delle Violette, 14, Modugno, 70026, BA, Italy*

Abstract

In recent years, a constant and fast information growing has characterized digital applications in the majority of real-life scenarios. Thus, a new information asset, namely Big Data, has been defined and lead to different challenges, mainly related to data storage, management and analysis. Focusing on the last challenge, several Big Data analytics techniques have been developed, based on Machine Learning and Deep Learning paradigms. When dealing with Big Data, traditional approaches often take a lot of time to produce even a single predictive model, due to the extremely high demand of computational resources.

The design of approaches specifically oriented to Big Data is required to overcome these computational issues. Most solutions rely on the deployment of Big Data analytics infrastructures on a cluster of machines and/or on parallelization techniques. When deployment and parallelization apply to Machine Learning and Deep Learning, we can refer to the terms Distributed Machine Learning and Distributed Deep Learning, respectively.

We here discuss the main principles and features of Distributed Machine Learning and Distributed Deep Learning frameworks. The main contribution of this work is a survey of solutions proposed in the literature, through the investigation of selected features and capabilities. In particular, the survey provides a comparative analysis according to the following classification criteria: implemented parallelization technique, supporting device, supported architecture, implemented communication mode, working mode, and class of algorithms.

*Corresponding author, e-mail address: vitoantonio.bevilacqua@poliba.it (V. Bevilacqua)

Postprint submitted to Neurocomputing

The paper also gives an overview of the most commonly used criteria and metrics for the performance evaluation of analysed frameworks; finally, some emerging but promising optimization techniques are reviewed apart from our classification.

Keywords: Big Data, Parallelization Algorithms, Distributed Machine Learning, Distributed Deep Learning

---

1. Introduction

In recent years, the term "Big Data" (BD) quickly spread among IT researchers and professionals. Although many different definitions for BD are available, the one probably summing up all them is proposed by De Mauro et al. [1]: "*Big Data represents the Information asset characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value*".

The concepts involved in this definition refer to: the properties of managed information ("Volume"), the requirements for processing this information ("Velocity and Variety"), and its economic "Value", that is one of the major impact factor of BD in the society. The definition also emphasizes the specificity for BD of technology and methods to be employed in the solution of heterogeneous analytical goals.

Big Data analytics include three major branches: statistical analysis (descriptive and inferential), *Machine Learning* (ML), and *Deep Learning* (DL) techniques. Concerning the last two branches, several algorithms have been developed, to achieve different goals in a variety of application scenarios [2, 3, 4, 5, 6, 7, 8, 9, 10].

ML and DL play a crucial role in big data analytics, since they provide different techniques for extracting meaningful information from huge volumes of data, making predictions, processing unstructured data, recognizing and detecting patterns and anomalies [11]. In this context, several challenges arise related to the inherent nature of BD and/or to the kind of analysis performed. We here survey relevant approaches addressing two of such challenges:

- **Data Size Management**: handling BD, especially in the training phase of ML and DL models, is extremely challenging, due to the huge data volume.

- **Model Complexity Management**: training and validation of non-linear models open new, specific, challenges, due to the model complexity.

The most successful solutions addressing both challenges rely on the distribution of computation across different machines, by using parallelization techniques and algorithms. In particular, clusters of machines embedding so-called "commodity hardware" seem to be the leading technological solution in BD context. On the one hand, parallelization of tasks reduces the computational load on a single machine; in this way, also complex ML/DL models can be trained with low-performance machines. On the other hand, parallelization introduces a complexity overhead in the overall management of different training tasks, also in terms of network management and communication.

When parallelization is applied to ML (respectively, DL), we can refer to *Distributed Machine Learning*–DML (respectively, *Distributed Deep Learning*–DDL). Both DML and DDL are subfields of *Distributed Artifial Intelligence* (DAI) with specific peculiarities related to model training and inference: parameters update sharing, computation of the gradient descent, model and data distribution, just to name a few.

This survey aims at investigating solutions related to BD analytics, focusing on distributed computation algorithms for ML and DL. In particular, it classifies 13 DML and 20 DDL frameworks according to the following features: implemented parallelization technique, supporting device, supported architecture, implemented communication mode, working mode and class of algorithms. It extends a previous work [12], focused only on DDL for BD, by (i) extending the research methodology, (ii) investigating in depth distributed computation algorithms; (iii) including DML approaches; (iv) including algorithms for the optimization of computation.

The rest of the work is structured as follow. After giving an overview of related work in Section 2, we present in Section 3 the applied research methodology. Section 4 defines the basic terminology used in the paper. Section 5 introduces the background knowledge at the basis of our classification of DML and DDL frameworks for Big Data, presented in Section 6. Section 7 reports the most used criteria and metrics for the evaluation of DML/DDL frameworks, while Section8 collects some relevant emerging optimization techniques, whose implementation is not evaluated in the classification. A conclusive section sums up the results of this study.

2. Related Works

In this section we compare our work to main published review and survey papers on BD analytics, DDL and DML.

To the best of our knowledge, related papers deal either with BD analytics methods and frameworks or with DML/DDL. As explained in our previous work [12], no other paper compares the analytical capabilities of different DML/DDL frameworks, proposing a standard methodology.

The work by Gupta and Rani [13] clarifies the definition of BD and presents some open-source BD processing frameworks. It also gives an overview of the major future research challenges about BD.

Among works focused on distributed solutions, the work by Ben Nun and Hoefler [14] aims at demystifying (as the title suggests) the mechanisms behind DDL. In other words, the paper reviews the technical insights of selected DDL frameworks, in terms of parallelization techniques underlying model training and inference, gradient descent computation, and communication.

Xing *et al.* [15] try to answer the following questions "How to distribute an ML program over a cluster? How to bridge ML computation with inter-machine communication? How to perform such communication? What should be communicated between machines?", by explaining the statistical analyses and algorithms behind ML and giving to the readers different principles and design tips for the development of a DML software.

Verbraeken *et al.* [16] provide, probably, the most comprehensive review. In fact, they inspect several DML aspects including systems architectures, ML algorithms, cluster topology, machines communication.

Other works focus on different aspects of DML and DDL, such as communication, optimization, applications. The work by Qiu *et al.* [17] reviews the main ML methods for BD, with a specific focus on signal processing techniques.

Otoo-Arthur and Van Zyl [18] investigate the role of BD analytics in high education field, referring to it as "Big Educational Data Analytics"; they found out that few researches tried to integrate BD and learning analytics in higher education.

Inoubli *et al.* [19] gave a survey on existing BD frameworks, studying their application in data storage, analytic and bath/stream processing.

Zhang *et al.* [20] investigated the architectural design behind the DML platforms, explaining how the different choices affect the performance of such

platforms; they chose Spark, PMLS, TensorFlow and MXNet as case studies.

Finally, ML model training acceleration has been widely investigated by Wang *et al.* ([21]); this survey gives an overview of acceleration and optimization techniques, keeping an acceptable model accuracy.

3. Research Methodology

Our review work follows the so-called Systematic Literature Review (SLR) approach, a protocol for literature search defined by Brereton *et al.* [22]. The SLR process follows three phases, detailed in the following subsections.

3.1. Plan review

In this step, the Research Questions (RQ)s leading the entire work are formulated and presented in Table 1.

Table 1: List of Research Questions

	RQs	Goal
1	What are the main parallelization techniques used in DML and DDL?	To explain the background behind the parallelization techniques, showing advantages and drawbacks.
2	What are the different communication methods among different workers? What kind of related architectures can be implemented?	To explain the communication strategies, the cluster architectures and the strategy for data exchange among workers(*e.g.*, partial updates, synchronization flags, partial predictions, partial gradients and so on).
3	What are the frameworks available (with related capabilities) in BD, DML and DDL contexts?	To give an overview of the available frameworks in this context and classify them according to the previous answers.

Our review protocol starts by querying the popular database SCOPUS with the following 5 topics: 1) "Big Data", 2) "Machine Learning", 3)"Deep

Learning", 4)"Big Data, Machine Learning, Deep Learning" and 5)"Distributed Artificial Intelligence for Big Data". The exact queries posted to SCOPUS are given in Appendix A, for the interested reader.

We also report in Appendix B, the number (Table B.11) and temporal trends (Figure B.7) of retrieved documents for each query. The analysis of such results lead us to focus on the less investigated topic: solutions involving BD and DAI at the same time (Appendix B, Table B.11, topic 5).

We immediately point out that, by addressing topic 5, we consider only DML and DDL; other subfields of DAI (such as federated learning) are mentioned but not examined thoroughly[1]. The temporal trend of researches related to topic 5 is depicted in Figure 1.

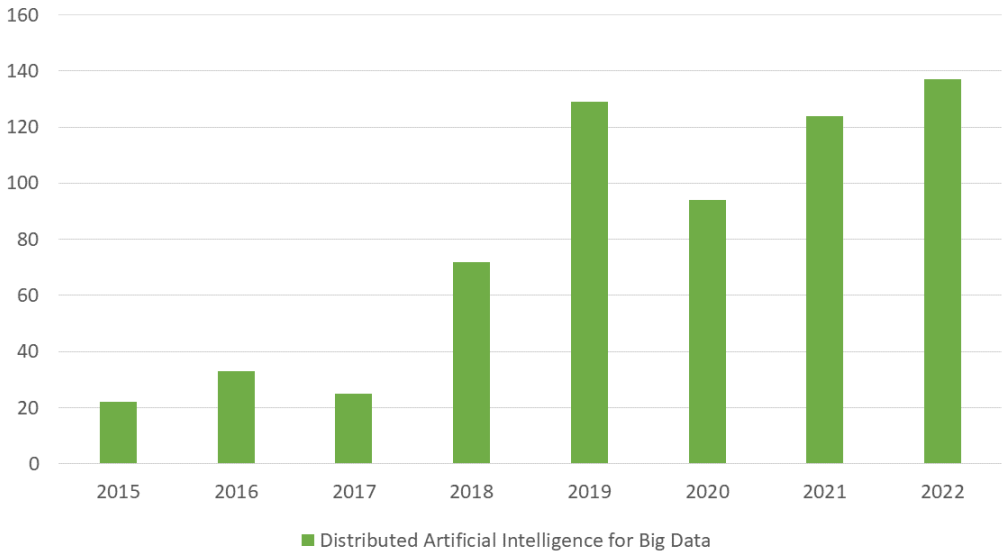

Figure 1: Trends of researches related to Distributed Artificial Intelligence for Big Data.

Due to the high number of retrieved articles, the adoption of a set of Inclusion/Exclusion criteria (IC/EC) is necessary. Interested readers can find more details about IC/EC in Appendix A

[1]More details about DAI classification may be found in Section 4.2.1

3.2. Conduct Review

We performed a literature research by querying SCOPUS, according to the topics in our review protocol, with a time interval ranging from 2015 to 2022.

Table B.11 shows that 636 works are retrieved with Query 5. We further filtered this list by DOI in order to remove the duplicates. The RQs in Section 3.1 and the IC defined in Appendix A have been used to filter the list of papers by title, then by abstract and, finally, by content. In this way, the list of papers has been further reduced. The reduction process is summarized and schematized in Figure 2.

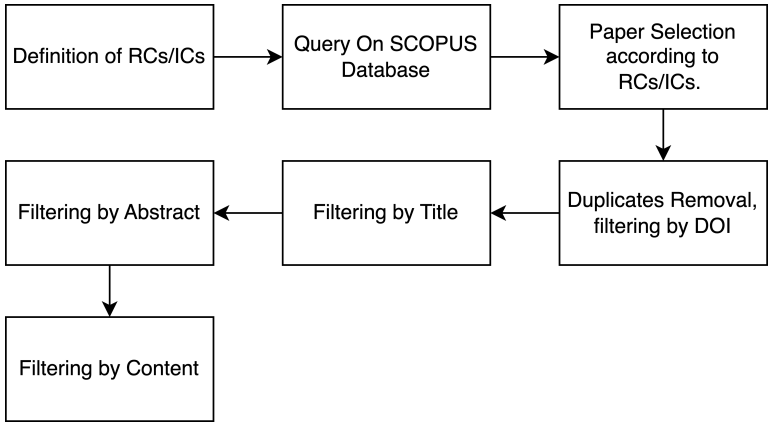

Figure 2: Paper selection workflow.

3.3. Document Review

The thorough study of selected papers lead us to identify 6 classification criteria and 13 DML and 20 DDL frameworks. Some emerging optimization techniques have been kept out of the classification features and detailed in Section 8.

We list below the identified classification criteria for DML/DDL frameworks:

1. **Parallelization Technique**: the implemented parallelization method (namely, Model, Data, or hybrid parallelization);
2. **Supporting Device**: the hardware resource supporting computation (*e.g.*, Central Processing Unit - CPU, Graphics Processing Unit - GPU, and Tensor Processing Unit - TPU);

3. **Supported Architecture**: the kind of deployment designed for the framework: centralized or decentralized architecture;
4. **Communication Mode**: the method adopted for communication, *i.e.* Synchronous or Asynchronous;
5. **Working Mode**: the framework behaviour in terms of capability to work in standalone mode, with an external back-end, or to integrate other frameworks;
6. **Class of Algorithms**: branch of learning algorithms at the basis of distribution, *i.e.* Machine Learning or Deep Learning.

The basic knowledge to understand such classification features is given in Section 4 and 5, before systematically comparing the selected frameworks according to them (Section 6).

4. Background Knowledge

In this section, we provide the basic terminology to understand the classification in this systematic review.

4.1. Big Data

Although the term Big Data has become common among professional and researchers, its definition is still ambiguous. In fact, there are several definitions, each focused on a different peculiarity, but almost all based on the *3Vs* paradigm [1]: Volume, Velocity, Variety. Thus, defining BD means speaking about the *3Vs*.

Volume refers to the enormous data quantity processed and generated by systems; Hamdaoui *et al.* [23] estimate that the number of data generated every day is about 2.5 quantillion (2.5×10^{18}) and it will reach 175 zettabytes (1.75×10^{23}) by 2025. The International Data Corporation estimates that approximately 90 zettabytes of this data will be generated only by IoT devices [24]. The ASKAP (Australian Square Kilometre Array Pathfinder) processes about 7.5 terabytes per second, and this number will reach approximately 750 terabytes by 2025; moreover in 2018 Facebook had about 1.45 billion daily active users [13]. Finally, according to *Internet Live Stats*, Google processed over 99,000 searches every second in 2022; this means that the average search number is more than 8.5 billion in a single day.

The term *Velocity* refers to the analysis speed, that is essential for the usefulness of results, in several applications requiring timeliness. This kind

of information processing is known by the term *on-line analysis* and asks for specific time restrictions. Conversely, when the time is not a key requirement for the analysis outcome, the so called *off-line* analysis is performed.

By *Variety*, BD definitions mean the need of processing different data types at the same time in the same environment: heterogeneous data, *e.g.*, audio, video, text and images, have to be managed as a whole.

The Cancer Genome Atlas (TCGA) dataset represents an evidence of BD Volume and Variety, with its over 2.5 petabytes of proteomic, transcriptomic, epigenomic, genomic and imaging data.

Some definitions refer to other two V-properties of BD: the *Value* generated by such data [25] and their *Veracity*: the grade of reliability associated to data. Other secondary aspects (*e.g.*, *Validity*, *Visualization*, *Vulnerability*, *Fine-Grained* and so on) are summed up by the work of Al-Mekhlal and Ali Khwaja [26].

4.2. Machine Learning and Deep Learning

Machine Learning [27, 28] is a branch of Artificial Intelligence aiming at building computational models able to *learn* patterns from data [29] and make predictions. ML techniques can be classified in three main categories, according to the kind of analysis performed, to the input used for model training and to the analysis target:

- **Supervised Learning**: produces models able to map an observed fact in input to a learned output; in this scenario, training is performed according to a labelled dataset, given as input;
- **Unsupervised Learning**: produces models able to find patterns in unlabelled data.
- **Reinforcement Learning**: produces models interacting with the environment to accomplish a specific task; the model training is performed according to a rewards-penalties rules.

ML is a widely studied research field, for which several models and techniques have been developed [30], such as Linear Regression, Logistic Regression, Support Vector Machine (SVM), Naive Bayes, Decision Tree based models, Principal Component Analysis (PCA), t-distributed Stochastic Neighbor Embedding (t-SNE), Clustering algorithms, and Artificial Neural Networks

(ANN). Due to their promising performance, such models are widely employed to accomplish different tasks [31, 32, 33, 34, 35, 36, 37, 38, 39].

Deep Learning [40] is a field of Machine Learning, referring to complex ANN-based models (Deep Neural Networks, DNNs) built and employed for hierarchical feature extraction and raw data processing. Deep Learning algorithms are useful for BD analytics tasks where the design of handcrafted algorithms is challenging, such as image processing, bioinformatics, computer vision, natural language processing (NLP), speech recognition, among others [41, 42, 43, 44, 45, 46, 47, 48, 49, 50].

The complexity of DL models concerns the number of layers and parameters. A relevant example is given by AlexNet [51], a Convolutional Neural Network (CNN) [40], with 8 deep layers and 62.3 million of learnable parameters. CNNs belong to a class of ANNs using convolution operations for processing data with grid-like topology. Other well known DL architectures are Restricted Boltzmann Machine (RBM), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), Deep Belief Networks, Bidirectional Recurrent Neural Networks (BRNN), generative models (*e.g.*, Generative Adversarial Networks - GANs and diffusion models [52]) and Transformers (for NLP tasks). In recent years, Transformer based models [53] have been also employed in computer vision and image processing fields [54].

4.2.1. DML and DDL

Distributed Machine Learning refers to a set of techniques and algorithms used to distribute the learning processes (data preparation, data exploration, model training and inference) across different nodes in a distributed environment.

Similarly to DML, Distributed Deep Learning deals with the distribution of DL processes across multiple nodes. Although the distribution principles are similar to DML, in such a scenario the overall management is more complex due to the architecture of the models employed (*e.g.*, number of neurons, number of hidden layers, connections between neurons) and to the kind of datasets (*e.g.*, a heterogeneous dataset analyzed through multimodal models).

The process distribution takes place through parallelization techniques, that can be applied at Single-Machine level and Multiple-Machine (Cluster) level. In the first case, it is possible to distribute and schedule different tasks by adopting multiprocessing algorithms, if the machine uses two or more processing units. Using specific frameworks it is possible to parallelize ML

or DL tasks across multiple CPUs, GPUs and TPUs on the same machine, speeding-up the process or handling complex models. On the other hand, training complex model is a computationally intensive task, especially with BD; in this situation it is common to face two main issues:

1. The computational resources are not sufficient to handle the learning process, due to the limited machine memory;
2. The training process cannot be completed in a reasonable time period, owing to model complexity and/or data used for training.

In these situations, the Cluster-Level parallelization comes into play. With this method, the overall computational load is distributed among different machines connected to the same network, reducing the computational load on the single machine. However, this approach makes the overall processes more complex. In fact, in this scenario there are different aspects that cannot be ignored, such as the parallelization management at single machine level and cluster level at the same time, the network topology, the network limitations (*e.g.*, bandwidth and latency), the communication management.

Moreover, due to the heterogeneity of cluster machines in distributed environments, different workers process data at different speed. This introduces two main problems. In synchronous communication strategy the slowest workers (known as *Stragglers*) slow down the entire system. To overcome this issue, an asynchronous strategy can be adopted, but the global model parameters updating can be affected by *out-of-date* parameters, pushed by slowest workers; intuitively, this problem (called *Gradient Staleness*) leads to a worsening of performance [55].

5. DML/DDL: key enabling techniques and algorithms

We here provide all the technical knowledge needed to understand the classification. In particular: Section 5.1 introduces main parallelization techniques, Section 5.2 deals with the communication architecture, and Section 5.3 explains how the communication and the synchronization among workers is performed.

A final discussion about advantages and potential application scenarios of the introduced techniques and algorithms is given in Section 5.4.

Some emerging optimizations in distributed learning are collected in Section 8 and kept out of our classification criteria, because rarely applied up to now.

5.1. Parallelization Techniques

The training process can be parallelized using different techniques. In this section we examine the main ones, *i.e.*, Data Parallelism, Model Parallelism and Hybrid Parallelism.

Data Parallelism The Data Parallelism algorithm [56] aims at handling the data size problem during the learning phase, by partitioning data in different sub-sets. In particular, in a cluster with N machines, data are partitioned in N batches and the model is replicated N times. Then, one batch and model are sent to each machine and the training of each model replica is performed locally, using the specific batch. The resulting model is the aggregation of the model replicas; the techniques for training the general model are discussed in Section 5.3. The Data Parallelism schema is shown in the left part of Figure 3. Although handling data size issues, this approach introduces different drawbacks, mostly related to computation of the SGD: in fact, not all training processes are performed in the same period, especially for heterogeneous cluster, not guaranteeing its convergence. Moreover, each machine holds model parameter updates that must be shared between nodes, increasing the network traffic. Finally, as explained by Ben-Nun and Hoefler [14], a trade-off is required between *utilization* (hardware efficiency) and *generalization* (statistical accuracy) of the model. This trade-off depends on the batch size: the increase of the batch size maximize the utilization, while the convergence of SGD is not guaranteed with a large batch size (or it can be very slow). For this reason, the choice of the batch size can be challenging.

Model Parallelism Model Parallelism aims at distributing the model layers across multiple devices (that can be CPUs, GPUs, TPUs or FPGA). After setting a number of model parts (usually corresponding to the number of available devices), each sub-set of model layers is sent to a specific device and the training is performed sequentially for all of them. In this way, the memory needed by each work is reduced, because the model is split among multiple workers; this can be useful for the training of heavy models. On the other hand, since each device needs to exchange information with the others, the overall performance depends on the communication efficiency. In a naive implementation, only one device at a time is active, while the others are idle. With this

approach, the model parallelism does not lead to a training speed-up. To overcome this problem the so-called *Pipeline Parallelism* [57] has been introduced, consisting in injecting multiple batches in the model pipeline at the same time.

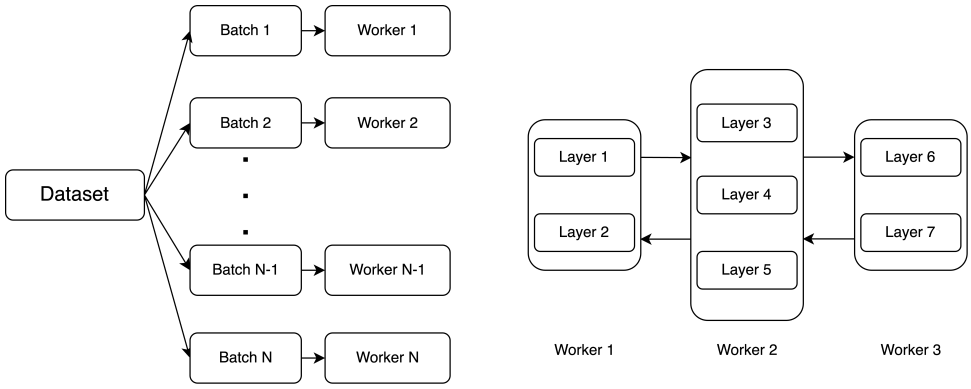

Figure 3: Data Parallelism for N workers (left) and Model Parallelism for three workers (right). In the first approach each worker holds also a global model replica. In the latter, the model training is performed sequentially, according to the model layers distribution among workers.

Pipeline Parallelism. implementation can follow two main paradigms: the *inter-batch* or *intra-batch* pipeline parallelism; both of them use different aspects of Data Parallelism and Model Parallelism, but they should not be confused with *Hybrid Parallelism*, examined in the next section.

- **Intra-batch pipeline** Let $\mathbf{D} = \{D_0, D_1,...,D_k\}$ be the set of devices holding a part of model sub-layers, with $k \geq 2$ and let $\mathbf{B} = \{B_0, B_1,...,B_n\}$ be a finite number of data micro-batches, with $n > 1$. $\mathbf{B_0}$ is processed by $\mathbf{D_0}$ and then the output of $\mathbf{D_0}$ is passed to $\mathbf{D_1}$ while $\mathbf{D_0}$ start to process $\mathbf{B_1}$. The output of $\mathbf{D_1}$ is passed to $\mathbf{D_2}$, while $\mathbf{D_1}$ receives the output of $\mathbf{D_1}$, that start to work on $\mathbf{B_2}$ and so on. When the last mini-batch has been processed in the last devices, the Backward Propagation (BWP) is performed in the same way as the Forward Propagation (FWP). In this way, all weights in all layers in all devices are up to date, mitigating the problem of stale weights. The two main frameworks implementing this approach are FPDeep [58] and GPipe [59]. The first one

proposes a framework for distributed training on FPGA clusters or cloud, together with a workload partitioning schema. The latter further splits each mini-batch into micro-batches, reducing the memory cost on each device; nevetheless, this leads to a worsening of performance because the FWP is performed at micro-batch level.

- **Inter-batch pipeline** The outstanding example of Inter-batch pipeline implementation is PipeDream [60]. Let $\mathbf{D} = \{D_0, D_1,...,D_k\}$ be the set of devices holding a part of model sub-layers with $k \geq 2$, and let $\mathbf{B} = \{B_0, B_1,...,B_n\}$ be a finite number of data mini-batches, with $n > 1$. In this scenario, each device processes a mini-batch sequentially but, differently from the intra-batch approach, the BWP starts when the $\mathbf{D_k}$ terminates the process of $\mathbf{B_0}$. Then $\mathbf{D_{k-1}}$ performs the BWP related to $\mathbf{B_0}$ while $\mathbf{D_k}$ processes $\mathbf{B_1}$ and so on.

The two approaches are shown in the **Figure 4**

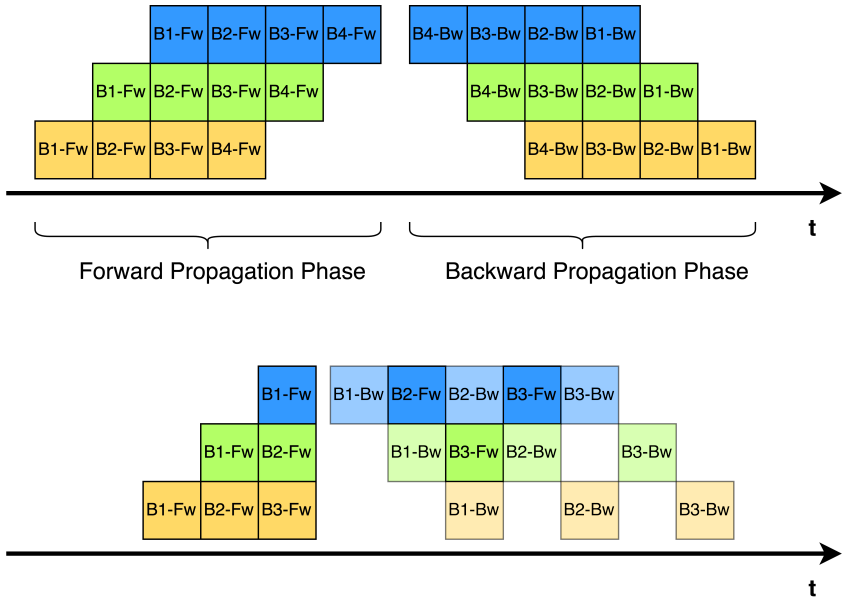

Figure 4: Intra-batch pipeline (GPipe [59], on top) and Inter-Batch Pipeline (PipeDream [60], on bottom). Each color refers to a different worker. *Fw* and *Bw* stand for Forward and Backward phase, while the number indicates the number of mini-batches or micro-batches (for Intra-batch) processed in forward or backward phase. t indicates time.

In this scenario, the batch size is fixed, so there is no need for utilization-generalization trade-off. Moreover, the Model Parallelism does not always lead to a better overall performance, because splitting the model while guaranteeing the load balance between machines is really challenging. In fact, the layers of a Deep Neural Network (DNN) are strictly dependent (*e.g.*, Multilayer Perceptron, CNNs and so on), and the communication during FWP and BWP for weight sharing is complex; thus, the overall management complexity generally increases.

Hybrid Parallelism *Hybrid Parallelism* consists in the application of Data and Model parallelism to the same training task, in order to overcome the drawbacks related to each of the two approaches. Two examples of hybrid parallelism are given by DistBelief [61] and Krizhevsky [62]. The first one adopts all of the three approaches: the training is performed by replicating the model multiple times; then each replica is split and trained simultaneously on different data batch. The second one, instead, works on the distribution of a CNN; in particular it proposes to apply data parallelism to the convolutional and pooling layers, while model parallelism is applied to the fully-connected layers.

The choice among possible parallelization techniques depends mostly on two factors: the addressed task and the overall management complexity. Specifically, the task is the type of analysis to be performed on a dataset. If the task requires the use of non-linear heavy models (e.g., CNNs, Large Transformers, or DNNs in general), the choice should fall on a model parallelism-like strategy. In fact, this strategy allows for splitting the model among multiple workers, thus lightening the workload and reducing the memory requirements for each single worker. Specifically, in case of clusters with heterogeneous hardware, data parallelism-like strategies are not recommended, since the computation time varies too much (several seconds to several minutes for each epoch) among workers, depending on the accelerator hardware used for computation; notably, the situation can get worst if some synchronous communication strategy is adopted (See Section 5.3 for more details). Alternatively, if the task does not require the use of complex models and the volume of data is huge (e.g., Big Data), the adoption of data parallelism-like strategies is the only reasonable choice. In fact, the difference in terms of computation time between workers with heterogeneous hardware is minimized. Obviously, in such a situation, the model parallelism does not bring any advantage and it would be unnecessary and wasteful. Apparently, the best

implementation choice is Hybrid Parallelism since it allows for handling big data and complex models simultaneously, addressing the drawbacks of both Data and Model parallelism approaches. Yet, this parallelization strategy introduces significant complexity in terms of cluster resource management and communication between nodes. Moreover, the application of model and data parallelization allows for accomplishing most big data analytics tasks. For this reason, in most DML and DDL frameworks, only data and model parallelism strategies are available as default options, leaving to the developer the chance to revert on hybrid strategies implementation.

5.2. Communication Architecture

During the learning process, each device needs to communicate with the other ones in order to exchange information crucial to the success of the learning process. To accomplish this task, two main configuration can be setup: centralized and decentralized.

Centralized Architecture For centralized network architectures and in data parallelism scenario, the choice of the configuration generally falls on *Parameter Server*(PS) [63]. The PS is an entity in charge for computing the global gradient during the training process. The gradient and weights update computation is divided in two main steps: Reduce and Broadcast. In the simplest scenario, each worker, holding a replica of the model, computes the local gradient and sends it to the PS; the PS computes the average gradient (reduce) and broadcasts the result (broadcast). Then all workers update their model weights. A simple representation of PS architecture is shown in Figure 5.

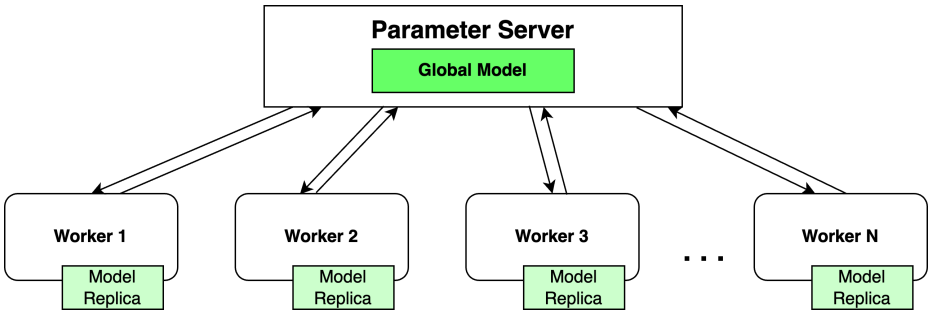

Figure 5: Parameter Server Architecture.

Notably, the PS is a logical concept, subject to different possible implementations. For instance, the Hierarchical PS (HPS) [64] is composed by multiple actors (HBM-PS, MEM-PS and SSD-PS), sharing a main memory. Moreover, for large environments, there can be several PSs communicating with each other and each PS is responsible for a sub-set of worker nodes. Sun *et al.* [65] implemented a PS based on Distributed Shared Memory (DSM). The implemented Distributed PS can reduce networking time by about 50%, and achieves up to 1.9× performance compared to Petuum [66]. Instead, Song *et al.* [67] implemented a Disk-Resident Parameter Server (DRPS) to reduce the hardware requirements for large-machine learning tasks. Specifically, in this implementation, the PS and the worker can be hosted on the same machine. In this case, workers and servers communicate using threads, instead of sending messages across the network; in this way the resources of each machine are fully exploited. Moreover, a lightweight key-value database is used as KV-Store; in particular, the KV-Stores running on servers store model parameters, while the KV-Stores running on workers are responsible for datasets management. This approach reduces the computational load on the single worker; also the network load is reduced with respect to AllReduce technique (Section 5.2). Nevertheless, the overall performance is conditioned by possible bottlenecks in the network.

Decentralized Architecture The decentralized architecture aims at distributing the communication among different workers without a central entity, causing the resulting network to have a peer-to-peer (P2P) topology. In such a network, the basic implementation is the *AllReduce* algorithm, shown in Figure 6: i) all workers are connected to each other and each one of them computes the weight updates locally and independently; ii) each worker exchanges a subset of it updates, to support the aggregation.; iii) the aggregation results are shared among them. The kind of information exchanged depends on the implementation and may vary from partial updates, to model parameters or partial aggregations.

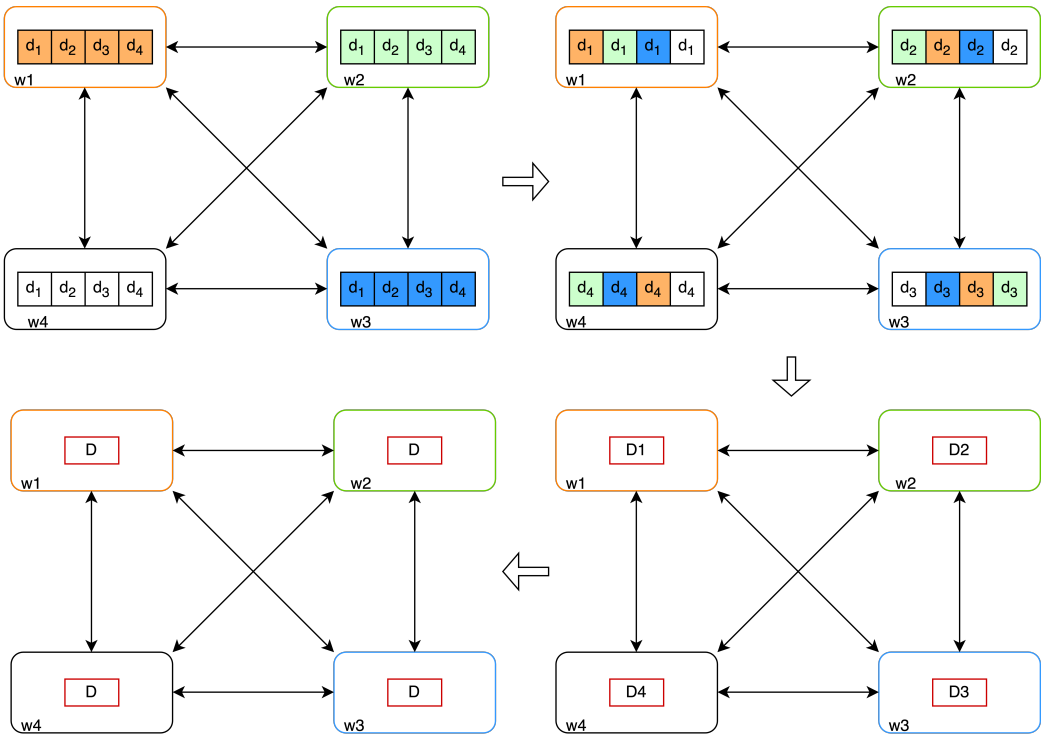

Figure 6: AllReduce Architecture. Each color indicates a different worker; d is the generic subset of data; D_n represent the first aggregation performed by each worker; D is the overall aggregation of all D_n.

In this scenario, the absence of a central computational entity avoids a possible network bottleneck. However, the network traffic increaseas, due to the message exchange between all nodes. There are several implementations of AllReduce, all focusing on an optimization of the algorithm in terms of latency and/or bandwidth. For instance, in a tree network topology: nodes are only connected with a node at the higher level; the reduce phase is performed going up until the root node is reached ; the Broadcast phase propagates from the root node to the leaves [68]. Another optimization is the *Ring AllReduce*, in which a node exchanges information only with its neighbours in a cyclical way, creating a P2P ring topology. The parameter exchange is performed until each node contains all the information needed for the aggregation operation. Then, the aggregation is performed and the results are

shared between all workers. Jiang *et al.* [69] implemented a two dimension hierarchical ring-based AllReduce algorithm to alleviate the latency and accelerate the synchronization process.

Another variant is the Tree-based AllReduce [70, 71], in which trees are used to perform Reduce and Broadcast phases, in order to optimize the latency and the bandwidth for small and large messages, respectively. Other two similar implementation are the*Round-Robin AllReduce* and the *Butterfly AllReduce* [72]. Liu *et al.* implemented the *AllReduce-Switch* architecture [73], supporting the in-network AllReduce aggregation, maintaining the number of workers independent of the bandwidth employed.

It is worth noting that the aggregation operation can be performed using different functions, such as sum, average, maximum and so on [74].

Another factor to be taken into account is the speed of computation algorithms. Lian *et al.* [75] tried to answer the following question: "*Can decentralized algorithms be faster than its centralized counterpart?*", choosing the Parallel Stochastic Gradient Descent (PSGD) algorithm as case study. In this work they evaluate four SGD implementations:

- CNTK [76] standard implementation of synchronous SGD.
- Centralized synchronous SGD using Message Passing Interface (MPI).
- Decentralized SGD using MPI within CNTK.
- Elastic Averaging SGD (EASGD) [77] implementation of Torch.

on different clusters. The study demonstrates that the D-PSGD can achieve the same convergence rate as the centralized one, outperforming this one by avoiding the traffic jam.

The optimal choice between the centralized and decentralized architecture depends on several factors mostly related to the network (i.e., topology, bandwidth, latency, and so on) and cluster hardware and configuration. A centralized architecture with a PS is a reasonable choice for clusters with a star network topology, where the network hub corresponds to the PS. However, since the PS is responsible for computing the global gradient update for all workers, the hardware resources should be able to handle such a computation, minimizing the time. Moreover, network traffic is a significant aspect,

as all data packets are channeled towards a single entity, potentially causing network bottlenecks. Reliability also plays a fundamental role: if a failure occurs at the PS level, the computation cannot be completed as the workers cannot communicate directly with each other. Therefore, recovery and redundancy policies should be adopted, burdening the management complexity. On the other hand, a decentralized approach is a suitable choice for P2P networks without the presence of a central server. In this scenario, where workers communicate directly with each other, the adoption of optimization strategies (e.g., Ring AllReduce) is essential as they reduce network traffic towards a single node. In terms of reliability, if one or more nodes fail, the computation can be easily completed by the remaining nodes. However, node management is more complex compared to centralized approaches. In fact, in the latter scenario only the PS must be aware of the presence of all cluster nodes, while in decentralized approaches all nodes must know about the presence of all the others; this introduces more complexity related to scalability. Considering all these aspects, the adoption and implementation of a PS are only reasonable if there is an existing centralized network architecture with a powerful central server. This choice is justified by the ability to configure a new environment for distributed analytics computation without changing the underlying architecture. For all other scenarios, the decentralized architecture is the best choice.

5.3. Synchronous and Asynchronous Communication

The model updates exchange between different machines can be performed adopting two different communication strategies: synchronous and asynchronous. During the computation of SGD, the updates exchange procedure is often crucial for its convergence speed and, consequently, it affects also the training speed. In the following sections we investigate the aspects related to both modes, explaining benefits and drawbacks.

Synchronous Communication The simplest implementation of a synchronous communication strategy is the Bulk Synchronous Parallel (BSP) strategy [78]: all nodes work on the same task and, at each step, each node pushes data to the other ones or to the PS, for decentralized and centralized environment respectively. All nodes stay in idle until the last node pushes the data; this behaviour sets the limit of the so-called *synchronization barrier*. Then, the model updating is performed and the next iteration is started. Intuitively, the general performance depends

mainly on the slowest machine and on the workload balance. Moreover, due to the synchronization step, the communication cost can be higher than the computational one.

To overcome these issues, a Stale Synchronous Parallel (SSP) strategy is proposed [79], based on the relaxation of some constraints with respect to BSP. In SSP, each node computes the parameter upgrades and updates the model independently of the other ones. When a threshold is reached (called "staleness threshold" and defined as the maximum difference between the number of iterations accomplished by two different machines) the synchronization step is performed. The higher is the staleness threshold, the lower is the communication cost and the more this communication strategy becomes similar to the asynchronous one. However, also in this case the overall performance depends on the slowest machine. To face such issues, Shi *et al.* [80] proposed a new synchronous parallel strategy, called Free Stale Synchronous Parallel (FSSP). Specifically, this strategy is similar to the SSP with a penalty parameter. When a slow node reaches the staleness threshold, it receives a penalty; when the number of penalties reaches a threshold, the node is excluded from the computational task (this operation is called "*punishment*"). To avoid the information loss due to the exclusion mechanism, the adjacent nodes share a sub-set of the same parameters. In this way, if the *i-th* node is excluded from the task, the neighbour nodes (*e.g.* the *i-1* and *i+1* nodes) can still update the parameter held by the *i-th* node. However, also two adjacent nodes could reach the penalty threshold. To protect the task integrity, when a node is punished, a lock operation is performed on the two adjacent nodes, preventing the punishment operation on them until the task is completed. Yang *et al.* proposed a Limited Synchronous Parallel model (LSP) for PS configurations [81]. Based on BSP concept, the authors developed a Dynamic Synchronous Parallel model (DSP), using an adaptive threshold. In particular, each worker, trough a performance monitoring process, pushes the iteration time with a synchronization barrier preemption signal to the PS. Then, the worker receives from the PS a synchronization flag: if the flag is equal to 0, the worker updates the model using the local gradients; else, if the flag is equal to 1, it enters the synchronization barrier, sends the local gradient to the PS and then receives the new global parameters. On the other hand, the PS pulls

data and performance from each worker, computes the adaptive limited threshold and sends the synchronization barrier preemption signal. According to the state of synchronization barrier, the PS determines whether the worker is successfully in preemption and consequently sets and sends the synchronization flag. If the flag is equal to 1, the PS pulls all the local gradients, aggregates them, computes the new global model parameters and sends them back to the workers.

Asynchronous Communication To mitigate the problems arising with the synchronous communication strategy, it is possible to revert to an asynchronous one. *HOGWILD!* [82] is among the most well known asynchronous communication algorithms. After setting N equal to the number of available workers, *HOGWILD!* partitions the dataset in N sub-sets and sends each subset to a different worker, that holds a copy of the model. The global model is stored in a shared memory accessible by all nodes. During the training phase, each worker computes the local gradient for every batch and pushes it to the shared memory without blocking the other nodes. Whenever needed, each node can pull the new global parameters from the shared memory and update its local model. This approach is suitable for centralized architectures, such as PS. For decentralized approaches, a new version of *HOGWILD!* has been developed, called *HOGWILD++* [83]. In fact, in this work the authors show the limitations of *HOGWILD!* in multi-socket CPUs systems with Non Uniform Access Memory (NUMA). These limitations are related to the high number of read/write operation to the memory blocks holding the global model. This causes a high number of cache invalidation requests and model coherence misses. To improve the scalability of *HOGWILD!*, so overcoming these issues, the authors propose to replace the global model vector with a set of local model vectors, shared by a cluster of cores connected to the same NUMA node. Each local vector is updated independently; then, after a fixed number of updates, each cluster shares its local vector state with its neighbours, using a distributed token mechanism. Due to the use of local variables and periodic updates exchange between cores, the cache invalidation and incoherence conditions are consistently reduced. Lian *et al.* [84] tried to overcome the central server bottleneck in asynchronous distributed environment, by keeping the best possible convergence rate. To accomplish this task, they implemented an Asynchronous Decen-

tralized Parallel Stochastic Gradient Descent (AD-PSGD) algorithm, described by the following steps: each worker holds a local replica of the global model and samples a mini-batch of training data. Then, it computes the local SGD and updates the local model. As final step, it randomly selects a neighbour i and merges its local model with the one hold by i. The asynchronous procedure reduces the idle time of each worker. The algorithm can theoretically achieve the same convergence of centralized/synchronous counterparts; its speed increases linearly with the number of workers.

A similar strategy is proposed by Tu, Zhou and Ren [85], that introduce a framework based on the combination of Gossip Protocol with SGD, called Gossip Rings SGD (GR-SGD) for machine learning applications related to Cyber-Physical Systems (CPS). In particular, a *Mixed Adjoint Matrix* $M^{(t)}$ is designed for leading the node selection. At each iteration, a node i is awakened and a neighbour node j is randomly selected.

Then, the selected node fetches a data mini-batch from the local distribution, computes the SGD locally and sends the update to j. The process is repeated until the training process is complete. $M^{(t)}$ is designed based on Poisson Clock and Markov Chain [86] and for this reason the probability of a worker awakening is not independently distributed (*i.e.* only a neighbour of an awakened worker can be awakened). To overcome this issue, the edge of an awakened worker is removed, so that the awakening probability is independently distributed; removing the edges of workers introduces a new network topology, called *Interaction Topology*.

Finally, Hu *et al.* [87] proposed the FDML (Feature Distributed Machine Learning) framework. Differently from the other works, in FDML the aggregation operation is not performed at parameter update level, but at prediction level. Moreover, the data are distributed among different workers without sharing: no worker shares its feature set with the other ones. The authors implemented an asynchronous distributed version of FDML, inspiring themselves to PS architecture. In this scenario, the server holds and updates a matrix $A_{i,j}$ (i= *1,...,n* and j= *1,...,m*) related to the latest m predictions of the n samples. On the other hand, each worker computes the gradient and update its local model, adopting the SGD algorithm. The key idea consists in the use

of the A matrix during the computation of SGD, and in the asynchronous update of A by all workers. This peculiarity causes different workers to be involved in different iterations; thus, the fastest workers can pull a stale local prediction pushed by the slowest ones: a stale synchronous protocol has been adopted to face this problem.

Two hybrid strategies have been implemented ([88], [89]), aiming at optimizing synchronous and asynchronous strategies. In the first work, the authors examined the source of the parameters staleness problem in asynchronous strategy, leading to a worsening of performance and slow training process. They found the reasons in the *Pushes After Pull* (PAP): a worker pulls the parameters to start the next iteration, meanwhile it misses other workers pushes. To overcome this issue, they propose a *Speculative Synchronization* (SpecSync) strategy based on a simple naive waiting idea. In particular, each worker pushes its parameters and pulls the others to start the next iteration, in asynchronous way. Moreover, each worker is aware about the number of global model updates performed, since the start of the new iteration. When the number of updates exceeds a threshold, the worker aborts the ongoing computation, pulls the fresh parameters and re-start the computation.

The work by Tan *et al.* [89] aims at optimizing the two strategies by adopting different parallel training schemes according to the workers training speed. This strategy is called *Adaptive Synchronous Strategy* (A2S). Specifically, the workers are divided in two groups: Sync and Asynch, containing the fastest and the slowest workers, respectively. A2S follows this workflow:

1. Each worker loads a data partition, computes the gradient and pushes it to a PS, waiting for the global update.
2. The PS receives all gradients and operates with two modes:

   M1: a *Counter* keeps track of iterations and epochs of each worker. If the workers completed an epoch and the Synch group did not receive any new gradient, it sets an update flag to *True*. The PS checks the flag variable. If true, a *Monitor* collects the epochs from the counter and updates the both groups. Then the system turns to M2.

   M2: the PS pushes the gradient to the two groups. For Synch group, each worker pushes the gradient to an *aggregator* that, in turn, pushes the resulting gradient to a SGD optimizer; afterwards, the

updated parameters are broadcast to all workers of the Synch group. On the other hand, each worker in the Asynch group, pushes independently its gradient to the SGD optimizer to update the global parameters; then updated parameters are sent back only to the worker that pushed its gradient.

The choice of a synchronization strategy affects performance in terms of SGD convergence and training speed. In both strategies, slowest workers can affect the training performance in different way. On the one hand, synchronous strategies tend to achieve better SGD convergence performance, but they can slow down the overall training process due to synchronization steps. The performance of synchronous strategies is also influenced by the presence of stragglers, even in the FSSP framework. On the other hand, asynchronous communication is generally faster, but its implementation is more challenging. Furthermore, in this situation, performance is impacted by the slowest worker, which can introduce outdated gradient updates to the faster workers, leading to a degradation in SGD convergence performance (Gradient Staleness). To address these drawbacks, it is recommended a hybrid strategy, such as SpecSync or A2S. SpecSync can be seen as a combination of both communication patterns, leveraging synchronous communication to resolve the Gradient Staleness typical of asynchronous communication. A2S, although more complex to implement, is suitable for scenarios involving a PS and heterogeneous workers with significant differences in computational speed.

5.4. General Discussion

In the previous sections, we have explored the three main choice factors affecting distributed computation for training ML and DL models:

- Parallelization Techniques
- Communication Architectures
- Communication Strategies

As discussed earlier, there are several factors to consider during the design process. These factors include network characteristics (such as topology, bandwidth, and latency), cluster hardware configuration, training speed of models, types of tasks involved, SGD convergence, management complexity, and more. It's crucial for developers to consider the interrelationship among

these factors, that affect among each other. For example, in a scenario with a significant disparity in computational speed among workers, choosing a data parallelism approach with an asynchronous communication pattern may result in performance degradation, if the Gradient Staleness problem is not properly addressed. Conversely, adopting a synchronous or hybrid communication strategy becomes preferable over an asynchronous one when there is already a central entity (e.g., the PS) capable of sending updates to all workers simultaneously. Therefore, answering the question "What strategies and techniques should I adopt for the distributed training of my ML/DL model?" is not a straightforward task, as there is no direct answer. The selection of the best strategies, and consequently of the best framework, depends on the specific needs of the developer. We identified the following primary factors that developers should consider when making their choice, w:

- **Underlying cluster architecture.** It is crucial to choose a framework that aligns with the architecture of the cluster. If the cluster follows a centralized architecture, opting for a framework developed for decentralized architecture would not be suitable. Conversely, frameworks designed for decentralized architecture may not fully leverage the capabilities offered by a PS.

- **Parallelization algorithm.** The framework should support parallelization algorithms capable of handling big data and complex models. In scenarios involving such requirements, the absence of these features can slow down the training process or, in the worst case, make the training infeasible due to memory limitations.

In DML scenarios, the supporting device and the communication mode take a backseat in the decision-making process. This is because the former can significantly enhance the training process speed, while the latter primarily affects training performance in terms of SGD computation and convergence, based on the computational speed of the workers. Instead, the focus should initially be on satisfying the basic requirements outlined above, making only later choices on the supported device and communication modalities. DDL scenarios are different and the supporting device takes a primary role for model training and inference (see Section 6.2.1). For the sake of synthesis, we provide in Table 2 a brief summary of the content of this section, along with potential benefits and drawbacks.

Table 2: Summary of key enabling techniques and algorithms, with a brief description, benefits and drawbacks.

Topic	Description	Benefits	Drawbacks	Considerations
Data Parallelism	The dataset is splitted into sub-sets. Each worker holds a replica of the model, that is trained on a sub-set. Then the model updates are shared among workers.	Acceleration of the training phase, Workload reduced at single node level, The addition of more nodes increases the capacity of handling Big Data.	Splitting data into sub-sets is challenging, increasing of the network traffic.	DP should be considered especially when the task concerns the use of simple ML models with Big Data. Suitable for edge computing.
Model Parallelism	The model layers are splitted among different workers. A worker trains its layers sub-set and sends the results to the next one. The same process is perfomed in reverted order for backward propagation.	Acceleration of the training phase, Workload reduced at single node level, Memory Requirements reduced The addition more nodes increases the capacity of training DL complex models.	Implementation complexity, Workers communication complexity, Idle workers for naive implementations, Model size dependency.	MP-based approaches are the only strategies for handling non-linear heavy models, that cannot fit the memory of a single worker.
Hybrid Parallelism	It combines Model Parallelism and Data Parallelism at the same time.	Join the benefits of DP and MP.	Increasing of the complexity during the implementation phases, Increasing of the overall management complexity.	Hybrid strategies should be considered only in case of real need. DP and ML are sufficient for accomplishing most BD analytics tasks.
Centralized Architecture	The Parameter Server retrieves partial gradient updates and computes the global gradient during the training process (reduce); then it broadcasts it to all workers.	Simple Implementation, faster computation of the global model gradient, low management complexity.	Hardware Resources sufficient, possible networks bottlenecks, Redundancy and Recovery policies needed.	Suitable for star network-like topology, with DP and Synchronous communication approaches. In asynchronous approaches, the adoption of a shared memory on PS side is suggested.
Decentralized Architecture	Communication distributed among different workers, with a peer-to-peer (P2P) network topology. Basic Implementation: AllReduce.	No particular hardware resources required. The training process can be easily completed even in case of node failure.	High Management Complexity, Network Traffic Increased w.r.t. PS.	In absence of a central entity and with heterogeneous hardware, it results the only reasonable choice.
Synchronous Communication	All workers perform the task independently and, after a period (time, fixed step, thresholds), perfom a synchronization step.	Easy to implement, better SGD convergence performance.	Presence of possible Stragglers, overhead due to synchronization step.	Suggested for PS-based scenario, but it can be implemented for decentralized architecture.
Asynchronous Communication	All workers perform the task independently and, according to the strategy adopted, they share partial gradient updated among themselves.	Stragglers problem addressed, the performance can match the same of synchronous approach, updates sharing faster w.r.t. synchronous approach, no synchronization step required.	Gradient Staleness, more management complexity, SGD convergence performance lower w.r.t. synchronous approach.	It should be preferred when there is an important difference between workers in terms of computational speed.
Hybrid Communication	It combines both aspects of synchronous and asynchronous communication to address the stragglers and gradient staleness issues.	Join the benefit of synchronous and asynchronous communication, mitigating the stragglers and gradient staleness issue.	Increasing of communication management complexity.	Due to the increase of management complexity, such approaches should be chosen only when stragglers and gradient staleness issues affect the overall performance in a non-negligible manner.

6. Classification of 13 DML and 20 DDL Frameworks

We here classify according to the criteria in Section 3.3, 13 DML and 20DDL frameworks retrieved by the methodology in Section 3.

Some frameworks show missing information (denoted by a NA value in tables) with respect to the classification criteria, due to the lack of documentation. Although frameworks for DDL often includes DML algorithms, we discuss them in separate subsections. The level of detail in the frameworks documentation is not uniform: only for a few frameworks, the specific parallelization algorithms, communication modalities, and cluster architecture supported are explicitly mentioned. Thus, for an homogeneous classification, we only indicate the general categories of distributed strategies adopted by each framework. We notice that the frameworks classification according to the class of algorithms (ML or DL) is implicit in the splitting in subsections.

6.1. Distributed Machine Learning Frameworks

Table 3 classifies DML frameworks with respect to the identified classification criteria: Supporting device, Parallelization technique, Supported Architecture, and Communication mode. For readability reasons, the frameworks Working Mode has been shown in Table 4.

This section shows retrieved DML frameworks and related capabilities. Some of them are built on top of other ML frameworks, and use them as back-end. In this case, we chose to classify only the back-end framework

and just mention the top-frameworks, unless their functionalities affect the back-end framework functioning.

The retrieved DML frameworks are listed below:

1. **H2O** [90] - Developed by H20.ai company, H20 is a distributed in-memory ML platform, supporting statistical and machine learning algorithms. Written in Java, it also offers Python, R, Scala, and JSON APIs. It is composed by several modules, such as *H2O4GPU* for GPU computation or *Sparkling Water* to exploit Spark functionalities. The last version released is H20-3.
2. **Apache Spark - MLib** [91] - Apache Spark is the most popular engine for data analysis. Adopting a Resilient Distributed Dataset (RDD), it allows the processing of streaming data as well as data batches. Moreover it includes a module for SQL analysis and a set of ML algorithms included in the library MLib. It offers API in Python, Java, Scala, R and SQL.
3. **TransmogrifAI** [92] - TransmogrifAI is an AutoML library built on top of Apache Spark. Its goal is to easily build ML workflow while minimizing the manual tuning.
4. **Vertica-ML** [93] - Vertica-ML is distributed in-memory ML system included in the Vertica database. It integrates machine learning functionalities and algorithms with SQL API in order to offer a full data science workflow.
5. **PyCOMPSs/dsLib** [94, 95] - PyCOMPSs is a Python version of COMP Superscalar (COMPSs) framework, designed for the development of applications in distributed environments. It is composed by two modules: the *programming module* offers simple functions and primitives for defining the application parallelism; the *runtime module*, deals with the analysis of functions defined in the programming module and with computation distribution, following a server-workers approach. dsLib is a collection of PyCOMPSs applications, and offers a distributed data management interface and and estimator API.
6. **Cymbalo** [96] - Built on top of Apache Spark, Cymablo is a distributed graph processing framework, employing a heterogeneous-aware data model, a hybrid computing model and a Vector-centric programming model.
7. **VariantSpark** [97, 98] - VariantSpark, as the name suggests, is a Spark-based framework for the analysis of genomic-scale datasets. It

implements a vertical data partitioning, it uses the RDD and it processes multiple trees in parallel.

8. **HotML** [99] - HotML is a distributed machine learning framework designed for PS architecture. In particular it introduces a dedicated PS to maximize the resource utilization, implementing a SSP-based communication and a workeload balancer to face the straggler problem.
9. **Litz** [100] - Litz is a PS-based framework supporting the development of distributed ML applications. It implements: an event-driven programming model separating the application from the cluster; a task-driven consistency model for model scheduling and consistency; and an elastic execution system for workload balance.
10. **SIREN**[101] - Developed by Wang *et al.*, SIREN is an asynchronous distributed machine learning framework based on decentralized architecture. Its goal is to achieve an high level of parallelism, using stateless functions. Moreover it implements a Deep Reinforcement Learning (DRL) algorithm for scheduling.
11. **Distributed WEKA** [102] - Waikato Environment for Knowledge Analysis (WEKA) is a collection of data mining and machine learning algorithms, implemented in JAVA. It has been released under GNU General Public License. Distributed WEKA is a WEKA plugin for distributing computation using Apache Spark.
12. **KungFu** [103] - Based on TensorFlow, KungFu is a distributed ML framework for the adaptive training. In fact, with its Adaptation Policies (APs), the developer can define the customization of the system parameters and model hyper-parameters during the training. APs implement asynchronous communication layers and can be used with other frameworks such as Pytorch and Keras.
13. **Apache Mahout** [104] - It is a distributed framework for the implementation of distributed algorithms. It has been designed specifically for statisticians, mathematicians and data scientists. It can use external back-end for computation distribution (*i.e.*, Apache Spark) and it supports native solvers for CPU/GPU/CUDA acceleration.

According to the descriptions above, TransmogrifAI, VariantSpark, Cymbalo, Distribute WEKA, and Mahout are Spark-based. For this reason, in Table 3 only Apache Spark is shown, since it is describes the mentioned frameworks, from a distributed learning point of view.

Table 3: Distributed Machine Learning Framework comparison according to the first four classification criteria.

	Parallelization Mode		Supporting Device	Supported Architecture		Communication Mode	
Framework	**Model Parallelism**	**Data Parallelism**		**Centralized**	**Decentralized**	**Asynchronous**	**Synchronous**
H2O	X	X	CPU, GPU	X	X	X	
Vertica		X	CPU	X	NA		X
PyCOMPSs - dsLib		X	CPU, GPU	X	X	X	X
Cymbalo		X	CPU	X		X	X
HotML	X	X	CPU	X			X
Litz		X	CPU, GPU	X	X	X	X
SIREN	X	X	CPU		X	X	X
Apache Spark		X	CPU, GPU	X		X	X
KungFu	X	X	CPU, GPU	X	X	X	

Table 4: Distributed Machine Learning Framework comparison according to the working mode.

Framework	**Working Mode: Compatibility**		
	Stand Alone	**Apache Spark**	**Others**
H2O	X	X	
Apache Spark - MLib	X		
TransmogrifAI		X	
Vertica-ML	X		
PyCOMPSs/dsLib	X	X	
Cymbalo		X	
VariantSpark		X	
HotML	X		
Litz	X		
SIREN	X		
Distributed WEKA		X	
KungFu			X
Apache Mahout		X	X

Notably, for Spark-based frameworks, the Supporting Device feature changes according to Spark version. In fact, in Spark the GPU acceleration is supported starting from *Spark 3.0.0*.

It should be noticed that the *Model Parallelism* feature is ambiguous in DML context. In fact, it is often expressed as the capability of the framework to train and evaluate different models at the same time on different nodes. Instead, in the literature Model Parallelism is defined as the single model partitioning on multiple workers. For this reason, although some frameworks claim model parallelism in their documentation, we did not indicate it in Table 3. As an example, in Spark model parallelism is expressed as the capability of evaluating models in parallel, so that Shrivastava *et al.* [105] proposed an extension of Spark, implementing the model parallelism feature.

6.1.1. DML Frameworks Discussion

In the previous section, we examined 13 DML frameworks. Table 3 demonstrates that all frameworks implement data parallelism as the baseline for distributed computation. This implementation is a reasonable choice for ML tasks that involve training and validating non-complex models for big data analytics. Data parallelism plays a fundamental role in speeding up training processes when handling large volumes of data. Model parallelism, implemented by H2O, HotML, SIREN, and KungFu, is a secondary feature. Regarding supported devices, CPU support is the baseline. However, utilizing powerful devices like GPUs or TPUs can significantly enhance computational speed. Among the frameworks, H2O, PyCOMPSs/dsLib, Litz, Apache Spark (and Spark-based frameworks compatible with the Spark version), and KungFu can exploit the capabilities offered by GPUs. KungFu potentially supports TPUs since it is built on top of TensorFlow (see Section 6.2 for more details) but the support is not explicited in the related documentation. All frameworks, except SIREN (designed to be serverless) support centralized architecture. Decentralized computation is only supported by a few frameworks, likely due to its greater management and communication complexity. Both synchronous and asynchronous communication modes are equally supported by all frameworks.

A key factor for discriminating frameworks is the kind of supported devices. In particular, by assuming that all frameworks support CPU computation, developers should prefer those that also implement GPU computation to reduce training time.

The framework compatibility is also a discriminating factor among frameworks satisfying all other requirements (in terms of architecture, parallelization and so on). On one hand, a stand-alone framework has fewer dependencies compared to a framework that relies on others, resulting in fewer

compatibility and configuration issues. On the other hand, if a framework supports another one, it can extend its functionalities and receive periodic updates, as long as the corresponding framework is also updated.

Additional features in frameworks can make one more attractive than another. Apache Spark is the most widely used framework as it integrates various programming languages (Python, R, Java, and Scala), multiple ML algorithms in its libraries, and allows for direct querying of data using SQL. It can also be easily integrated with other ML/data analytics ecosystems such as TensorFlow, PyTorch, Pandas, mlflow, numpy, and more. These features place Apache Spark (with MLlib) at the forefront of DML frameworks in terms of usability and integration. H2O and TransmogrifAI offer an interesting feature called AutoML, which automates model training and tuning with minimal human effort. Additionally, H2O supports in-memory computation and its algorithms can be implemented in Python, R, and H2Oflow. These frameworks are recommended for those who are not experts in fine-tuning models or those who do not require control over model hyperparameters. For ML graph-based algorithms, Cymbalo is the best choice, providing a heterogeneous data-aware model, a vector-aware graph layout for reduced memory footprint, a hybrid computational model for handling different ML algorithms, and a vector-centric programming model for accurate and straightforward programming. PyCOMPSs/dsLib is suitable for IoT environments, offering a task-based programming model for the Cloud-Edge-IoT continuum. Similar to Spark, Vertica-ML offers a set of ML algorithms and SQL APIs for building a data science workflow.

Regarding compatibility, Apache Spark is the computation engine most used in the field of DML and most embedded in other frameworks. This is evident in Table 4, where we highlight the compatibility and dependence of the retrieved frameworks with other frameworks or with Apache Spark. Specifically, seven frameworks are compatible with Spark (excluding Apache Spark itself). Other seven frameworks can work as standalone solutions, while KungFu is the only framework that relies entirely on another framework, namely TensorFlow. If TensorFlow is already present in the environment, the developer should prioritize KungFu. However, in other scenarios, Apache Spark emerges as the preferred choice since it can be seamlessly integrated with other environments.

6.2. Distributed Deep Learning Frameworks

The retrieved DDL frameworks are shorthly described below:

1. **PyTorch** [106] - Based on Torch library and initially developed by Meta, PyTorch is an open-source Deep Learning framework. It offers a interface in Python, as well as in C++. PyTorch is composed by several components: *torch*, a tensor library; *torch.autograd*, a library supporting differentiable Tensor operations; *torch.jit*, a module for the creation of serializable and optimizable models; *torch.nn*, a neural network library; *torch.multiprocessing*, for data loading and Hogwild training; *torch.utils*, a library containing all utilities function. The last released version is PyTorch 2.0.
2. **TensorFlow** [107] - Together with PyTorch, TensorFlow (TF) is one of the most used end-to-end deep learning frameworks, initially developed by Google Machine Intelligence Research team. It provides APIs in Python, Java, Go, RUST and C/C++, offering a support for both CPU and GPU computation.
3. **Caffe2** [108] - Caffe2 is a modular, scalable and lightweight deep learning framework. Caffe2 is now deprecated and included in PyTorch, but it is still widely used.
4. **Chainer** [109, 110, 111] - Chainer is a python-based deep learning framework, designed to be powerful, intuitive and flexible. It provides high-level APIs for model development and supports CUDA/cuDNN computation. It is available in its last version, Chainer v5.
5. **BigDL2.0** [112] - BigDL2.0 is framework for building distributed end-to-end deep learning applications. BigDL2.0 project was born from the merging of BigDL [113] and AnalyticsZoo [114] projects, developed by Intel. It provides the following libraries: *Orca* for distributed Big Data pipeline with TF, PyTorch, Spark and Ray; *Nano* for the acceleration of FT and PyTorch applications; *DLlib* a Spark library for Deep Learning; *Chronos* for Time series analysis; *Friesian* end-to-end module for Reccomendation Systems; *PPML* for secure Big Data and AI processing.
6. **Apache SINGA** [115] - SINGA is a python framework for distributed deep learning. It provides different features, such as several deep learning model examples (model Zoo), distributed training features, memory and parameter optimization, automatic gradient computation, among others.
7. **Elephas** [116] - Based on Master-worker architecture, Elephas is an extension of Keras (the high-level library on top of TF) allowing the

implementation of DL algorithm using the Spark scale. It implements parallel data algorithms over , using RDDs and dataframes.

8. **TensorflowOnSpark** [117] - Similarly to SINGA, TFOnSpark enables the development of DL algorithms integrating them with Spark and Hadoop. Due to such an integration, it is possible to use RDD with TF and TFrecords on HDFS (Hadoop Distributed File System). Moreover it supports CPU and GPU computation.
9. **Oneflow** [118] - OneFlow is a scalable, efficient and user-friedly DL framework offering a program model with PyTorch-like API. Moreover it enables the execution distribution via Global View API and a model deployment acceleration using Static Graph Compiler.
10. **Horovod** [119] - In its first versions, Horovod has been developed by Uber's team. It is designed to be fast, portable, extremely scalable and easy to use. Moreover, it can run on top of Apache Spark and, once configured, allows for switching withiut additional settings to other integrated frameworks, as TF, MxNet, and PyTorch.
11. **SDDLF** [120] - Based on to PS architecture, Spark based distributed Deep Learning framework (SDDLF) is framework aiming at training deep neural networks using Spark as back-end with HDFS. Its design is inspired to Google DistBelief.
12. **TFSM** [121] - TFSM is a DDL framework developed with a specific focus on optimal paramater sharing. Such a framework uses TF as back-end with Shared Memory (TFSM); the underlying architecture is PS-workers with a shared memory accessible by all workers, in an asynchronous way.
13. **ShmCaffe** [122] - Similar to the previous one, ShmCaffe is a DDL framework designed for high performance computing (HPC) tasks. It is Caffe-based with a virtual shared memory, called Soft Memory Box (SMB), accessible by all workers. Differently from TFSM, the access to the shared memory is handled either in asynchronous or synchronous way, in order to improve the scalability.
14. **Petuumv2** [66] - Originally born as the ML platform Petuum, since the release of its second version it integrates also deep learning functionalities. For DDL algorithms it uses three main modules: *Tunn* for efficient hyper-parameters tuning; *AdaptDL*, a dynamical DL training and scheduling framework; *AutoDist*, a DDL training engine for TF.
15. **Apache MXNet** [123] - MXNet is another widely used framework

for the development of DDL models, supporting imperative and symbolic programming. It includes a dependency scheduler for functions parallelization on the fly, and a graph optimization layer for symbolic execution optimization. It provides NumPy-like interface and it is high portable, since it can be used with external ecosystems. MXNet offers support for several programming languages, such as C++, Java, Python, R, Scala, Julia and so on.

16. **Petastorm** [124] - Developed at Uber ATG, Petastorm is an open source data library enabling the single or distributed training and evaluation of deep learning models. It analyses datasets in Apache Parquet format, and supports also other frameworks as Tensorflow, PyTorch, and PySpark; it can also work without an external back-end.
17. **Mesh TensorFlow** [125] - MeshTF aims at formalizing and implementing distribution strategies among different computation hardware sources. Mesh TF is built as a top-level on TensorFlow
18. **DeepSpeed** [126] - DeepSpeed is a deep learning optimizator framework developed by Microsoft. It is composed by three different modules: DeepSpeed-Training, DeepSpeed-Inference and DeepSpeed-Compression, for optimizing models training, inference and compression, respectively. It uses PyTorch as external back-end, resulting a lightweight wrapper on it.
19. **DeepSpark** [127] - DeepSpark is a deep learning based framework, built on top of Apache Spark. It implements data parallelism algorithm, with asynchronous SGD for optimizing the model training. It also supports two DL frameworks, Caffe and Tensforflow, integrating them with Spark. It is one of the first framework intgrating TF with Spark.
20. **CoCoNet** [128] - Communication and Computation optimization for neural Networks (CoCoNet) is a deep learning framework developed by Jangda *et al.*, providing a domain specific language for ML/DL applications distribution, a set of semantics for programs optimizaion and a compiler for jointly GPUs communication and optimization. CoCoNet code has been integrated as *torch.distributed* module. It is worth noticing that CoCoNet implements also the Pipeline parallelism.

The *Qubole* [129], is not discussed here due to the lack of related documentation. Moreover, we did not include *Keras* and *Mesh TensorFlow* because they are built on top of TensorFlow, inheriting its capabilities.

Table 5 compares the frameworks above, according to the classification criteria discussed in Section 3.3; the working mode is reported in Table 6.

Table 5: Distributed Deep Learning Framework comparison according to classification criteria.

	Parallelization Mode		**Supporting Device**	**Supported Architecture**		**Communication Mode**	
Framework	**Model Parallelism**	**Data Parallelism**		**Centralized**	**Decentralized**	**Asynchronous**	**Synchronous**
PyTorch	X	X	CPU, GPU, TPU	X	X	X	X
TensorFlow	X	X	CPU, GPU, TPU	X	X	X	X
Caffe2		X	CPU, GPU		X		X
Chainer	X	X	CPU, GPU, TPU		X		X
BigDL2.0		X	CPU, GPU	X	X		X
Elephas		X	CPU, GPU	X		X	X
SINGA	X	X	CPU, GPU	X	X	X	X
TensorFlowOnSpark	X	X	CPU, GPU	X	X	X	X
OneFlow	X	X	CPU, GPU		X		X
Horovod	X	X	CPU, GPU	X	X	X	X
SDDLF	X	X	CPU	X		X	
TFSM		X	CPU, GPU	X		X	
ShmCaffe		X	CPU, GPU	X		X	X
Petuumv2	X	X	CPU, GPU	X	X	X	X
MXNet	X	X	CPU, GPU	X	X	X	X
Petastorm	X	X	CPU, GPU	X	X	X	X
DeepSpeed	X	X	CPU, GPU	X	X	X	X
DeepSpark		X	CPU, GPU	X		X	
CoCoNet	X	X	CPU, GPU		X	X	X

Table 6: Distributed Deep Learning Framework comparison according to the working mode.

Framework	Working Mode: Compatibility	
	Stand Alone	External Back-End
PyTorch	X	
TensorFlow	X	
Caffe2	X	
Chainer	X	
BigDL2.0		X
Elephas		X
SINGA	X	
TensorFlowOnSpark		X
OneFlow	X	
Horovod		X
SDDLF		X
TFSM		X
ShmCaffe		X
Peetumv2	X	
MXNet	X	
Petastorm	X	
DeepSpeed	X	
DeepSpark		X
CoCoNet		X

6.2.1. DDL Frameworks Discussion

As for DML, we have examined 20 DDL frameworks, according to the features set as classification criteria.

As for parallelization, model parallelism plays a significant role in DL tasks, particularly during the training and inference phases. Unlike DML or ML in general, DL tasks involve training and validating complex non-linear models that often cannot fit on a single device. Additionally, training complex models from scratch (e.g., CNNs, large transformers, and GANs) can take several weeks, even when performed on multiple devices in parallel. For this reason, developers often adopt the transfer learning approach [130] by fine-tuning pre-trained models. In this scenario, MP becomes a key strategy. Supporting this assumption, Table 6 shows that almost all DDL frameworks support MP, even though DP remains the baseline for parallelization, as for DML.

Similar considerations apply to supported devices. It is highly recommended to use GPUs as accelerator devices due to their ability to efficiently perform parallel mathematical operations. In recent years, also TPUs have been increasingly adopted. TPUs are optimized for matrix-tensor calculations compared to CPUs/GPUs and can offer performance similar to GPUs with lower energy consumption. However, TPU support is still limited, due to its relatively recent adoption. While all frameworks support CPU and GPU computation (except for SDDLF), only PyTorch, TensorFlow, and Chainer provide support for TPU computation.

As stated for DML, developers must select a framework that aligns with the architecture of the cluster. Table 6 demonstrates that half of the frameworks support both centralized and decentralized architectures, similarly to DML.

In terms of communication mode, most frameworks equally support synchronous and asynchronous modes, with 12 frameworks supporting both of them. In scenarios where there is a significant difference in computational speed among cluster workers, asynchronous communication is generally preferred.

Regarding the working mode, as discussed for DML frameworks, a standalone framework has fewer dependencies and therefore fewer compatibility issues. However, a framework built on top of another framework implements the same functionalities and extends them. In this case, it is crucial to ensure that the top framework is up to date according to the updates of the underlying framework. Regarding periodic updates, most frameworks are up to date, with the exception of SDDLF, TFSM, ShmCaffe, and DeepSpark. Caffe2 and CoCoNet have been integrated into PyTorch.

PyTorch and TensorFlow stand out as the most widely used DL frameworks, offering a wide variety of models. Both frameworks leverage XLA (Accelerated Linear Algebra), an open-source ML compiler that optimizes computations for GPUs, CPUs, and TPUs accelerators. In contrast, Chainer provides its own API for TPU computation. While TensorFlow and PyTorch can be used in distributed environments, the responsibility for setting up communication between different workers falls upon the developer. These frameworks offer functions for distributing computations among different accelerator devices hosted on the same worker. However, they are not suitable for developers tht prefer to handle the communication setup between workers. Among the frameworks examined, Petastorm, Mesh TF, Horovod, and BigDL provide APIs that support developers in setting up communication

between workers. Additionally, except for Mesh TF, these frameworks are compatible with Spark. It is important to note that while Petastorm, Mesh TF, Horovod, and BigDL include support for PyTorch and TensorFlow, only Mesh TF is limited to TensorFlow, and none of them offer TPU computation support.

7. Evaluation Metrics for DML/DDL Frameworks

It is worth investigating the evaluation of Distributed Artificial Intelligence frameworks, with specific reference to DML/DDL ones. Here, we report and analyze the most significant evaluation criteria and metrics adopted by the works retrieved.

The evaluation criteria/metrics can be very different [131, 132]. Thus, their applicability and effectiveness depend on several factors, including the cluster architecture and configuration (*e.g.*, centralized or decentralized architecture, network type and configuration, maximum network bandwidth available, number of workers, single worker configuration), the models used as case study, and the datasets involved.

For this reason, the choice of the best criteria/metrics to evaluate a framework is quite challenging.

We here collect all criteria/metrics employed in the works analyzed in Section 6. We notice that, although several works declare the used benchmarks and the achieved performance, their experimental setups are very different from each other, making impossible a direct comparison of frameworks in terms of performance.

As a guideline, the experiments should adopt the so-called Design of Experiments (DoE) [133, 134] approach, according to the following principles: Comparison, Randomization, Statistical Replication, Blocking, Orthogonality and Multifactorial experiments. In DML and DDL context there are several works for performance study based on DoE approach [135, 136, 137, 138, 139].

The works retrieved and analyzed in Section 6 refer to a really heterogeneous set of evaluation criteria and/or metrics, whose main items are summarized below.

Computational Time In a single-machine approach, the computational time (CT) can be defined as:

$$CT = T_{start} + T_{dataL} + T_{tr} + T_{eval} \tag{1}$$

Where T_{start} is the startup time needed by the framework, T_{dataL} is the time related to data loading, T_{tr} is the training time and T_{eval} is the time related to model evaluation and inference. In turn, T_{tr} changes according to the training setup, (*e.g.* batch size, learning rate, number of epochs, computation devices and so on). Intuitively, in a distributed environment, there are other factors that shall be taken in account. In such a scenario we can model the CT as follows:

$$CT' = T'_{start} + T'_{data} + T'_{tr} + T_{eval} + T_{NL} + T_{fail} \tag{2}$$

Differently from Eq.1, T'_{start} includes also the overhead for cluster startup, nodes communication and job scheduling, T'_{data} is the time required for data loading and distribution among nodes (see Section 5.1), T'_{tr} is the training time considering also the model replica and the parameter updates exchanges/aggregation. Finally, T_{NL} and T_{fail} represent, respectively, the time related to the network latency and the wasted time related to node failing and recovering; ideally $T_{fail} = 0$. We can refer to CT' as the end-to-end time processing time, T_{e2e}.

Memory Footprint The memory footprint refers to the amount of memory required by the running process. For *in-memory* [140] computation frameworks (*e.g.* Apache Spark) there is a trade-off between memory footprint and processing time: the higher is the first, the lower is the second [141] .

Number of Computational Devices According to the framework implementation and the cluster resources, the elaboration process can use different kinds of devices, such as CPUs (including also the number of cores), GPUs or, for specific tasks with DL state-of-the-art frameworks, Tensor Processing Units (TPUs) [142]. Although a cluster can include heterogeneous devices, for performance evaluation with respect to the number of computational devices, it is suggested the use of homogeneous ones.

Number of Workers/Nodes Similar to the previous one, the performance evaluation can also consider the number of workers. It is worth noting that a single worker can include more computational devices. Moreover, for centralized approaches, the PSs shall be evaluated separately from the other workers.

Dataset and Model Size Other two elements during the performance evaluation are the dataset and the model size. The first one is generally expressed in terms of rows for tabular data and number of images for imaging dataset. Some of popular big data dataset that are often used for ML/DL frameworks testing and evaluation are: ImageNet (imaging, [143]), MNIST (imaging [144]) CelebA (imaging, [145]), FFHQ (imaging, [146]), MIMIC-III (imaging, signals, text [147]), TCGA dataset (clinical, molecular, and imaging [148]). The last framework can include different elements, such as number of learnable parameters, hidden layers (for ANN-based models), and maximum depth for three-based models.

Throughput The last element evaluated is the throughput, defined as the number of data units processed over in unit of time. According to the data processed, it is expressed as $Images/second$ or $Samples/seconds$.

Model Evaluation Metrics This term refers to the metrics useful to evaluate the quality of a model. In this context, the evaluation concerns the trend of loss functions over the time, as

$$NLL(\theta) = -\log(p(y|x;\theta)) \tag{3}$$

$$CE = -\sum_{c=1}^{M} y_{i,c} \log(p_{i,c}) \tag{4}$$

Where Eq.3 is *Negative Log-likelihood* and the Eq.4 is the *Cross-Entropy*. In Eq.3, p is the prediction probability of y given an observation x, with θ parametrization, while for Eq.4, M is the number of classes, log refers to the natural logarithm, y is 0 or 1 if class label c is the correct classification with respect to the observation i and p refers to the predicted probability observation i is of a class c.

These metrics, with the *model accuracy* and the *error rate* are useful for evaluating the distributed SGD which convergence, as explained in Section 5, is not guaranteed or can be very slow.

Other minor elements evaluated in a distributed scenario are:

- The number of epochs or iterations needed by a model to complete the training phase.

- The *Speedup Factor*, defined as

$$SF = T_i/T_N \tag{5}$$

  i.e., the ratio of the running time of i nodes over the running time of all N nodes in the cluster.

Table 7 classifies the works in Section 6, according to the kind of evaluation they perform on the DDL/DML framework. 7 main classes of evaluation have been identified .

Table 7: Classes of evaluation criteria adopted in the frameworks retrieved in Section 6.

N.	**Evaluation**	**Works**
1.	Computational Time w.r.t. Workers/Nodes number	[94, 98, 100, 110, 120, 112, 105]
2.	Computational Time w.r.t. Dataset Size	[93, 96, 98, 66, 128]
3.	Computational Time w.r.t. Model Size	[93, 98, 105, 123]
4.	Computational Time w.r.t. Number of computational Devices	[95, 126, 105, 109, 122]
5.	Throughput w.r.t. Number of Computational Devices	[103, 105, 112, 118, 119, 66]
6.	Computational Device utilization and memory footprint trends	[96, 100, 101, 107, 118, 123, 125]
7.	Model evaluation metrics trend w.r.t. Workers/Nodes number	[100, 105, 121, 122]
8.	Model evaluation metrics trend w.r.t. Epochs or Iterations	[120, 105, 122]
9.	Model evaluation metrics trend w.r.t. Computational Time	[100, 121]

Other metrics, proposed but quite unusual, are: framework latency w.r.t. number of workers [103] or data size [118], Throughput w.r.t. model size [106], task scheduling w.r.t. dispatch [112], and data scaling w.r.t. number of cores [102]

8. Emerging Optimizations enabling DML/DDL.

In this section, we briefly report some relevant most recent contributions related to distributed learning optimization, emerging from our research. Intuitively, each optimization acts on a different aspects of a distributed learning system. Such contributions can be classified in three different areas: SGD and Parameters optimization, Communication optimization and Jobs Scheduling Optimization.

SGD and Model Parameters Optimization Among recent DAI optimizations, both the computation of SGD and the exchange of model parameter updates play an important role [149, 150, 151].

In this field, Gu *et al.* [152] optimize the computation of Data-Parallel system with PS; the authors build an optimization framework based on Apache Spark, and evaluate distributed SGD-based algorithms, such as Elastic Averaging SGD (EASGD). For EASGD, they achieved a convergence gain, resulting in an increasing of 5.7% speed with respect to other distributed SGD-based optimization algorithms.

Alternatively, the *sketch* based methods [153] can be used for gradient compression as done for *SketchML* [154], in which the authors exploit the sketch method to compress sparse and non-uniform gradient in key-value pairs, achieving a 10× speed with respect to the existing methods.

Wu *et al.* [155] proposed a sign-bit compression-based learning synchronization framework, called Marsit, preventing the *Time Consumption* and *Performance Deterioration* of cascade gradient compression [149]; Marsit reduced the training time up to 35%, maintaining the same accuracy of uncompressed methods.

GRACE framework [156] with TensorFlow and PyTorch API, implements 16 compression methods falling in the following classes: quantization, sparsification, adaptive [157] and Deep Grandient Compression (DGC) [158].

An alternative to gradient compression is the *in-network aggregation*: during the training phase, the updates coming from each worker propagate across the network reaching a specific node where an aggregation operation is performed. Then, only the aggregated updates are

further propagated. Grounding on this approach, Sapio et al. propose *SwitchML* [159], a method for aggregating the model updates exchanged for communication optimization and, consequently, speeding up the training process; their approach achieved a speed up to 5.5× with respect to number of real-world benchmark models. A similar approach is proposed by Zhao et al.[160], with Select Neighbors and Parameters framework (SNAP) for edge computing. In particular, the edge servers update their replica of the global model, with the weighted sum of the parameters exchanged with their neighbours; as further optimization, only the weights significantly changed are sent to the neighbour. Moreover, the matrix weight is customized according to the network topology, increasing the convergence rate.

A summary of above optimizations is reported in Table 8.

Table 8: Summary of Distributed Learning optimization related to SGD computation and model parameters.

Authors and Work	**Description**	**Optimization**	**Results**
R. Gu et al. [152]	Optimization framework for Data Parallel system with PS based on Apache Spark.	SGD Parallelization	+5.7% speed w.r.t. to other distributed SGD-based optimization algorithms
J. Jiang et al. [154]	SketchML, a sketch-based method to compress sparse and non-uniform gradient in key-value pairs.	Gradient Compression	10x speed up w.r.t. othe compression methods.
Wu et al. [155]	Marsit, a sign-bit compression method preventing the perfomance deterioration due to cascade gradient compression.	Gradient Compression	35% training time reduction, retaining the same accuracy of uncompressed methods.
H. Xu et al. [156]	GRACE, a PyTorch/TensorFlow based framework, implementing quantization, sparsification, adaptive and Deep grandient compression methods.	Gradient Compression	Survey on 16 different gradient compression methods, comparing them in terms of accuracy, throughput and communication volume.
Sapio et al. [159]	SwitchML, a method for aggregating the model updates exchanged for communication optimization. Implemetation based on Horovod.	In-network aggregation	5.5x speed up, w.r.t. other real-world benchmark models
Y. Zhao et al. [160]	SNAP framework for computing and exchanging the weighted sum of the model parameters between neighbours. Matrix weight customized according to the network topology.	In-network aggregation, Matrix weight customization.	Communication cost reduced by 99.6%, w.r.t. TernGrad.

Communication Optimization A particular attention has been payed to

network traffic, or in general to communication optimization [161]. Yokoyama and Araki [162] proposed a communication method for decreasing the redundant communication, reducing the network traffic . In particular, they identified the redundant communications and replaced them using a direct communication scheme, reducing the number of redundant messages exchanged. Their implementation is based on MPI and a customized version of PS called *Symmetric Node Composition*.

Sandha *et al.* in [163] demonstrated that embedding DML in a database engine reduces the communication overhead, w.r.t. frameworks moving data from a database to an analytical engine.

Another communication optimization is given by *Timed Dataflow* (TDF) [164]. In TDF with PS configuration, the network traffic is reduced by implementing a timed parameter storage system and a hybrid parameter filter. The objective is discarding the unchanged parameters during the pull operation and dropping the gradients during the pushing operation, according to a selection criteria. TDF achieved a significant network traffic reduction (between 77% to 79%).

Duang *et al.* proposed two graph partition heuristics for data and parameters partition, aiming at minimizing the total training time [165]. The two approaches have been implemented with PS configuration, showing that the communication efficiency is improved up to 14 times with respect to random partitioning.

Table 9 reports a summary of workers communication optimization methods.

Table 9: Summary of optimization related to workers communication.

Authors and Work	Description	Optimization	Results
Yokoyama and Araki [162]	Identification of redundant communications and replacement using a direct communication scheme.	Direct Communication scheme, Symmetric Node Composition	Shift from $2n_m + 2n_d$ to $2n_m + n_d$ where n_m and n_d are the elements in reduced communication and direct transmission respectively.
Sandha et al. [163]	Running distributed machine learning processes inside a database engine.	Integration of ML process inside a DB engine	*NA, only implementation.*
Sun et al. [164]	Implementation of a timed parameter storage system for discharging the unchanged parameters and a hybrid gradient filter for dropping the gradients during according to a selection criteria.	Timed Dataflow, with hybrid gradient filter	Network traffic reduction (between 77% to 79%).
Duang et al. [165]	Graph partition heuristics for data and parameters partition respectively, for training time minimization.	Heuristic Data Partitioning, Heuristic Parameters Partitiong	Communication efficiency improved up to 14x , w.r.t. random partitioning.
Sapio et al. [159]	SwitchML, a method for aggregating the model updates exchanged for communication optimization. Implemetation based on Horovod.	In-network aggregation	5.5x speed up, w.r.t. other real-world benchmark models
Y. Zhao et al. [160]	SNAP framework for computing and exchanging the weighted sum of the model parameters between neighbours. Matrix weight customized according to the network topology.	In-network aggregation, Matrix weight custom.	Communication cost reduced by 99.6%, w.r.t. TernGrad.

Job Scheduling Optimization There are several works aiming at optimizing the workload balancing. Bao *et al.* proposed *Harmony* [166], a Deep Reinforcement Learning (DRL) based framework that can be employed in either PS or AllReduce architecture. Specifically, Harmony encodes the workload interference (defined as the resources contention) in a Neural Network, mapping job and cluster states into job placement decision; moreover, it uses DRL to take near-optimal decisions according to run-time information without any kind of mathematical decision model or heuristic. Also Lu and Wang implemented a DRL framework, called Distributed Actor-critic Reinforcement Learning (DARL) [167]. DARL is aimed at improving cluster resources utilization, by dynamically adapting the single worker training load according to the cluster status without parameter settings.

For DML tasks in Edge-Cloud Networks, the classical scheduling algorithms cannot be easily employed, due to the peculiarities of such kind of networks (*e.g.*, low-latency and scarcity of resources, high delay and rich computing capacity, frequent communications between server and workers, and so on). To overcome this obstacle, Zhou *et al.* designed

and implemented *Dynamic Pricing and Scheduling* (DPS) framework [168] composed by a job control, a cost function design (price function) and a scheduling orchestrator. DPS is able to dynamically assign a score (price) to PSs and workers, according to the current system status and historical information. Moreover, for a new job, the framework computes a deployment cost and, depending on this, it decides whether accept the job or not. Another adaptive scheduling framework is proposed by Zhou et al. [169]: a resource detection system is implemented and, according to the data retrieved by the detector, the task scheduling system dynamically changes the environmental parameters and schedules calculations.

To address the problems related to job scheduling in clusters with heterogeneous workers while minimizing the execution time, another scheduling algorithm has been designed [170]. In this work, the authors propose an online job scheduling algorithm (for both PS and AllReduce configuration) that chooses the time window, the number of concurrent workers, and the PSs related to a job execution, while trying to minimize the average execution time. The algorithm is composed by a module for grouping unprocessed ML jobs in batches and a job-batch scheduling algorithm; results show that it outperforms the state-of-art AI jobs schedulers.

Mahajan et al. proposed THEMIS [171] a fairness scheduling framework for ML tasks in GPU clusters. The THEMIS scheduling is based on *finish-time fairness* metric, formally defined as $T_{shared}/T_{independent}$, *i.e.* the ratio of a ML task running time in a shared cluster with other N tasks (*shared finish time*, T_{shared}) and the running time related to the same task but without other concurrency tasks in the cluster (*independent finish time*, $T_{independent}$). Implementing a cross-app and multiple-app scheduler called *ARBITER*, THEMIS improves the fairness of 2.25× and is from 5% to 250% more efficient with respect to state-of-the-art schedulers.

Table 10 shows a summary of the most recent optimization methods found.

Table 10: Summary of optimization methods related to Job Scheduling.

Authors and Work	Description	Optimization	Results
Bao et al. [166]	Harmony, a DRL based scheduling framework, for enconding the workload interference, mapping job and cluster states into job placement decision.	Workload interference encoding, near-optimal decisionsbased on DRL	16% - 42% speed up of average job completion time.
Lu and Wang [167]	DARL, a framework for improving cluster resources utilization, by dynamically adapt the single worker training load according to the cluster status without parameter settings.	Dynamical worker adaptation based on DRL	DARL outperforms the BSP scheme by 57.8% and SSP by 50.3% in terms of per-iteration time.
R. Zhou et al. [168]	Dynamic Pricing and Scheduling framework composed by a job control, a cost function design and a scheduling orchestrator. DPS assignes dynamically a score to PSs and workers, according to the current system status and historical information.	Dynamic Pricing and Scheduling framework	Social welfare improved by at least 95%.
X. Zhou et al. [169]	Implementation of resource detection system, task scheduling system dynamically changes the environmental parameters and schedules calculations.	Adaptive Scheduling framework (ASF), with resources detection	ASF achieves a better accuracy, w.r.t. BSP when the number of parallel processes is less than 5. It presents also a good scalability, robustness and adaptability.
R. Zhou et al. [170]	Online job scheduling algorithm, choosing the time window, the number of concurrent workers and PSs, for minimizing the average execution time.	Online Job Scheduling	The online scheduling algorithm outperforms other scheduling algorithms by 30% in case of PS and AllReduce.
Mahajan et al. [171]	The THEMIS scheduling, based on finish-time fairness metric. Implementation of a cross-app and multiple-app scheduler (ARBITER),.	Scheduling based on finish-time fairness metric, Cross-app and Multiple-app Scheduler.	Fairness improved by 2.25× and efficiency improved up to 5% to 250% w.r.t. state-of-the-art schedulers.

9. Conclusion

In this work we present a survey on DML and DDL frameworks for Big Data Analytics , with related distributed computation algorithms and strategies. We conducted such a survey by introducing the main challenges related to Big Data analytics, answering to specific Research Questions explicitly defined.

In order to find appropriate answers, we queried SCOPUS database, retrieving several works. Then, we filtered them according to explicit Inclusion/Exclusion Criteria, defined in Appendix B. After examining the technical aspects behind this branch of distributed artificial intelligence, we selected 13 DML and 20 DDL frameworks, respectively. Then, we compared them according to the following criteria: implemented parallelization technique, supporting device, supported architecture, communication mode, working mode and class of algorithms.

Additionally, we investigated evaluation criteria and metrics proposed in

the analyzed frameworks, to give an overview of the evaluation trend in this field.

With reference to DML, Apache Spark results one of the most popular frameworks, often employed as back-end for other frameworks, due to its capabilities in processing with high speed data batches and data streams.

As concerns parallelization, data parallelism is implemented in all frameworks, since it is the baseline for distributing computation. Model parallelization (in its proper definition, given in Section 5.1) is instead implemented in a small number of frameworks, due to the generally low complexity of ML models, not really asking for parallelization. In this context, parallelization is more often conceived as the act of training and evaluating different models at the same time, implemented in several frameworks (*e.g.* Apache Spark).

Concerning the Supporting Device, half of the frameworks support the GPU computation. In the Spark-based ones, the GPU computation can be exploited according to the adopted Spark version.

The centralized architecture is supported by all frameworks except SIREN, that has been developed specifically to be serverless. Instead, only a few frameworks support decentralized computation, probably because of its bigger management and communication complexity. Finally, both communication modes (synchronous and asynchronous) are equally supported by all frameworks.

The comparison between Table 5 and Table 3 highlights the differences between DML and DDL frameworks, in terms of identified classification criteria.

Data parallelism is again implemented in all DDL frameworks, but here also model parallelism is implemented in the majority of cases. In fact, DL models are generally more complex and can not fit one device. Only CocoNet explicitly implements the pipeline parallelism.

The generally bigger complexity of DL models is also at the basis of the higher computational power required by such models; thus, DDL frameworks often require support for GPUs computation, as shown in Table 5. We also investigated the use of TPUs as accelerator device, due to its recent adoption. As we showed, TPUs are supported only by three of the frameworks retrieved: TensorFlow, PyTorch and Chainer. In the future, TPU support might be more widely implemented, due to its promising performance in terms of computation and energy efficiency.

As for DML, in DDL scenario the centralized architecture is more supported than the decentralized one; nevertheless, the latter is implemented in

several DDL frameworks. The synchronous and asynchronous communication results equally implemented, as for DML.

Notably, all frameworks implementing model/data parallelism and synchronous/asynchronous communication mode can potentially implement hybrid strategies, investigated in Section 4.

As we discussed in the sections above, the choice of the best framework is not straightforward: several aspects must be considered, ranging from hardware configuration to analytic tasks and requirements. Although such factors seems to be not correlated, we discussed how a choice on a specific factor can affect the other ones. Among the definied classification criteria defined, we identified three priority factors in design choices by developers: cluster architecture, parallelization algorithms and supporting device.

The classification criteria we identified allow for evaluating the most frequent capabilities provided by DML and DDL frameworks. The paper also includes an overview of some recent—and thus rarely implemented up to now— optimization techniques and algorithms, mainly related to SGD and parameters optimization, Communication optimization, and Job scheduling optimization.

To the best of our knowledge, no related survey classify DML and DDL frameworks according to unified criteria, providing a comparison of their capabilities; this survey acts as a starting point for approaching the study of DML/DDL for Big Data Analytics.

Acknowledgements and Fundings

The study was partially funded under the National Recovery and Resilience Plan (NRRP), project "National Centre for HPC, Big Data and Quantum Computing – CN HPC", Mission 4: "Istruzione e Ricerca", Component 2: "Dalla ricerca all'impresa", Investment 1.4: "Potenziamento strutture di ricerca e creazione di 'campioni nazionali di R&S' su alcune Key Enabling Technologies", CUP: D93C22000430001, funded by European Union-NextGenerationEU.

Appendix A. List of Queries performed

1. TITLE-ABS-KEY(("big data" AND NOT("machine learning" OR "deep learning")) AND (limit-to(pubyear,2021) OR
limit-to(pubyear,2020) OR limit-to(pubyear,2019) OR limit-to(pubyear,2018) OR limit-to(pubyear,2017) OR limit-to(pubyear,2016) OR limit-to(pubyear,2015))

2. TITLE-ABS-KEY(("machine learning" AND NOT("big data" OR "deep learning")) AND (limit-to(pubyear,2022) OR limit-to(pubyear,2021) OR limit-to(pubyear,2020) OR limit-to(pubyear,2019) OR limit-to(pubyear,2018) OR limit-to(pubyear,2017) OR limit-to(pubyear,2016) OR limit-to(pubyear,2015))

3. TITLE-ABS-KEY(("deep learning" AND NOT("big data" OR "machine learning")) AND (limit-to(pubyear,2022) OR limit-to(pubyear,2021) OR limit-to(pubyear,2020) OR limit-to(pubyear,2019) OR limit-to(pubyear,2018) OR limit-to(pubyear,2017) OR limit-to(pubyear,2016) OR limit-to(pubyear,2015))

4. TITLE-ABS-KEY(("big data" AND ("deep learning" OR "machine learning")) AND (limit-to(pubyear,2022) OR limit-to(pubyear,2021) OR limit-to(pubyear,2020) OR limit-to(pubyear,2019) OR limit-to(pubyear,2018) OR limit-to(pubyear,2017) OR limit-to(pubyear,2016) OR limit-to(pubyear,2015))

5. Documents about "Distributed Artificial Intelligence for Big Data" are retrieved by merging the results of the following queries:
   - TITLE-ABS-KEY(("Big Data" AND ("distributed machine learning" OR "distributed deep learning"))
   - TITLE-ABS-KEY(("distributed deep learning framework") OR ("distributed machine learning framework")
   - TITLE-ABS-KEY((("distributed deep learning" OR distributed machine learning") AND ("review" OR "survey"))
   - TITLE-ABS-KEY(("distributed machine learning" AND "Big Data"))
   - TITLE-ABS-KEY(("distributed deep learning" AND "Big Data"))

   Also these queries have been limited from 2015 to 2022.

Appendix B. Research methodology details

Table B.11 shows the number of documents retrieved for each topic (query).

Table B.11: List of queries performed on SCOPUS, with related numbers of retrieved documents. Topic 5 refers only to DML and DDL research solutions.

Query (identifier)	Topic	Returned Papers (number)
1	Big Data	101,044
2	Machine learning	275,678
3	Deep learning	173,717
4	Big Data and (Machine Learning or Deep Learning)	22,417
5	Distributed Artificial Intelligence for Big Data	636

We report in Figure B.7, the temporal trends of retrieved documents for each query.

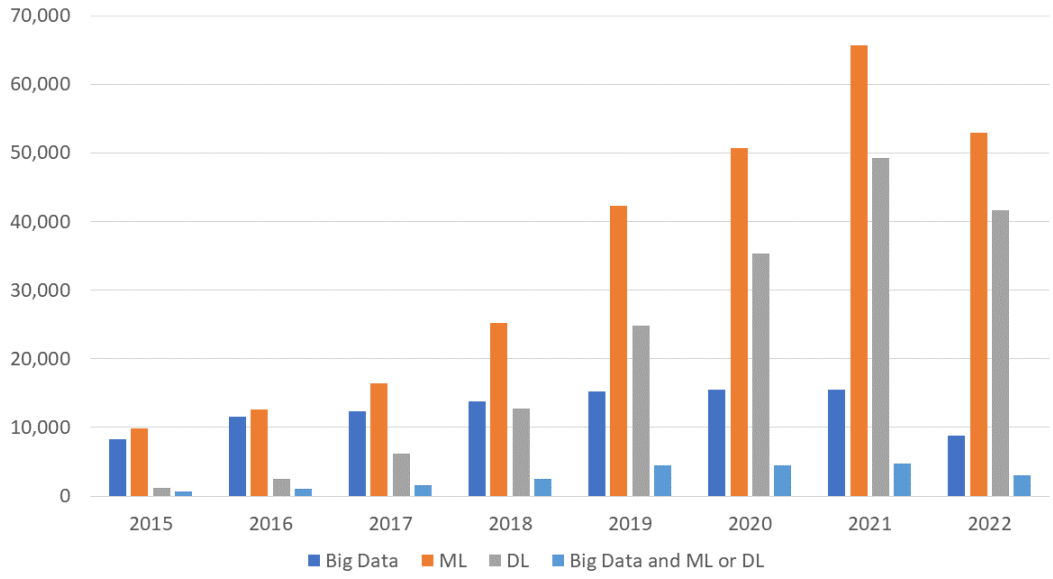

Figure B.7: Trends of researches in Big Data, Machine Learning (ML), Deep Learning(DL), Big Data and (Machine Learning or Deep Learning).

Figure B.7 shows an increasing interest in all queried topics in the last years. The most addressed topics are *Machine learning*, *Deep Learning*, and *Big Data*, with numbers of retrieved papers in the order of hundred of thousands, followed by *Big Data AND Machine Learning AND Deep learning*. Comparatively, *Distributed Artificial Intelligence for Big Data* is a niche topic; nevertheless, its trend follows the one of the other topics, showing an increasing interest also in this field.

Table B.11 shows that 636 works are retrieved with Query 5. Thus, some selection criteria have been adopted for the fifth topic: *Distributed Artificial Intelligence for Big Data* and shown in Table B.12.

Table B.12: List of selection criteria for research works about *Distributed Artificial Intelligence for Big Data*. IC stands for Inclusion Criterion, EC for Exclusion Criterion.

Criterion	Description
IC	The work includes theoretical elements about parallelization techniques, Stochastic Gradient Descent (SGD) computation algorithm in distributed environments, model parameter exchange methods and protocols.
IC	The work includes a detailed description about DML or DDL framework implementation and structure.
IC	The work includes DML or DDL framework test, preferably with Design of Experiment (DoE) methodology.
EC	The work is not related to DML or DDL.
EC	The work includes few specific applications of DML or DDL framework.
EC	The work refers to a legacy framework.
EC	The work refers to another similar study, but it is a previous version or not-extended version.

References

[1] A. De Mauro, M. Greco, M. Grimaldi, What is big data? A consensual definition and a review of key research topics, AIP Conference Proceedings 1644 (1) (2015) 97. doi:10.1063/1.4907823.
URL https://aip.scitation.org/doi/abs/10.1063/1.4907823

[2] De-shuanghuang, Radial basis probabilistic neural networks: Model and application, International Journal of Pattern Recognition and Artificial Intelligence 13 (11 2011). doi:10.1142/S0218001499000604.

[3] D.-S. Huang, X. Du, A constructive hybrid structure optimization methodology for radial basis probabilistic neural networks, IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council 19 (2009) 2099–115. doi:10.1109/TNN.2008.2004370.

[4] C.-Y. Lu, D.-S. Huang, Optimized projections for sparse representation based classification, Neurocomputing 113 (2013) 213–219.

doi:https://doi.org/10.1016/j.neucom.2013.01.009.
URL https://www.sciencedirect.com/science/article/pii/S0925231213001677

[5] S.-D. M. De-Shuang Huang, Linear and nonlinear feedforward neural network classifiers: A comprehensive understanding, Journal of Intelligent Systems 9 (1) (1999) 1–38 [cited 2023-03-01]. doi:doi:10.1515/JISYS.1999.9.1.1.
URL https://doi.org/10.1515/JISYS.1999.9.1.1

[6] F. Han, D.-S. Huang, A new constrained learning algorithm for function approximation by encoding a priori information into feedforward neural networks, Neural Computing and Applications 17 (2008) 433–439. doi:10.1007/s00521-007-0135-5.

[7] Z.-Q. Zhao, D.-S. Huang, A mended hybrid learning algorithm for radial basis function neural networks to improve generalization capability, Applied Mathematical Modelling 31 (7) (2007) 1271–1281. doi:https://doi.org/10.1016/j.apm.2006.04.014.
URL https://www.sciencedirect.com/science/article/pii/S0307904X06000965

[8] J.-X. Du, D.-S. Huang, G.-J. Zhang, Z.-F. Wang, A novel full structure optimization algorithm for radial basis probabilistic neural networks, Neurocomputing 70 (1) (2006) 592–596, neural Networks. doi:https://doi.org/10.1016/j.neucom.2006.05.003.
URL https://www.sciencedirect.com/science/article/pii/S0925231206001366

[9] X.-F. Wang, D.-S. Huang, A novel density-based clustering framework by using level set method, IEEE Transactions on Knowledge and Data Engineering 21 (11) (2009) 1515–1531. doi:10.1109/TKDE.2009.21.

[10] N. Altini, A. Brunetti, S. Mazzoleni, F. Moncelli, I. Zagaria, B. Prencipe, E. Lorusso, E. Buonamico, G. E. Carpagnano, D. F. Bavaro, M. Poliseno, A. Saracino, A. Schirinzi, R. Laterza, F. Di Serio, A. D'introno, F. Pesce, V. Bevilacqua, Predictive machine learning models and survival analysis for covid-19 prognosis based on hematochemical parameters, Sensors 21 (24), cited by: 7; All Open Access, Gold Open Access, Green Open Access (2021).

doi:10.3390/s21248503.
URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85121418218&doi=10.3390%2fs21248503&partnerID=40&md5=524f7e9a409695d41eb45cecc5ec3a1e

[11] D. E. O'Leary, Artificial intelligence and big data, IEEE Intelligent Systems 28 (2) (2013) 96–99. doi:10.1109/MIS.2013.39.

[12] F. Berloco, V. Bevilacqua, S. Colucci, A Systematic Review of Distributed Deep Learning Frameworks for Big Data, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 13395 LNAI (2022) 242–256. doi:10.1007/978-3-031-13832-4_21/COVER.
URL https://link.springer.com/chapter/10.1007/978-3-031-13832-4_21

[13] D. Gupta, R. Rani, A study of big data evolution and research challenges:, Journal of information Science 45 (3) (2018) 322–340. doi:10.1177/0165551518789880.
URL https://journals.sagepub.com/doi/abs/10.1177/0165551518789880

[14] T. Ben Nun, T. Hoefler, Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, ACM Computing Surveys 52 (4) (feb 2018). doi:10.48550/arxiv.1802.09941.
URL https://arxiv.org/abs/1802.09941v2

[15] E. P. Xing, Q. Ho, P. Xie, D. Wei, Strategies and Principles of Distributed Machine Learning on Big Data, Engineering 2 (2) (2015) 179–195. arXiv:1512.09295, doi:10.48550/arxiv.1512.09295.
URL https://arxiv.org/abs/1512.09295v1

[16] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, J. S. Rellermeyer, A survey on distributed machine learning, ACM Comput. Surv. 53 (2) (mar 2020). doi:10.1145/3377454.
URL https://doi.org/10.1145/3377454

[17] J. Qiu, Q. Wu, G. Ding, Y. Xu, S. Feng, A survey of machine learning for big data processing, Eurasip Journal on Advances in Signal Processing 2016 (1) (2016) 1–16.

doi:10.1186/S13634-016-0355-X/FIGURES/5.
URL https://asp-eurasipjournals.springeropen.com/articles/10.1186/s13634-016-0355-x

[18] D. Otoo-Arthur, T. Van Zyl, A systematic review on big data analytics frameworks for higher education - Tools and algorithms, ACM International Conference Proceeding Series (2019) 79–87doi:10.1145/3377817.3377836.
URL https://www.researchgate.net/publication/340057491_A_Systematic_Review_on_Big_Data_Analytics_Frameworks_for_Higher_Education_-_Tools_and_Algorithms

[19] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, E. Mephu Nguifo, An experimental survey on big data frameworks, Future Generation Computer Systems 86 (2018) 546–564. arXiv:1610.09962, doi:10.1016/J.FUTURE.2018.04.032.

[20] K. Zhang, S. Alqahtani, M. Demirbas, A comparison of distributed machine learning platforms, 2017 26th International Conference on Computer Communications and Networks, ICCCN 2017 (sep 2017). doi:10.1109/ICCCN.2017.8038464.

[21] H. Wang, Z. Qu, Q. Zhou, H. Zhang, B. Luo, W. Xu, S. Guo, R. Li, A Comprehensive Survey on Training Acceleration for Large Machine Learning Models in IoT, IEEE Internet of Things Journal 9 (2) (2022) 939–963. doi:10.1109/JIOT.2021.3111624.

[22] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, Journal of Systems and Software 80 (4) (2007) 571–583. doi:10.1016/J.JSS.2006.07.009.

[23] I. Hamdaoui, M. El Fissaoui, K. El Makkaoui, Z. El Allali, Hadoop-Based Big Data Distributions: A Comparative Study, Lecture Notes on Data Engineering and Communications Technologies 147 (2023) 242–252. doi:10.1007/978-3-031-15191-0_24/COVER.
URL https://link.springer.com/chapter/10.1007/978-3-031-15191-0_24

[24] V. Janev, Semantic Intelligence in Big Data Applications, Smart Connected World (2021) 71–89arXiv:2107.03853, doi:10.1007/978-3-030-76387-9_4/COVER.
URL https://link.springer.com/chapter/10.1007/978-3-030-76387-9_4

[25] M. Chen, S. Mao, Y. Liu, Big data: A survey, Mobile Networks and Applications 19 (2) (2014) 171–209. doi:10.1007/S11036-013-0489-0/FIGURES/6.
URL https://link.springer.com/article/10.1007/s11036-013-0489-0

[26] M. Al-Mekhlal, A. Ali Khwaja, A synthesis of big data definition and characteristics, in: 2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), 2019, pp. 314–322. doi:10.1109/CSE/EUC.2019.00067.

[27] A. L. Samuel, Some studies in machine learning using the game of checkers, IBM Journal of Research and Development 3 (3) (1959) 210–229. doi:10.1147/rd.33.0210.

[28] Z.-H. Zhou, Machine learning, Springer Nature, 2021.

[29] D.-S. Huang, Systematic theory of neural networks for pattern recognition, Publishing House of Electronic Industry of China, Beijing 201 (1996).

[30] S. Ray, A quick review of machine learning algorithms, in: 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), 2019, pp. 35–39. doi:10.1109/COMITCon.2019.8862451.

[31] Z.-L. Sun, D.-S. Huang, Y.-M. Cheung, J. Liu, G.-B. Huang, Using fcmc, fvs, and pca techniques for feature extraction of multispectral images, IEEE Geoscience and Remote Sensing Letters 2 (2) (2005) 108–112. doi:10.1109/LGRS.2005.844169.

[32] Z.-Q. Zhao, D.-S. Huang, W. Jia, Palmprint recognition with 2dpca+pca based on modular neural networks, Neurocomputing 71 (2007) 448–454. doi:10.1016/j.neucom.2007.07.010.

[33] D.-S. Huang, J.-X. Mi, A new constrained independent component analysis method, IEEE Transactions on Neural Networks 18 (5) (2007) 1532–1535. doi:10.1109/TNN.2007.895910.

[34] D.-S. Huang, W.-B. Zhao, Determining the centers of radial basis probabilistic neural networks by recursive orthogonal least square algorithms, Applied Mathematics and Computation 162 (2005) 461–473. doi:10.1016/j.amc.2003.12.105.

[35] D.-S. Huang, H. Ip, K. Law, Z. Chi, Zeroing polynomials using modified constrained neural network approach, Neural Networks, IEEE Transactions on 16 (2005) 721 – 732. doi:10.1109/TNN.2005.844912.

[36] F. Han, Q.-H. Ling, D.-S. Huang, An improved approximation approach incorporating particle swarm optimization and a priori information into neural networks, Neural Comput. Appl. 19 (2) (2010) 255–261. doi:10.1007/s00521-009-0274-y.
URL https://doi.org/10.1007/s00521-009-0274-y

[37] D.-S. Huang, A constructive approach for finding arbitrary roots of polynomials by neural networks, IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council 15 (2004) 477–91. doi:10.1109/TNN.2004.824424.

[38] F. Han, Q.-H. Ling, D.-S. Huang, Modified constrained learning algorithms incorporating additional functional constraints into neural networks, Information Sciences 178 (3) (2008) 907–919, including Special Issue "Ambient Intelligence". doi:https://doi.org/10.1016/j.ins.2007.09.008.
URL https://www.sciencedirect.com/science/article/pii/S0020025507004276

[39] N. Altini, A. Brunetti, E. Puro, M. G. Taccogna, C. Saponaro, F. A. Zito, S. De Summa, V. Bevilacqua, Ndg-cam: Nuclei detection in histopathology images with semantic segmentation networks and grad-cam, Bioengineering 9 (9), cited by: 6; All Open Access, Gold Open Access, Green Open Access (2022). doi:10.3390/bioengineering9090475.
URL https://www.scopus.com/inward/record.uri?eid=

2-s2.0-85138664687&doi=10.3390%2fbioengineering9090475&partnerID=40&md5=e9d580ebbe619f378a524e7fd97f85de

[40] I. J. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, Cambridge, MA, USA, 2016, http://www.deeplearningbook.org.

[41] N. Altini, B. Prencipe, G. D. Cascarano, A. Brunetti, G. Brunetti, V. Triggiani, L. Carnimeo, F. Marino, A. Guerriero, L. Villani, A. Scardapane, V. Bevilacqua, Liver, kidney and spleen segmentation from ct scans and mri with deep learning: A survey, Neurocomputing 490 (2022) 30–53. doi:https://doi.org/10.1016/j.neucom.2021.08.157.
URL https://www.sciencedirect.com/science/article/pii/S0925231222003149

[42] Z.-Q. Zhao, H. Glotin, D. Xie, J. Gao, X. Wu, Cooperative sparse representation in two opposite directions for semi-supervised image annotation, IEEE transactions on image processing : a publication of the IEEE Signal Processing Society 21 (2012) 4218–31. doi:10.1109/TIP.2012.2197631.

[43] Z.-L. Sun, D.-S. Huang, Y.-m. Cheung, Extracting nonlinear features for multispectral images by fcmc and kpca, Digital Signal Processing 15 (2005) 331–346. doi:10.1016/j.dsp.2004.12.004.

[44] J.-X. Mi, D.-S. Huang, B. Wang, X. Zhu, The nearest-farthest subspace classification for face recognition, Neurocomputing 113 (2013) 241–250. doi:https://doi.org/10.1016/j.neucom.2013.01.003.
URL https://www.sciencedirect.com/science/article/pii/S0925231213001434

[45] J.-G. Lee, S. Jun, Y. Cho, H. Lee, G. Kim, J. B. Seo, N. Kim, Deep learning in medical imaging: General overview, Korean Journal of Radiology 18 (2017) 570. doi:10.3348/kjr.2017.18.4.570.

[46] Y. Li, C. Huang, L. Ding, Z. Li, Y. Pan, X. Gao, Deep learning in bioinformatics: Introduction, application, and perspective in the big data era, Methods (04 2019). doi:10.1016/j.ymeth.2019.04.008.

[47] N. Altini, G. De Giosa, N. Fragasso, C. Coscia, E. Sibilano, B. Prencipe, S. M. Hussain, A. Brunetti, D. Buongiorno, A. Guerriero, I. S. Tatò,

G. Brunetti, V. Triggiani, V. Bevilacqua, Segmentation and identification of vertebrae in ct scans using cnn, k-means clustering and k-nn, Informatics 8 (2) (2021). doi:10.3390/informatics8020040.
URL https://www.mdpi.com/2227-9709/8/2/40

[48] J. Zhang, D.-S. Huang, T.-M. Lok, M. R. Lyu, A novel adaptive sequential niche technique for multimodal function optimization, Neurocomputing 69 (16) (2006) 2396–2401, brain Inspired Cognitive Systems. doi:https://doi.org/10.1016/j.neucom.2006.02.016.
URL https://www.sciencedirect.com/science/article/pii/S0925231206001378

[49] J.-X. Du, D.-S. Huang, X.-F. Wang, X. Gu, Shape recognition based on neural networks trained by differential evolution algorithm, Neurocomputing 70 (4) (2007) 896–903, advanced Neurocomputing Theory and Methodology. doi:https://doi.org/10.1016/j.neucom.2006.10.026.
URL https://www.sciencedirect.com/science/article/pii/S0925231206002967

[50] N. Altini, G. D. Cascarano, A. Brunetti, I. De Feudis, D. Buongiorno, M. Rossini, F. Pesce, L. Gesualdo, V. Bevilacqua, A deep learning instance segmentation approach for global glomerulosclerosis assessment in donor kidney biopsies, Electronics 9 (11) (2020). doi:10.3390/electronics9111768.
URL https://www.mdpi.com/2079-9292/9/11/1768

[51] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM 60 (6) (2017) 84–90. doi:10.1145/3065386.
URL https://doi.org/10.1145/3065386

[52] H. GM, M. K. Gourisaria, M. Pandey, S. S. Rautaray, A comprehensive survey and analysis of generative models in machine learning, Computer Science Review 38 (2020) 100285. doi:https://doi.org/10.1016/j.cosrev.2020.100285.
URL https://www.sciencedirect.com/science/article/pii/S1574013720303853

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 30, Curran Associates, Inc., 2017.
URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[54] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16x16 words: Transformers for image recognition at scale (2020). doi:10.48550/ARXIV.2010.11929.
URL https://arxiv.org/abs/2010.11929

[55] W. Dai, Y. Zhou, N. Dong, H. Zhang, E. P. Xing, Toward Understanding the Impact of Staleness in Distributed Machine Learning, 7th International Conference on Learning Representations, ICLR 2019 (oct 2018). arXiv:1810.03264, doi:10.48550/arxiv.1810.03264.
URL https://arxiv.org/abs/1810.03264v1

[56] W. D. Hillis, G. L. Steele, Data parallel algorithms, Communications of the ACM 29 (12) (1986) 1170–1183. doi:10.1145/7902.7903.
URL https://dl.acm.org/doi/abs/10.1145/7902.7903

[57] L. Zhao, R. Xu, T. Wang, T. Tian, X. Wang, W. Wu, C. in Ieong, X. Jin, Bapipe: Exploration of balanced pipeline parallelism for dnn training, ArXiv abs/2012.12544 (2020).

[58] T. Geng, T. Wang, A. Sanaullah, C. Yang, R. Xu, R. Patel, M. Herbordt, FPDeep: Acceleration and Load Balancing of CNN Training on FPGA Clusters, Proceedings - 26th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2018 (2018) 81–84doi:10.1109/FCCM.2018.00021.

[59] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, Z. Chen, GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism, Advances in Neural Information Processing Systems 32 (2019).

[60] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, M. Zaharia, Pipedream: Generalized pipeline parallelism for dnn training, in: Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 1–15. doi:10.1145/3341301.3359646.
URL https://doi.org/10.1145/3341301.3359646

[61] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, A. Ng, Large Scale Distributed Deep Networks, Advances in Neural Information Processing Systems 25 (2012).

[62] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks (apr 2014). arXiv:1404.5997, doi:10.48550/arxiv.1404.5997.
URL https://arxiv.org/abs/1404.5997v2

[63] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, B.-Y. Su, Scaling Distributed Machine Learning with the Parameter Server, in: Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, OSDI'14, USENIX Association, USA, 2014, pp. 583–598.

[64] W. Zhao, D. Xie, R. Jia, Y. Qian, R. Ding, M. Sun, P. Li, Distributed Hierarchical GPU Parameter Server for Massive Scale Deep Learning Ads Systems (mar 2020). arXiv:2003.05622, doi:10.48550/arxiv.2003.05622.
URL https://arxiv.org/abs/2003.05622v1

[65] C. Sun, Y. Zhang, W. Yu, R. Zhang, M. Z. A. Bhuiyan, J. Li, DPS: A DSM-based parameter server for machine learning, Proceedings - 14th International Symposium on Pervasive Systems, Algorithms and Networks, I-SPAN 2017, 11th International Conference on Frontier of Computer Science and Technology, FCST 2017 and 3rd International Symposium of Creative Computing, ISCC 2017 2017-November (2017) 20–27. doi:10.1109/ISPAN-FCST-ISCC.2017.48.

[66] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, Y. Yu, Petuum: A New Platform for Distributed Machine

Learning on Big Data, IEEE Transactions on Big Data 1 (2) (2015) 49–67. doi:10.1109/TBDATA.2015.2472014.

[67] Z. Song, Y. Gu, Z. Wang, G. Yu, DRPS: efficient disk-resident parameter servers for distributed machine learning, Frontiers of Computer Science 16 (4) (2022) 1–12. doi:10.1007/S11704-021-0445-2/METRICS.
URL https://link.springer.com/article/10.1007/s11704-021-0445-2

[68] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Gossip algorithms: Design, analysis and applications, Proceedings - IEEE INFOCOM 3 (2005) 1653–1664. doi:10.1109/INFCOM.2005.1498447.

[69] Y. Jiang, H. Gu, Y. Lu, X. Yu, 2D-HRA: Two-dimensional hierarchical ring-based all-reduce algorithm in large-scale distributed machine learning, IEEE Access 8 (2020) 183488–183494. doi:10.1109/ACCESS.2020.3028367.
URL https://www.researchgate.net/publication/347269049_2D-HRA_Two-Dimensional_Hierarchical_Ring-Based_All-Reduce_Algorithm_in_Large-Scale_Distributed_Machine_Learning

[70] P. Patarasuk, X. Yuan, Bandwidth efficient all-reduce operation on tree topologies, in: 2007 IEEE International Parallel and Distributed Processing Symposium, 2007, pp. 1–8. doi:10.1109/IPDPS.2007.370405.

[71] C. Yang, A. Amazon, Tree-based allreduce communication on mxnet, Tech. Rep. (2018).

[72] P. Patarasuk, X. Yuan, Bandwidth optimal all-reduce algorithms for clusters of workstations, Journal of Parallel and Distributed Computing 69 (2) (2009) 117–124. doi:https://doi.org/10.1016/j.jpdc.2008.09.002.
URL https://www.sciencedirect.com/science/article/pii/S0743731508001767

[73] Y. Liu, J. Zhang, S. Liu, Q. Wang, W. Dai, R. C. C. Cheung, Scalable Fully Pipelined Hardware Architecture for In-Network Aggregated AllReduce Communication, IEEE Transactions on Circuits and Systems I: Regular Papers 68 (10) (2021) 4194–4206. doi:10.1109/TCSI.2021.3098841.

[74] G. Pellegrini, A. Tibo, P. Frasconi, A. Passerini, M. Jaeger, Learning aggregation functions (2020). doi:10.48550/ARXIV.2012.08482.
URL https://arxiv.org/abs/2012.08482

[75] X. Lian, C. Zhang, H. Zhang, C. J. Hsieh, W. Zhang, J. Liu, Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent, Advances in Neural Information Processing Systems 2017-December (2017) 5331–5341. arXiv:1705.09056, doi:10.48550/arxiv.1705.09056.
URL https://arxiv.org/abs/1705.09056v5

[76] CNTK, https://github.com/microsoft/CNTK, accessed: 2023-01-20.

[77] S. Zhang, A. Choromanska, Y. LeCun, Deep learning with elastic averaging sgd (2014). doi:10.48550/ARXIV.1412.6651.
URL https://arxiv.org/abs/1412.6651

[78] L. G. Valiant, A bridging model for parallel computation, Communications of the ACM 33 (8) (1990) 103–111. doi:10.1145/79173.79181.
URL https://dl.acm.org/doi/10.1145/79173.79181

[79] Q. Ho, J. Cipar, H. Cui, J. K. Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, E. P. Xing, More effective distributed ml via a stale synchronous parallel parameter server, in: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1, NIPS'13, Curran Associates Inc., Red Hook, NY, USA, 2013, p. 1223–1231.

[80] H. Shi, Y. Zhao, B. Zhang, K. Yoshigoe, A. V. Vasilakos, A free stale synchronous parallel strategy for distributed machine learning, ACM International Conference Proceeding Series (2019) 23–29doi:10.1145/3341620.3341625.
URL https://dl.acm.org/doi/10.1145/3341620.3341625

[81] R. Yang, J. Zhang, J. Wan, L. Zhou, J. Shen, Y. Zhang, Z. Wei, J. Zhang, J. Wang, Parameter Communication Consistency Model for Large-Scale Security Monitoring Based on Mobile Computing, IEEE Access 7 (2019) 171884–171897. doi:10.1109/ACCESS.2019.2956632.

[82] B. Recht, C. Re, S. Wright, F. Niu, Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, in: J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 24, Curran Associates, Inc., 2011.
URL https://proceedings.neurips.cc/paper/2011/file/218a0aefd1d1a4be65601cc6ddc1520e-Paper.pdf

[83] H. Zhang, C.-J. Hsieh, V. Akella, HogWild++: A New Mechanism for Decentralized Asynchronous Stochastic Gradient Descent (2017) 629–638doi:10.1109/ICDM.2016.0074.

[84] X. Lian, W. Zhang, C. Zhang, J. Liu, Asynchronous Decentralized Parallel Stochastic Gradient Descent, 35th International Conference on Machine Learning, ICML 2018 7 (2017) 4745–4767. arXiv:1710.06952, doi:10.48550/arxiv.1710.06952.
URL https://arxiv.org/abs/1710.06952v3

[85] J. Tu, J. Zhou, D. Ren, An asynchronous distributed training algorithm based on Gossip communication and Stochastic Gradient Descent, Computer Communications 195 (2022) 416–423. doi:10.1016/J.COMCOM.2022.09.010.

[86] J. R. Norris, Markov chains, no. 2, Cambridge university press, 1998.

[87] Y. Hu, D. Niu, J. Yang, S. Zhou, Fdml: A collaborative machine learning framework for distributed features, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 2232–2240. doi:10.1145/3292500.3330765.
URL https://doi.org/10.1145/3292500.3330765

[88] C. Zhang, H. Tian, W. Wang, F. Yan, Stay fresh: Speculative synchronization for fast distributed machine learning, Proceedings - International Conference on Distributed Computing Systems 2018-July (2018) 99–109. doi:10.1109/ICDCS.2018.00020.

[89] M. Tan, W. X. Liu, J. Luo, H. Chen, Z. Z. Guo, Adaptive synchronous strategy for distributed machine learning, International Journal of Intelligent Systems 37 (12) (2022) 11713–11741.

doi:10.1002/INT.23060.
URL https://onlinelibrary.wiley.com/doi/full/10.1002/int.23060https://onlinelibrary.wiley.com/doi/abs/10.1002/int.23060https://onlinelibrary.wiley.com/doi/10.1002/int.23060

[90] H2O.ai, H2O, version 3.10.0.8 (Oct. 2016).
URL https://github.com/h2oai/h2o-3

[91] Apache spark, https://spark.apache.org/mllib/, accessed: 2023-01-10.

[92] Transmogrifai, https://docs.transmogrif.ai/en/stable/, accessed: 2022-11-30.

[93] A. Fard, A. Le, G. Larionov, W. Dhillon, C. Bear, Vertica-ml: Distributed machine learning in vertica database, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 755–768. doi:10.1145/3318464.3386137.
URL https://doi.org/10.1145/3318464.3386137

[94] E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R. M. Badia, J. Torres, T. Cortes, J. Labarta, Pycompss: Parallel computational workflows in python, The International Journal of High Performance Computing Applications 31 (1) (2017) 66–82. arXiv:https://doi.org/10.1177/1094342015594678, doi:10.1177/1094342015594678.
URL https://doi.org/10.1177/1094342015594678

[95] J. Álvarez Cid-Fuentes, S. Solà, P. Álvarez, A. Castro-Ginard, R. M. Badia, dislib: Large Scale High Performance Machine Learning in Python, in: Proceedings of the 15th International Conference on eScience, 2019, pp. 96–105.

[96] X. Tian, B. Xie, J. Zhan, Cymbalo: An efficient graph processing framework for machine learning, in: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), 2018, pp. 572–579. doi:10.1109/BDCloud.2018.00090.

[97] Variant spark, https://variantspark.readthedocs.io/en/latest/, accessed: 2022-12-10.

[98] A. Bayat, P. Szul, A. R. O'Brien, R. Dunne, B. Hosking, Y. Jain, C. Hosking, O. J. Luo, N. Twine, D. C. Bauer, VariantSpark: Cloud-based machine learning for association study of complex phenotype and large-scale genomic data, GigaScience 9 (8), giaa077 (08 2020). arXiv:https://academic.oup.com/gigascience/article-pdf/9/8/giaa077/33571442/giaa077_reviewer_3_report_original_submission.pdf, doi:10.1093/gigascience/giaa077.
URL https://doi.org/10.1093/gigascience/giaa077

[99] Y. Zhang, J. Li, C. Sun, M. Z. A. Bhuiyan, W. Yu, R. Zhang, Hotml: A dsm-based machine learning system for social networks, Journal of computational science 26 (2018) 478–487.

[100] A. Qiao, A. Aghayev, W. Yu, H. Chen, Q. Ho, G. A. Gibson, E. P. Xing, Litz: Elastic framework for High-Performance distributed machine learning, in: 2018 USENIX Annual Technical Conference (USENIX ATC 18), USENIX Association, Boston, MA, 2018, pp. 631–644.
URL https://www.usenix.org/conference/atc18/presentation/qiao

[101] H. Wang, D. Niu, B. Li, Distributed machine learning with a serverless architecture, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 1288–1296. doi:10.1109/INFOCOM.2019.8737391.

[102] A.-K. Koliopoulos, P. Yiapanis, F. Tekiner, G. Nenadic, J. Keane, A parallel distributed weka framework for big data mining using spark, 2015. doi:10.1109/BigDataCongress.2015.12.

[103] L. Mai, G. Li, M. Wagenländer, K. Fertakis, A.-O. Brabete, P. Pietzuch, KungFu: Making training in distributed machine learning adaptive, in: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), USENIX Association, 2020, pp. 937–954.
URL https://www.usenix.org/conference/osdi20/presentation/mai

[104] Apache mahout, https://mahout.apache.org//, accessed: 2023-01-10.

[105] D. Shrivastava, S. Chaudhury, D. Jayadeva, A data and model-parallel, distributed and scalable framework for training of deep networks in apache spark (2017). doi:10.48550/ARXIV.1708.05840.
URL https://arxiv.org/abs/1708.05840

[106] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.
URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[107] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[108] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM International Conference on Multimedia, MM '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 675–678. doi:10.1145/2647868.2654889.
URL https://doi.org/10.1145/2647868.2654889

[109] T. Akiba, K. Fukuda, S. Suzuki, Chainermn: Scalable distributed deep learning framework (2017). doi:10.48550/ARXIV.1710.11351.
URL https://arxiv.org/abs/1710.11351

[110] S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa, S. Saito, S. Suzuki, K. Uenishi, B. Vogel, H. Yamazaki Vincent, Chainer: A deep learning framework for accelerating the research cycle, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2019, pp. 2002–2011.

[111] S. Tokui, K. Oono, S. Hido, J. Clayton, Chainer: a next-generation open source framework for deep learning, in: Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS), 2015.
URL http://learningsys.org/papers/LearningSys_2015_paper_33.pdf

[112] J. J. Dai, Y. Wang, X. Qiu, D. Ding, Y. Zhang, Y. Wang, X. Jia, L. C. Zhang, Y. Wan, Z. Li, J. Wang, S. Huang, Z. Wu, Y. Wang, Y. Yang, B. She, D. Shi, Q. Lu, K. Huang, G. Song, Bigdl: A distributed deep learning framework for big data, in: Proceedings of the ACM Symposium on Cloud Computing, SoCC'19, Association for Computing Machinery, 2019, pp. 50–60. doi:10.1145/3357223.3362707.
URL https://arxiv.org/pdf/1804.05839.pdf

[113] Intel analytics bigdl, https://github.com/intel-analytics/BigDL/tree/branch-0.14, accessed: 2023-01-10.

[114] Intel analytics zoo, https://github.com/intel-analytics/analytics-zoo, accessed: 2023-01-10.

[115] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. Tung, Y. Wang, Z. Xie, M. Zhang, K. Zheng, Singa: A distributed deep learning platform, in: Proceedings of the 23rd ACM International Conference on Multimedia, MM '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 685–688. doi:10.1145/2733373.2807410.
URL https://doi.org/10.1145/2733373.2807410

[116] Elephas, distributed deep learning with keras and pyspark, http://maxpumperla.com/elephas/, accessed: 2022-03-22.

[117] Tensorflowonspark, https://github.com/yahoo/TensorFlowOnSpark, accessed: 2022-03-22.

[118] J. Yuan, X. Li, C. Cheng, J. Liu, R. Guo, S. Cai, C. Yao, F. Yang, X. Yi, C. Wu, H. Zhang, J. Zhao, OneFlow: Redesign the Distributed Deep Learning Framework from Scratch (oct 2021). arXiv:2110.15032, doi:10.48550/arxiv.2110.15032.
URL https://arxiv.org/abs/2110.15032v4

[119] A. Sergeev, M. D. Balso, Horovod: fast and easy distributed deep learning in TensorFlow, arXiv preprint arXiv:1802.05799 (2018).

[120] A. Khumoyun, Y. Cui, L. Hanku, Spark based distributed deep learning framework for big data applications, in: 2016 International Conference on Information Science and Communications Technologies (ICISCT), 2016, pp. 1–5. doi:10.1109/ICISCT.2016.7777390.

[121] E.-J. Lim, S.-Y. Ahn, Y.-M. Park, W. Choi, Distributed deep learning framework based on shared memory for fast deep neural network training, in: 2018 International Conference on Information and Communication Technology Convergence (ICTC), 2018, pp. 1239–1242. doi:10.1109/ICTC.2018.8539420.

[122] S. Ahn, J. Kim, E. Lim, W. Choi, A. Mohaisen, S. Kang, Shmcaffe: A distributed deep learning platform with shared memory buffer for hpc architecture, in: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 2018, pp. 1118–1128. doi:10.1109/ICDCS.2018.00111.

[123] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, arXiv preprint arXiv:1512.01274 (2015).

[124] Uber petastorm, https://github.com/uber/petastorm, accessed: 2023-01-10.

[125] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, B. Hechtman, Mesh-TensorFlow: Deep learning for supercomputers, in: Neural Information Processing Systems, 2018.

[126] Microsoft deepspeed, https://github.com/microsoft/DeepSpeed, accessed: 2023-01-10.

[127] H. Kim, J. Park, J. Jang, S. Yoon, Deepspark: A spark-based distributed deep learning framework for commodity clusters, arXiv: Learning (2016).

[128] A. Jangda, J. Huang, G. Liu, A. H. N. Sabet, S. Maleki, Y. Miao, M. Musuvathi, T. Mytkowicz, O. Saarikivi, Breaking the computation and communication abstraction barrier in distributed machine learning workloads, in: Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 402–416. doi:10.1145/3503222.3507778.
URL https://doi.org/10.1145/3503222.3507778

[129] Qubole data service, https://docs.qubole.com/en/latest/user-guide/qds.html (2011).

[130] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, Q. He, A comprehensive survey on transfer learning, Proceedings of the IEEE 109 (1) (2021) 43–76. doi:10.1109/JPROC.2020.3004555.

[131] S. Alqahtani, M. Demirbas, Performance Analysis and Comparison of Distributed Machine Learning Systems (sep 2019). arXiv:1909.02061, doi:10.48550/arxiv.1909.02061.
URL https://arxiv.org/abs/1909.02061v1

[132] K. Chandiramani, D. Garg, N. Maheswari, Performance Analysis of Distributed and Federated Learning Models on Private Data, Procedia Computer Science 165 (2019) 349–355. doi:10.1016/J.PROCS.2020.01.039.

[133] R. A. Fisher, Design of experiments, British Medical Journal 1 (3923) (1936) 554.

[134] R. A. Fisher, The Arrangement of Field Experiments (1992) 82–91doi:10.1007/978-1-4612-4380-9_8.
URL https://link.springer.com/chapter/10.1007/978-1-4612-4380-9_8

[135] J. Rodrigues, G. Vasconcelos, P. Maciel, Screening hardware and volume factors in distributed machine learning algorithms on spark: A

design of experiments (doe) based approach, Computing 103 (10 2021). doi:10.1007/s00607-021-00965-3.

[136] J. B. Rodrigues, G. C. Vasconcelos, P. R. M. Maciel, Time and cost prediction models for language classification over a large corpus on spark, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), 2020, pp. 1702–1709. doi:10.1109/SSCI47803.2020.9308299.

[137] H. Ahn, H. Kim, W. You, Performance study of spark on yarn cluster using hibench, in: 2018 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia), 2018, pp. 206–212. doi:10.1109/ICCE-ASIA.2018.8552137.

[138] H. Y. Ahn, H. Kim, W. You, Performance study of distributed big data analysis in yarn cluster, in: 2018 International Conference on Information and Communication Technology Convergence (ICTC), 2018, pp. 1261–1266. doi:10.1109/ICTC.2018.8539474.

[139] A. Ulanov, A. Simanovsky, M. Marwah, Modeling scalability of distributed machine learning, in: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), 2017, pp. 1249–1254. doi:10.1109/ICDE.2017.160.

[140] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L.-Y. Chen, B. Zhang, P. Deaville, In-memory computing: Advances and prospects, IEEE Solid-State Circuits Magazine 11 (3) (2019) 43–55. doi:10.1109/MSSC.2019.2922889.

[141] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, E. Eleftheriou, Memory devices and applications for in-memory computing, Nature Nanotechnology 2020 15:7 15 (7) (2020) 529–544. doi:10.1038/s41565-020-0655-z.
URL https://www.nature.com/articles/s41565-020-0655-z

[142] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R.

Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, D. H. Yoon, In-datacenter performance analysis of a tensor processing unit, SIGARCH Comput. Archit. News 45 (2) (2017) 1–12. doi:10.1145/3140659.3080246.
URL https://doi.org/10.1145/3140659.3080246

[143] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255.

[144] Y. LeCun, C. Cortes, MNIST handwritten digit database (2010) [cited 2016-01-14 14:24:11].
URL http://yann.lecun.com/exdb/mnist/

[145] Z. Liu, P. Luo, X. Wang, X. Tang, Deep learning face attributes in the wild, in: Proceedings of International Conference on Computer Vision (ICCV), 2015.

[146] T. Karras, S. Laine, T. Aila, A style-based generator architecture for generative adversarial networks, IEEE Transactions on Pattern Analysis & Machine Intelligence 43 (12) (2021) 4217–4228. doi:10.1109/TPAMI.2020.2970919.

[147] A. E. Johnson, T. J. Pollard, L. Shen, L. W. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, R. G. Mark, MIMIC-III, a freely accessible critical care database, Scientific Data 2016 3:1 3 (1) (2016) 1–9. doi:10.1038/sdata.2016.35.
URL https://www.nature.com/articles/sdata201635

[148] The cancer genome atlas program, https://www.cancer.gov/tcga, accessed: 2023-07-03.

[149] M. Safaryan, P. Richtarik, Stochastic Sign Descent Methods: New Algorithms and Better Theory (jul 2021).
URL https://proceedings.mlr.press/v139/safaryan21a.html

[150] F. Han, D.-S. Huang, Improved extreme learning machine for function approximation by encoding a priori information, Neurocomputing 69 (16) (2006) 2369–2373, brain Inspired Cognitive Systems. doi:https://doi.org/10.1016/j.neucom.2006.02.013.
URL https://www.sciencedirect.com/science/article/pii/S0925231206001317

[151] S. Li, Z.-H. You, H. Guo, X. Luo, Z.-Q. Zhao, Inverse-free extreme learning machine with optimal information updating, IEEE Transactions on Cybernetics 46 (5) (2016) 1229–1241. doi:10.1109/TCYB.2015.2434841.

[152] R. Gu, S. Fan, Q. Hu, C. Yuan, Y. Huang, Parallelizing machine learning optimization algorithms on distributed data-parallel platforms with parameter server, in: 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), 2018, pp. 126–133. doi:10.1109/PADSW.2018.8644533.

[153] M. Charikar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, in: Proceedings of the 29th International Colloquium on Automata, Languages and Programming, ICALP '02, Springer-Verlag, Berlin, Heidelberg, 2002, p. 693–703.

[154] J. Jiang, F. Fu, T. Yang, B. Cui, Sketchml: Accelerating distributed machine learning with data sketches, in: Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 1269–1284. doi:10.1145/3183713.3196894.
URL https://doi.org/10.1145/3183713.3196894

[155] F. Wu, S. He, S. Guo, Z. Qu, H. Wang, W. Zhuang, J. Zhang, Sign bit is enough: A learning synchronization framework for multi-hop all-reduce with ultimate compression, in: Proceedings of the 59th ACM/IEEE Design Automation Conference, DAC '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 193–198. doi:10.1145/

3489517.3530417.
URL https://doi.org/10.1145/3489517.3530417

[156] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, P. Kalnis, Grace: A compressed communication framework for distributed machine learning, in: 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), 2021, pp. 561–572. doi:10.1109/ICDCS51616.2021.00060.

[157] N. Dryden, T. Moon, S. A. Jacobs, B. Van Essen, Communication quantization for data-parallel training of deep neural networks, in: 2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC), 2016, pp. 1–8. doi:10.1109/MLHPC.2016.004.

[158] C. Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, K. Gopalakrishnan, Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training, 32nd AAAI Conference on Artificial Intelligence, AAAI 2018 (2022) 2827–2835arXiv:1712.02679, doi:10.1609/aaai.v32i1.11728.

[159] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, P. Richtarik, Scaling distributed machine learning with In-Network aggregation, in: 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21), USENIX Association, 2021, pp. 785–808.
URL https://www.usenix.org/conference/nsdi21/presentation/sapio

[160] Y. Zhao, J. Fan, L. Su, T. Song, S. Wang, C. Qiao, SNAP: A communication efficient distributed machine learning framework for edge computing, Proceedings - International Conference on Distributed Computing Systems 2020-November (2020) 584–594. doi:10.1109/ICDCS47774.2020.00072.

[161] J. Zerwas, K. Aykurt, S. Schmid, A. Blenk, Network Traffic Characteristics of Machine Learning Frameworks under the Microscope, Proceedings of the 2021 17th International Conference on Network and Service Management: Smart Management for Future Networks and Services, CNSM 2021 (2021) 207–215doi:10.23919/CNSM52442.2021.9615524.

[162] H. Yokoyama, T. Araki, Efficient distributed machine learning for large-scale models by reducing redundant communication, in: 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 2017, pp. 1–8. doi:10.1109/UIC-ATC.2017.8397638.

[163] S. S. Sandha, W. Cabrera, M. Al-Kateb, S. Nair, M. Srivastava, In-database distributed machine learning: Demonstration using teradata sql engine, Proc. VLDB Endow. 12 (12) (2019) 1854–1857. doi:10.14778/3352063.3352083.
URL https://doi.org/10.14778/3352063.3352083

[164] P. Sun, Y. Wen, T. N. B. Duong, S. Yan, Timed dataflow: Reducing communication overhead for distributed machine learning systems, Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS 0 (2016) 1110–1117. doi:10.1109/ICPADS.2016.0146.

[165] Y. Duan, N. Wang, J. Wu, Minimizing training time of distributed machine learning by reducing data communication, IEEE Transactions on Network Science and Engineering 8 (2) (2021) 1802–1814. doi:10.1109/TNSE.2021.3073897.

[166] Y. Bao, Y. Peng, C. Wu, Deep learning-based job placement in distributed machine learning clusters with heterogeneous workloads, IEEE/ACM Transactions on Networking (2022) 1–14doi:10.1109/TNET.2022.3202529.

[167] H. Lu, K. Wang, Distributed machine learning based mitigating straggler in big data environment, in: ICC 2021 - IEEE International Conference on Communications, 2021, pp. 1–6. doi:10.1109/ICC42927.2021.9500531.

[168] R. Zhou, N. Wang, Y. Huang, J. Pang, H. Chen, DPS: Dynamic Pricing and Scheduling for Distributed Machine Learning Jobs in Edge-Cloud Networks, IEEE Transactions on Mobile Computing (2022). doi:10.1109/TMC.2022.3195765.

[169] X. Zhou, J. Zhang, J. Wan, L. Zhou, Z. Wei, J. Zhang, Scheduling-efficient framework for neural network on heterogeneous distributed systems and mobile edge computing systems, IEEE Access 7 (2019) 171853–171863. doi:10.1109/ACCESS.2019.2954897.

[170] R. Zhou, J. Pang, Q. Zhang, C. Wu, L. Jiao, Y. Zhong, Z. Li, Online scheduling algorithm for heterogeneous distributed machine learning jobs, IEEE Transactions on Cloud Computing (2022) 1–1doi:10.1109/TCC.2022.3143153.

[171] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, S. Chawla, Themis: Fair and efficient GPU cluster scheduling, in: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), USENIX Association, Santa Clara, CA, 2020, pp. 289–304.
URL https://www.usenix.org/conference/nsdi20/presentation/mahajan