Enhancing Distributed Neural Network Training Through Node-Based Communications

Sergio Moreno-Álvarez, *Graduate Student Member, IEEE*, Mercedes E. Paoletti, *Senior Member, IEEE*, Gabriele Cavallaro, *Senior Member, IEEE*, and Juan M. Haut, *Senior Member, IEEE*

***Abstract*— The amount of data needed to effectively train modern deep neural architectures has grown significantly, leading to increased computational requirements. These intensive computations are tackled by the combination of last generation computing resources, such as accelerators, or classic processing units. Nevertheless, gradient communication remains as the major bottleneck, hindering the efficiency notwithstanding the improvements in runtimes obtained through data parallelism strategies. Data parallelism involves all processes in a global exchange of potentially high amount of data, which may impede the achievement of the desired speedup and the elimination of noticeable delays or bottlenecks. As a result, communication latency issues pose a significant challenge that profoundly impacts the performance on distributed platforms. This research presents node-based optimization steps to significantly reduce the gradient exchange between model replicas whilst ensuring model convergence. The proposal serves as a versatile communication scheme, suitable for integration into a wide range of general-purpose deep neural network (DNN) algorithms. The optimization takes into consideration the specific location of each replica within the platform. To demonstrate the effectiveness, different neural network approaches and datasets with disjoint properties are used. In addition, multiple types of applications are considered to demonstrate the robustness and versatility of our proposal. The experimental results show a global training time reduction whilst slightly improving accuracy. Code: https://github.com/mhaut/eDNNcomm.**

***Index Terms*— Data parallelism, deep learning, high-performance computing (HPC), neural networks, synchronous communications.**

Manuscript received 12 July 2022; revised 8 June 2023; accepted 18 August 2023. This work was supported in part by the Consejería de Economía, Ciencia y Agenda Digital of the Junta de Extremadura, in part by the European Regional Development Fund (ERDF) of the European Union under Grant GR21040, Grant GR21099 and Grant IB20040, in part by the Spanish Ministerio de Ciencia e Innovacion under Project PID2019-110315RB-I00 (APRISA), in part by the DEEP-EST Project (computing resources), and in part by the European Union's Horizon 2020 Research and Innovation Programme under Grant 754304. *(Corresponding author: Juan M. Haut.)*

Sergio Moreno-Álvarez is with the Departamento de Ingeniería de Sistemas Informáticos y Telemáticos, Escuela Politécnica, Universidad de Extremadura, 10003 Cáceres, Spain.

Mercedes E. Paoletti and Juan M. Haut are with the Departamento de Tecnología de Computadores y Comunicaciones, Escuela Politécnica, Universidad de Extremadura, 10003 Cáceres, Spain (e-mail: juanmariohaut@unex.es).

Gabriele Cavallaro is with the Jülich Supercomputing Centre, Forschungszentrum Jülich, 52428 Jülich, Germany.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2023.3309735.

Digital Object Identifier 10.1109/TNNLS.2023.3309735

I. Introduction

ADVANCES in machine learning (ML) algorithms have been partially fostered by the unparalleled growth in the amount of data available to train deep neural network (DNN) models. Indeed, with the adequate number of training samples to cover the parameter space, deep models provide interesting solutions for a wide range of applications and tasks, such as medical imaging analysis [1], remote sensing applications [2], or natural language processing [3], among others. As a common denominator, the corresponding data is prepared and presented as input to DNN-based models, which conduct a feature extraction procedure through a number of hierarchically stacked layers. These operational layers perform data transformations by combining the input features and the parameters comprised by each layer, which are organized as data-fit filters. Thus, after applying a transfer function, each layer provides the neural response to the existence of particular features. This process is refined as the data is processed by deeper layers, which extract increasingly complex and abstract features. In this context, the paramount importance of the input data is well known, which must be sufficient both to cover the parameter space and capture the variation over the input space.

Large data volumes provide opportunities for training increasingly complex models. Concurrently, advancements in both hardware devices and algorithms have substantially enhanced the performance of such models. However, the exponential growth of data and its high processing cost have boosted computational requirements. To tackle this challenge, fast computing systems and high-performance computing (HPC) platforms have been employed to facilitate collaborative training, where data is partitioned and distributed across multiple computing nodes, which communicate with each other to update the model parameters. The scalability offered by HPC platforms has significantly improved the training speed, but faces high communication volumes. Several domains benefited from the aforementioned points, such as natural language processing, computer vision or speech recognition. In the following, the opening remarks are introduced for both HPC and distributed training approaches.

A. Performance Barriers in Distributed Training

HPC provides a structured process for deploying and managing computing resources, such as graphics processing units (GPUs) or central processing units (CPUs). Processes

collaborate to solve the same problem, parallelizing workload partitions. In this regard, the most popular distributed technique is the *data parallelism scheme*, which allocates a replica of the model in every process. Each replica takes as input a portion of the original data, which has been previously divided into partitions of equal size. Moreover, the data is shuffled after each training epoch to ensure that all replicas see the input space, furthering the convergence of the model. Hence, each replica is trained with the complete dataset after a certain number of epochs. In contrast, *model parallelism* distributes network layers between different processes, thus, only a single network model is deployed and partitioned. Alternatively, *hybrid parallelism scheme* combines data and model parallelism. Usually, the communication and synchronization overhead of model and hybrid schemes is higher than that of data parallelism, making the latter technique more popular.

Despite the prevailing emphasis on computation phase optimization in numerous initiatives aimed at accelerating DNN, it is imperative to acknowledge that communication exerts a significant influence on the overall execution time. In this regard, replicas must repeatedly exchange information with each other, using the available communication channels. For instance, gradients are transmitted at the conclusion of each training iteration among replicas to facilitate the parameter updating step locally. Depending on the properties of the training algorithm, parameter servers (PSs) provide a centralized system, whilst collective operations offer a decentralized alternative [5]. Bottlenecks could appear depending on the amount of data to be communicated and the number of replicas competing for communication channels. Moreover, both the number of training iterations and the model parameters have a direct impact on the volume of data to be exchanged between replicas. Reducing these communications is an essential strategy to improve performance and minimize training time, but must ensure that model reliability is not jeopardized.

Regarding communication bottlenecks, channels such as InfiniBand, TCP, NVLink, or shared memory exhibit varying bandwidths, and hence, leveraging faster channels for data exchange can markedly diminish transfer time. These differences in communication speed are common in modern distributed platforms, where multiple communication channels are available. Moreover, different communication schemes are used depending on the type of synchronization. For instance, the *synchronous approach* introduces barriers, that is, synchronization points, at gradient exchanges to ensure the synchronization of all replicas. As a result, a delay emerges in the execution time caused by replica idle times. This shortcoming is aggravated in the case of resources with different computational capabilities, where the slowest replica will drag down the others by forcing them to wait for it, thus degrading the run-time. Contrary to the synchronous strategy, the *asynchronous approach* does not introduce synchronization points. This has a drawback, as asynchronous communications introduces staleness in the replica gradients. Each replica uses the available gradients to optimize the weights, without knowing whether they have been updated correctly or not. Thus, stragglers processes negatively affect the model reliability [6]. The management of stragglers processes is not common in homogeneous HPC platforms. However, they encounter communication delays, as the massive transmission of gradients leads to saturation of the communication channels. Furthermore, the communication between replicas inherently entails performance losses for both communication approaches.

B. Cutting-Edge DNNs: An Overview

The efficient training of DNNs holds great significance in order to harness the full potential of deep architectures. As the scale and sophistication of these models increase, the demand for communication and information processing between distributed processes intensifies. It is imperative to optimize the computational and communication facets of distributed learning [7]. It is noteworthy that the design of complex architectures does not universally guarantee performance gains. On the contrary, the performance may be degraded with deeper networks due to the well-documented challenge of vanishing gradients. This challenge is further compounded by the increased communication of gradients in distributed systems. Some promising DNNs deal with this.

1) Residual Networks (ResNets): ResNets [8] implement shortcut connections that propagate directly data representations with different levels of abstraction. These "direct data paths" improve the reuse of data during the forward step, whilst alleviating gradient vanishing during backward step. Nevertheless, these models comprise millions of parameters, requiring high computation and communications.

Furthermore, ResNet models are frequently employed for classification tasks, particularly in scenarios requiring fine-grained analysis. Such tasks are computational demanding due to the need to detect small interclass variations.

2) Vision Transformers (ViTs): Recently, ViT models have shown better performance than ResNets in image classification [9]. ViT utilizes self-attention layers, which are also known as *heads*, providing different and relevant features from the embedded input [10]. This facilitates the model to selectively attend to discriminative visual features, augmenting its capacity to comprehend and analyze intricate visual patterns. Indeed, the computational framework incorporates transformer blocks, which encompass multihead self-attention (MHSA), normalization layers, feed-forward networks (FFNs), and skip connections. The inherent computations entail substantial computational and communication overheads, posing significant resource demands.

3) Generative Adversarial Networks (GANs): Image generation models have garnered increasing attention within the scientific community, particularly, GANs [11], which are composed of two independent networks, that is, a generator ($\mathcal{G}$) and a discriminator ($\mathcal{D}$). The objective of the former is to learn the distributions of the input data, with the aim of generating new samples. Meanwhile, the later determines the data source, that is, whether the data is real or generated by $\mathcal{G}$. In this context, GAN represents a min–max optimization problem for $\Omega(\mathcal{D}, \mathcal{G})$, as described in the following equation:

$$\Omega(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{(x)}\big[\log \mathcal{D}(x)\big] + \mathbb{E}_{(z)}\big[\log(1 - \mathcal{D}(\mathcal{G}(z)))\big]. \quad (1)$$

GAN also suffers an intensive computation in the training phase due to: 1) two networks have to be trained, increasing

the computational load and 2) the increase of the data to be processed through new false data generated.

4) Recurrent Neural Networks (RNNs): RNNs [12] are computational demanding due to inherent complexity in weight initialization, optimization algorithm selection, and network instability. In this context, the significance of accelerating model convergence or reducing training times becomes paramount. This is also intensified considering the substantial number of iterations involved in the process. Thus, the computational and communication complexity in distributed systems is heightened. The functionality is briefly resumed in the following equation:

$$\mathcal{H}^t = f\left(\mathbf{W}_{(X,\mathcal{H})} \cdot \mathbf{X}^t + \mathbf{W}_{(\mathcal{H},\mathcal{H})} \cdot \mathcal{H}^{t-1} + \mathbf{b}_{\mathcal{H}}\right) \quad (2)$$

where $\mathbf{W}$ comprises the weights between input data $\mathbf{X}$ and hidden state vector $\mathcal{H}$ and $\mathbf{b}$ is the bias, at time step t. During forward propagation, the output from each time step needs to be communicated to the next time step, which leads to a large amount of data being transmitted between processes.

C. Exploring Strategies to Reduce Communication Costs

Reducing communication costs is a challenging task, requiring a combination of algorithmic, architectural, and hardware optimizations. The aforementioned complexity of DNNs and the performance challenges associated with distributed training pose significant obstacles in scaling the training process to large clusters. Considering the optimization achieved through data parallelization techniques regarding the computation aspect, the remaining challenge lies in addressing the communication aspect. Thus, it is imperative to develop efficient and scalable communication approaches to minimize data transmission whilst ensuring accuracy and convergence. One of the most widely adopted approach for efficient communication is the all-reduce scheme, which offers effective means of data exchange in distributed systems. Table I provides an overview of the main implementations of the all-reduce approach.

It must be noted that latency and bandwidth determines the communication costs in large-scale distributed systems. In this context, communication strategies should be designed to be flexible and adaptive, allowing them to work with a wide range of DNNs architectures, configurations, and applications. For instance, the ring all-reduce algorithm efficiently performs parallel reduction of data across multiple devices [13], whilst dense all-reduce [14] allows gradient accumulation across all replicas. However, it suffers from large overheads for the training of deep models with millions of parameters.

Reducing the number of parameters through gradient sparsification techniques is an elegant solution to solve this drawback. This have been implemented in all-reduce implementations based on top-k selection, where only the largest k components of the gradients are selected. Nonetheless, sparse all-reduce suffers from scalability problems. For instance, TopkA [15] takes advantage of an all-gather approach to gather sparse gradients, and then perform the sparse all-reduce locally. Meanwhile, gTopk [16] reduces the communication volume by using tree-like communications, selecting only the largest gradient components through tree levels. Lastly, gradient quantization is a common technique to reduce the number of bits that represent the gradient values, such as mixed [17] or lower $FP16$ precision values [18]. Furthermore, methods such as halving–doubling or doubles binary trees (DBTrees) have been studied. For instance, DBTrees performed better than ring all-reduce proposal in [23] for specific cases. Specifically, MultiTree [24] reduces the ring procedure by $2\log_t R$, where t represents the tree grade. This enhances the scalability and reduces the latency.

TABLE I

COMMUNICATION COSTS FOR SPARSE AND NON-SPARSE ALL-REDUCE STRATEGIES WITH R REPLICAS, k SPARSE GRADIENT COMPONENTS, LATENCY α, TRANSMISSION TIME β, AND OPERATION TIME γ

Method	Overhead	Bandwidth
Dense [19]	$2(logR)\alpha$	$2R(\frac{R-1}{R}\beta)$
TopkA [15]	$(logR)\alpha$	$2k(R-1)\beta$
Ok-Topk [20]	$(2R+2logR)\alpha$	$2k(\frac{R-1}{R})\beta$
gTopk [16]	$2(logR)\alpha$	$4k(logR)\beta$
Gaussian-k [21]	$2(logR)\alpha$	$2k(R-1)\beta$
Binary tree [22]	$2\alpha(logR)$	$(2\beta+\gamma)logR$
Ring [13]	$2(R-1)$	$\frac{R-1}{R}\gamma+\frac{2(R-1)}{R}\beta$
Recursive doubling [22]	$(logR)\alpha$	$(\beta+\gamma)logR$

Overwhelming communication issues are present in distributed platforms besides HPC, such as cloud computing (CC) and federated learning (FL). Nevertheless, these approaches pose distinct challenges in the realm of communication. In the domain of CC, data is distributed and processed among diverse nodes within a non-dedicated network, thereby leading to potential bottlenecks in communication between worker nodes and master node. Similarly, in FL, communication between the central server and participating devices can be a limiting factor, particularly, when the number of devices is large or when network connections are unreliable. FL allows the training of a model across multiple decentralized devices. Thus, as the number of replicas increases, the communication cost becomes a significant issue. Each device trains its own local model, where communication is required to exchange model updates between the devices, resulting in high communication costs. Additionally, the devices may have different hardware capabilities and network conditions, further complicating the communication process. These challenges pose significant obstacles to achieving efficient and scalable training. Despite this, FL has been found to be a useful approach for privacy-preserving applications.

D. Contributions

The huge computations and communications involved in the training of DNN-based processing methods have motivated this work. As previously stated, the computation has been addressed usually by distributing the workload using parallelism schemes. Nevertheless, communication problems resulting from data parallelization could lead to noticeable delays that are often not taken into account in the algorithm design. In this respect, the factors that most influence the communication cost are the frequency of data exchanges between replicas, synchronization and the amount of data transferred through the communication channels. Regarding the former, the number of exchanges increases the total communication

time proportionally, whilst synchronization forces the faster replicas to wait for the slower ones. Regarding the latter, elements such as input data and model parameters are crucial. In addition, the memory storage must have capacity for these large data requirements. Finally, traffic and concurrent transmissions over the communication channels could lead to network congestion. In fact, congestion is directly related to the scaling of the number of replicas. Thus, the higher the number of replicas, the higher the congestion on the channels.

Existing approaches aim to address the challenge of excessive communication by employing a combination of the aforementioned strategies. Nevertheless, these approaches often address highly specific scenarios and are challenging to accommodate in real-world settings. Consequently, there is a pressing need to develop alternative methods that can be easily applied across a broad range of situations, encompassing different deep models, diverse hardware configurations and network topologies. The advancements in communication management are instrumental in the successful implementation of distributed training frameworks, facilitating the seamless execution of intricate models within computing infrastructures. Against this background, communication-efficient scheme has to address the above issues in an optimal way for data-parallel distributed algorithms. Thus, the proposal focuses on optimizing and reducing communications between resources. In this context, the contributions of this work are the following.

1) A novel decentralized communication scheme is proposed to reduce the number of message transmissions through time-consuming communication channels.
2) A scalable implementation is provided to maximize the performance of computationally expensive algorithms by reducing unnecessary overhead and hence, optimizing the utilization of high-speed computational and communication hardware.
3) The applicability of the proposal over different depth models, such as classification or image generation models, is comprehensively explored. Obtained performance is ensured through a rigorous accuracy evaluation.
4) A versatile communication scheme that is complementary to additional optimization strategies, cluster configuration, and network topologies.

In summary, the proposed approach aims to reduce communication costs in distributed training by optimizing the communication pattern between nodes. Specifically, replicas of the same node are required to communicate more frequently with each other, which is referred to as intranode communication. In contrast, communication between replicas of different nodes, or internode communication, is minimized. This approach is designed to reduce the overall communication volume and minimize the impact of internode communication, which tends to be more expensive due to network overheads. Considering that each epoch is composed of several iterations, each model replica optimizes its weights using intranode gradients in each training iteration, whilst internode communication is performed in an ad hoc way for parameter sharing. This significantly minimizes the exchange of gradients as a function of the number of training iterations, preventing channel congestion. Nonetheless, the proposal entails a limitation. Beyond a certain point of scalability, the time gain is negligible, as the reduced number of communications between nodes does not produce sufficient speedup. This could be due to different reasons, such as a low amount of data or an excessive scaling. To overcome this limitation, the batch size should be adapted considering hardware configuration of the platform and data amounts. This implies that smaller batch sizes result in a higher frequency of communication exchanges between replicas compared to larger batch sizes.

The document is organized into several sections: a review of related work in Section II; a detailed description of the proposed decentralized communication scheme in Section III; an analysis of results from a comprehensive set of experiments in Section IV; and concluding remarks on the proposal's milestones and future work in Section V.

II. Related Work

Communication plays a key role in the total execution time of deep models that has not been addressed in the literature with the same seriousness as computation. Indeed, throughout the last few years, HPC methods have been striving to optimize both computation and communications jointly. For instance, Clarke et al. [25] implemented a tool to balance the computation of matrix multiplications between processes. In heterogeneous platforms, the communications have been modeled considering performance differences between communication channels [26]. Additionally, in [27] an automatic tool to manipulate communication cost expressions is implemented.

Classic scientific applications and distributed DNNs are frequently executed on HPC platforms, giving rise to shared requirements and challenges in these domains. For instance, Rico-Gallego et al. [28] proposes a data parallelism approach for HPC, conducting data partition based on the computational capacities of the devices to minimize communication costs in scientific applications. These techniques have been extrapolated to develop similar versions for distributed neural networks. In this context, computation awareness has been improved for data parallelism [4] and model parallelism [29]. As a consequence, partitioning the workload between resources from different nodes could produce communication bottlenecks that should be handled. Straightforwardly, the communication is a blocking or non-blocking gradient distribution, where the average of the data is calculated across all contributors and then, spread to the replicas before the optimization step. As a result, the training performance is affected by the communications for both blocking and non-blocking approaches.

In this regard, different communication approaches have been implemented to optimize the communication step in distributed DL frameworks. For instance, Coquelin et al. [30] proposed a distributed asynchronous communication based on a hierarchical gradient sharing scheme for multiple GPU nodes. Additionally, some works as [31] set the focus on batch optimizations for intermittent communications. This process entails the execution of stochastic gradient descent (SGD) on local replicas, followed by the subsequent aggregation of local information from all replicas through averaging.

This mode of communication has been found effective for replicas with disjoint data, that is, FL [32], and for heterogeneous data [33]. In [34], the decoupling of the all-reduce primitive (DeAR) into two consecutive operations is proposed to enable pipeline strategies. This allows for the overlap of both back-propagation and feed-forward computations, thereby avoiding additional communications. However, this does not account for potential channel contingencies in internode communications. Alternatively, Zhang et al. [35] proposed the weighted feedback-based pipelining (WFBP) mechanism, designed to optimize the performance by effectively overlapping communication overheads with computational tasks. Despite its near-optimal performance, WFBP has the drawback of not considering the feed-forward stage in the training process. BlueConnect [36] minimized communication overhead by decomposing scatter–gather operations under a hierarchical communication structure. This is particularly effective in fat-tree topologies, where each level of the hierarchy encompasses various communication strategies. Nevertheless, this poses synchronization challenges, thereby impacting performance under certain environments. Finally, Horovod [37] uses an optimal all-reduce ring algorithm for multiple GPUs. This conducts two communication phases. In the former, each replica sends and receives gradient chunks $R-1$ times. Next, replicas send the results of adding the received data to the corresponding stored data chunks $R-1$ times. Hence, a replica communicates with two of its peers $2\cdot(R-1)$ times. The obtained bandwidth is calculated as $\beta = 2(R-1)/R$.

Although decentralized approaches using collectives have reported a better performance for different topologies [38], centralized approaches are also used for multiple purposes. In this term, PS has been implemented for asynchronous and synchronous approaches. However, it suffers of stale parameters that negatively affect the convergence of the model and thus the final accuracy. Some works propose delayed PS updates and warm-up phases to overcome this problem [39]. As aforementioned, centralized approaches are common in CC, where the master node is responsible for the execution of the jobs and the order of computation among the workers. Meanwhile, workers are in charge of processing parts of the total computation. Recently, cloud algorithms have evolved and adapted techniques previously developed for HPC platforms. The popularity of these implementations has grown due to CC advantages, such as fault tolerance, reduced computation costs, and scalability. On the other hand, CC faces important challenges as security or replication management.

Finally, specific methodologies for managing communications in distributed DL algorithms have been shown to improve cross-platform training. For instance, Wangni et al. [40] reduced the communications complexity using a length reduction of sparse gradients combined with dropout techniques, whilst Ivkin et al. [41] exposed the bandwidth problem of distributed training and proposed a solution based on the communication of gradient sketches. Moreover, synchronous and asynchronous techniques were deeply studied by Tang et al. [42] for multiple system architectures, such as PS and all-reduce collectives, among with gradient quantization methods, compression techniques, or sparsification.

III. Proposed Node-Based Communication Algorithm

The proposed methodology is based on the principle of synchronous data parallelism. This section outlines the methodology employed for this process, elucidating the intricacies of forward propagation, gradient calculation, and parameter updating. Then, the innovative approach is introduced to minimize the use of costly communication channels.

A. Distributed Data-Parallel Training

Considering N training samples $\{\mathcal{X},\mathcal{Y}\} = \{\mathbf{X}_n,\mathbf{Y}_n\}_{n=1}^{N}$ drawn from an unknown distribution, and a DNN comprising a set of parameters $\boldsymbol{\theta} = \{\mathbf{W}_l,\mathbf{b}_l\}_{l=1}^{L} \in \Omega$, that is, weights and biases that are hierarchically organized across L operational layers, the DNN defines a non-linear transformation $f_{\boldsymbol{\theta}}: \mathcal{X}\rightarrow\mathcal{Y}$ where $\boldsymbol{\theta}$ is adjusted to approximate its response predictions $f(\mathbf{X}_n,\boldsymbol{\theta}) \in \mathcal{Y}$ to the desired output instances $\mathbf{Y}_n \in \mathcal{Y}$ given inputs $\mathbf{X}_n \in \mathcal{X}$. This follows the empirical risk minimization principle based on a differentiable loss function $\ell(\mathbf{Y}, f(\mathbf{X},\boldsymbol{\theta}))$, which measures the difference between the prediction outputs and the actual problem instances. Then, the goal is to find the optimal $\boldsymbol{\theta}^*$ that minimizes the loss, that is, $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}}\ \ell\ (\boldsymbol{\theta})$, knowing that $\boldsymbol{\theta}^*$ is a stationary point over the loss landscape, thus $\nabla\ell(\boldsymbol{\theta}^*)=0$.

In this context, the DNN training procedure involves a combination of T finite and predefined forward–backward propagation steps to update $\boldsymbol{\theta}$ toward $\boldsymbol{\theta}^*$, down the loss surface along the direction of the gradient, where parameters update follows the delta rule for each time step t:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \mu_t \nabla\ell\left(\boldsymbol{\theta}^{t}\right). \tag{3}$$

Consequently, feature extraction is first performed during the forward propagation, that is, for the nth training sample, L intermediate data representations are hierarchically calculated and propagated across the layers, where the lth layer takes the features filtered by the previous layer, refines them, and passes them to the next layer following (4), where $\mathbf{X}_n^{(l-1)}$ and $\mathbf{X}_n^{(l)}$ represent the inputs and neural activation responses of the lth layer, regarding the nth training sample, and $\boldsymbol{\theta}_l$ comprises its adjustable parameters. The activation function is denoted as σ

$$\mathbf{X}_n^{(l)} = \sigma\left(\mathbf{X}_n^{(l-1)},\ \boldsymbol{\theta}_l\right) = \sigma\left(\mathbf{X}_n^{(l-1)} \star \mathbf{W}_l + \mathbf{b}_l\right). \tag{4}$$

Equation (4) is applied hierarchically through the L layers, obtaining a L-layers composition over the input data $\mathbf{X}_n$ until the final abstract representation $\mathbf{X}_n^{(L)}$ is obtained. This is finally processed to obtain the final prediction. Then, parameters update is conducted during the backward stage by means of (3) and following the chain rule. At this point, it is noteworthy that the unbiased gradient estimator $\mathbf{g}$ is obtained instead of the expected $\nabla\ell(\theta)$ over training samples with $\mathbb{E}[\mathbf{g}] \simeq \nabla\ell(\boldsymbol{\theta})$. In particular, mini-batch approaches consider K batches of size $B = N/K$, that is, the ith batch comprises a subset of the training samples $\mathcal{B}_i \subset \{\mathcal{X},\mathcal{Y}\} = \{\mathbf{X}_n,\mathbf{Y}_n\}_{n=1+B(i-1)}^{n+B-1}$, thus $\sum_{i=1}^{K}|\mathcal{B}_i| = KB = N$. To cover all the data, time steps are defined in terms of a number of epochs E, and a number of iterations per epoch K, that is, $t = i + K(e-1)$, $\forall e \in [1,E]$

and $\forall i \in [1, K]$, thus $T = EK$. Notations $\boldsymbol{\theta}^t \equiv \boldsymbol{\theta}^{e,i}$ are equivalent. Consequently, the batch-based estimator of the gradient is used per time step

$$\mathbf{g}^t \equiv \mathbf{g}^t_{\mathcal{B}_i} = \frac{1}{B} \sum_{n=1+B(i-1)}^{n+B-1} \nabla \ell\big(\mathbf{Y}_n, f\big(\mathbf{X}_n, \boldsymbol{\theta}^t\big)\big). \tag{5}$$

Assuming a distributed environment with data-parallel scheme and comprising R replicas, training samples are organized on partitions of N/R samples and then assigned to the replicas, as well as a copy of the untrained model parameters $\boldsymbol{\theta}_r$. Within the rth replica, the data is organized into batches of size B, which are sequentially processed by the corresponding model iteration by iteration. After forward pass, the loss function is applied to obtain the prediction error over the data batch, and the corresponding gradients are computed $\mathbf{g}^t_r$ at step t. Hence, gradients are calculated in each training iteration for all replicas. To ensure that all replicas perform the same iterations, the batch size B must be the same for all replicas. Since gradients are back-propagated along the DNN architecture to fine-tune $\boldsymbol{\theta}$, a reduction operation is performed on the local gradients, calculating their mean to obtain the global gradients $\mathbf{G}^t$

$$\begin{aligned} \mathbf{G}^t &= \frac{1}{R} \sum_{r=1}^{R} \mathbf{g}^t_r \\ &= \frac{1}{R} \sum_{r=1}^{R} \left(\frac{1}{B} \sum_{n=1+B(i-1)}^{n+B-1} \nabla \ell\big(\mathbf{Y}_n, f\big(\mathbf{X}_n, \boldsymbol{\theta}^t\big)\big) \right). \end{aligned} \tag{6}$$

Regarding the exchange of information, this implies an internode communication frequency of $(1 + EK)$, where the first part identifies the initial communication of parameters and the second, exchange of gradients per iteration. Once $\mathbf{G}^t$ is obtained, parameters are updated accordingly to (3) to minimize the loss. The parameter update or the tth step of (7) is reformulated in terms of $\mathbf{G}^t$ and the current parameters $\boldsymbol{\theta}^t_r$ of each replica, where μ_t is the learning rate

$$\boldsymbol{\theta}^{t+1}_r = \boldsymbol{\theta}^t_r - \mu_t \mathbf{G}^t. \tag{7}$$

In this way, distributed training is performed to take advantage of the multiple nodes and their resources available in the distributed environment. Nonetheless, the shortcomings of this approach are analyzed in the following discussion.

B. Empirical Findings and Limitations in Distributed Training

The design of a proper communication algorithm could improve the performance for large communications, reducing the network contention. The communication overhead can be mitigated by reducing the number of data exchanges, that is, *communication rounds*. The number of rounds depends on several factors, including the batch size, communication periods, number of nodes, and communication strategy [43]. In the following, the impact of multiple factors on communication time is comprehensively analyzed to elucidate their impact.

Gradients and parameters are determined solely by the network architecture, implying that the message size for each iteration remains constant, that is, $|\boldsymbol{\theta}|$, regardless of the batch size B. Nonetheless, larger batch sizes reduce the number of communication rounds, where communications are commonly performed at the end of each iteration. However, the use of larger batch sizes may increase the risk of converging to local minima during training [44]. Furthermore, this practice may result in significant communication overhead, specially when dealing with a large number of training iterations, for example, in mini-batch approaches. One promising approach is one-shot averaging [45], which conducts communication only at the end of the entire training process. Thus, communication frequency can be adjusted to minimize its overhead.

Observed trends lead to a common conclusion: the utilization of traditional communication strategies in distributed training with an increasing number of replicas results in linearly escalating communication costs. Consequently, bottlenecks arise during specific communication instances. One potential avenue is to overlap communication with computation, which entails optimizing communication operations and reducing the overall communication time. However, the implementation of such scheduling approaches is intricate and may compromise model accuracy. Alternative solutions such as gradient compression impact the model performance in terms of accuracy due to the inherent trade-off between accuracy and traffic reduction. Similarly, these considerations are applicable to quantization and sparsification methods.

C. Overview of the Proposed Methodology

The proposed approach leverages the computational resources available in an HPC platform (GPUs and CPUs), to address costly computations and execution management, respectively, considering data parallelism to distribute the workload among all the computational resources, that is, to provide a grade of horizontal scalabity. Nevertheless, the workload distribution entails additional communication times when sharing information between replicas. In this regard, two types of communication, that is, *intranode* communication and *internode* communication are used depending on whether there is only communication between replicas located on the same node or between replicas hosted on different nodes of the distributed environment, respectively. Internode communication channels are slower than intranode paths, such as shared memory or intraGPU communication channels such as NVLink.

To reduce communication costs, the proposed method avoids gradient exchanges through time-consuming communication channels by promoting intranode communications instead of internode communications. In this regard, let M be the number of nodes, where the mth node contains R_m replicas, thus $R = \sum_m^M R_m$. Furthermore, assuming a batch size of B samples, let us define each epoch $e \in [1, E]$ by its number of training iterations $K = N/(R \cdot B)$.

At the beginning of every epoch e, the N samples $\{\mathcal{X}, \mathcal{Y}\} = \{\mathbf{X}_n, \mathbf{Y}_n\}_{n=1}^N$ are shuffled and randomly grouped into $R = M \cdot R_m$ data partitions, where partition $\mathcal{P}_r \subset \{\mathcal{X}, \mathcal{Y}\} = \{\mathcal{X}_r, \mathcal{Y}_r\} = \{\mathbf{X}_{r,n}, \mathbf{Y}_{r,n}\}_{n=1}^{N/R}$ is sent to the rth replica hosted by the mth node. This ensures the uniform distribution of the data between

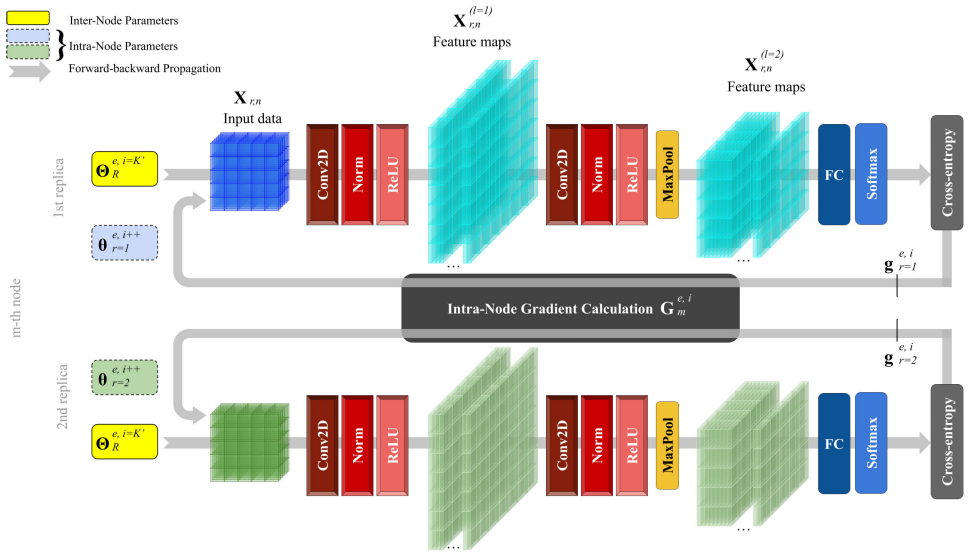


Fig. 1. Graphical overview of the proposal considering two replicas within the mth node, that is, $R_m = 2$ with $r = 1$ and $r = 2$. The nth input sample of each replica $\mathbf{X}_{r,n}$ is processed by a standard CNN. During the ith iteration in epoch e, fast intranode communication is conducted to collect the local gradients $\mathbf{g}_r^{e,i}$ and compute the corresponding intranode gradients $\mathbf{G}_m^{e,i}$, which is used to update parameters for the next iteration $\boldsymbol{\theta}_r^{e,i++}$. Furthermore, every K' iterations, replicas are updated with global parameters $\boldsymbol{\Theta}_R^{e,i=K'}$ by internode parameter communication to cover all training data.

the R replicas, which contributes to both the stability of the consensus among the different replicas and the convergence of the model. Then, the rth replica sequentially splits its data partition into K batches, where the ith batch $\mathcal{B}_i \subset \{\mathcal{X}_r, \mathcal{Y}_r\} = \{\mathbf{X}_{r,n}, \mathbf{Y}_{r,n}\}_{n=1+B(i-1)}^{n+B-1}$ is processed at the ith iteration of all epochs and globally shuffling the content. This ensures that all data is processed at the end of an epoch, avoiding oscillations in the optimization process.

Focusing on the ith iteration at epoch e, the R_m replicas of node m are each trained on their own batch $\mathcal{B}_i$, producing their corresponding local gradients $\mathbf{g}_r^{e,i}$. These are collected to calculate the intranode aggregated gradients $\mathbf{G}_m^{e,i}$

$$\mathbf{G}_m^{e,i} = \frac{1}{R_m} \sum_{r=1}^{R_m} \mathbf{g}_r^{e,i}. \tag{8}$$

In contrast to other strategies, replicas of the same node are trained in isolation for about $K' \leq K$ iterations, communicating only the gradients obtained between them to update their parameters according to $\mathbf{G}_m^{e,i}$, using fast intranode communication channels, and thus reducing the communication load in comparison with the baseline, (6), which conducts internode all-reduce communications at each iteration, sending large information across the network as the data grows. Consequently, during K' iterations, intranode replicas exclusively handle information from its node m, disregarding information from outside nodes. To obtain a global coverage of the training data, global internode communication is performed at both the K'th and the final iterations of each epoch,[1] when the proposed scheme aggregates all local parameters from replicas to obtain the global parameters

$$\boldsymbol{\Theta}_R^{e,i=K'} = \frac{1}{R} \sum_{m=1}^{M} \sum_{r=1}^{R_m} \boldsymbol{\theta}_r^{e,i=K'} \tag{9}$$

where $\boldsymbol{\theta}_r^{e,i=K'}$ are the rth replica parameters after K' iterations of an epoch, and $\boldsymbol{\Theta}_R^{e,i=K'}$ are the global parameters, shared with all replicas. This ensures that, at the beginning of each epoch and after K' iterations, all replicas have the same information ensuring the global consensus. This means an internode communication frequency of $E(1 + \lfloor K/K' \rfloor)$. The procedure is depicted by Fig. 1.

1) Convergence Proof: Provided that N training samples are uniformly distributed among the R replicas, the goal is to find the minimizer $\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta} \in \Omega} \ell(\boldsymbol{\Theta}_R)$, where ℓ is the average of the local objectives computed on the corresponding data

$$\begin{aligned} \ell(\boldsymbol{\Theta}_R) &= \frac{1}{R} \sum_{m=1}^{M} \sum_{r=1}^{R_m} \left(\frac{1}{B} \sum_{n=1+B(i-1)}^{n+B-1} \ell_n(\boldsymbol{\Theta}_R) \right) \\ &= \frac{1}{R} \sum_{m=1}^{M} \sum_{r=1}^{R_m} \left(\frac{1}{B} \sum_{n=1+B(i-1)}^{n+B-1} \ell(\mathbf{Y}_n, f(\mathbf{X}_n, \boldsymbol{\Theta}_R)) \right). \end{aligned}$$

In this context, the proposed algorithm is indeed a distributed mini-batch approach with two synchronization schemes, where the former is conducted during all the $E \cdot K$ steps regarding the exchange of gradients $\mathbf{g}_r$ between replicas hosted in the same node (intranode synchronization), and the latter is conducted during $E(1 + \lfloor K/K' \rfloor)$ steps regarding the exchange of parameters $\boldsymbol{\Theta}_R$ between all the replicas (internode synchronization).

Focusing on the intranode synchronization, and assuming $K' = K$, it must be noted that replicas in a worker node

[1] Knowing that $K' \leq K$, the method is forced to perform the global exchange of parameters at least once (when $K' = K$) with a maximum of $(1 + \lfloor K/K' \rfloor)$ times when $K' < K$, completing the consensus among all replicas in the distributed environment.

m conduct in an isolated way a distributed mini-batch optimization during K iterations per epoch. Performing over a finite set of $(N/R)\cdot R_m$ sampled independently and identically distributed (i.i.d.) data and starting from parameters $\boldsymbol{\theta}_m = \boldsymbol{\Theta}_R$, the convergence of the mth node depends on the convergence of its replicas. It could be assumed that each $\ell_n(\boldsymbol{\theta}_r)$ has a Lipschitz continuous gradient with constant $C > 0$ with respect to the distance between any two points $\boldsymbol{\theta}_r, \tilde{\boldsymbol{\theta}}_r \in \Omega$, such that the norm of the gradient at them is bounded $\|\nabla\ell(\boldsymbol{\theta}_r) - \nabla\ell(\tilde{\boldsymbol{\theta}}_r)\|_2 \leq C\|\boldsymbol{\theta}_r - \tilde{\boldsymbol{\theta}}_r\|_2$. This ensures that the objective function has a minimum and the gradients are not too sensitive. Assuming the complete isolation, that is, without the aggregation of the other replicas, and due to the distribute mini-batch inner convergence, after i iterations, the rth replica converges almost surely to a stationary point of $\ell(\boldsymbol{\theta}_r)$ such that $\boldsymbol{\theta}_r^* = \arg\min_{\boldsymbol{\theta}_r} \ell(\boldsymbol{\theta}_r)$, and thus $\nabla\ell(\boldsymbol{\theta}_r^*) = 0$. In this context, the consensus mechanism of (8) ensures that a wider range of the data space is covered, enabling the rth replica to converge faster and with fewer oscillations. As a consequence, the procedure converges within the node m.

Regarding the internode synchronization, it is conducted at least one time per epoch, averaging the parameters by means of (9). This implies that, after a maximum of K iterations per epoch, the replicas can provide divergent parameters $\boldsymbol{\theta}_r^{e,i=K}$ depending on the data on which they have been executed. This may introduce oscillations that hinder the convergence of the model. To avoid this, there are two factors that reduce divergence in the global consensus. On the one hand, the hyperparameter K' regulates the number of synchronization points, acting as a trade-off mechanism between the consensus and the number of communications, that is, the smaller the number, the more the replicas are globally synchronized, so that the oscillations between the parameters of one iteration and another is reduced, but the frequency of internode communication increases. On the other hand, R_m acts as a regulator. In fact, the consensus mechanism of (8) reduces variations, so that nodes with a high R_m tend to provide more stable results. As a consequence, the control of K' and R_m positively influences the model convergence.

D. Implementation Essentials

The proposed methodology has been integrated over the ColossalAI framework [38], where the baseline is already implemented. ColossalAI is an open-source PyTorch-based system that provides different paradigms of parallelization, such as data parallelism. It is relatable to its ease of use and scalable training for distributed platforms.

Two different MPI groups are created for the proposed method. Firstly, the intranode group g_group will conduct communications in each training iteration among local replicas. Secondly, the internode group G_group will provide the average parameters among global replicas.

The baseline method only requires the internode group. The implementation for an all-reduce communication, shown in Fig. 2, is described in Algorithm 1 along with a brief description of the basic steps to perform DL training. These basic steps include: 1) obtaining the batch data from the dataset; 2) passing the data through the neural network model; 3) calculating the error; and 4) calculating and optimizing the parameters. On the other hand, the proposed methodology shown in Fig. 3 is described by Algorithm 2. Note that the gradients all-reduce step is performed for intranode g_group replicas. Meanwhile, the parameters all-reduce step is conducted for global internode parameter communications G_group between all the replicas.

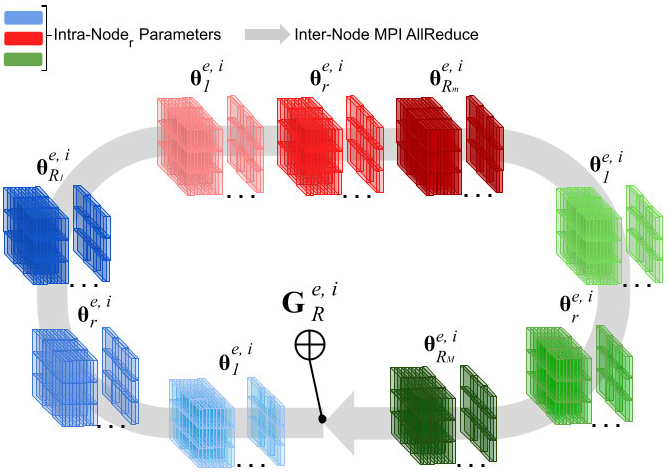


Fig. 2. Example of a communication scheme using all-reduce. Each color represents a replica, where different color shades identify replicas within the same node $r \in R_m$, with $m \in [1, M]$ a specific node. All local gradients are collected at every iteration to compute the global gradient $\mathbf{G}^{e,i}$.

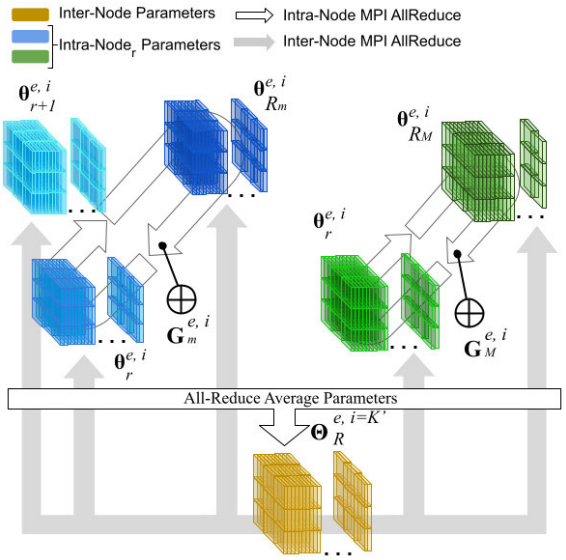


Fig. 3. Proposed scheme based on all-reduce collectives. Each color identifies a replica, where different color shades represent replicas within the same node $r \in R_m$, with $m \in [1, M]$ a specific node. Fast intranode gradient calculation is conducted at every iteration to update replica parameters hosted on the same node. Internode parameter communication is performed when conducted K' number of iterations.

IV. Experimentation Results

To provide significant and representative evidences of the efficiency and effectiveness of the proposed scheme, exhaustive experimentation has been conducted for large models, analyzing different datasets and discussing a variety of applications. The experimentation focuses on evaluating the

Algorithm 1 Training Procedure of Distributed Training. The Algorithm Comprises Forward Propagation, Loss and Gradient Calculation, and Parameter Update

for $\forall e \in [1, E]$ **do**
  for $\forall i \in [1, K]$ **do**
    $t = i + K(e-1)$
    ▷ *Batch processing:*
    $\mathcal{B}_i \subset \{\mathcal{X}_r, \mathcal{Y}_r\} = \{\mathbf{X}_{r,n}, \mathbf{Y}_{r,n}\}_{n=1+B(i-1)}^{n+B-1}$
    $\mathbf{X}_{r,n}^{(l)} = \sigma(\mathbf{X}_{r,n}^{(l-1)},\ \boldsymbol{\theta}_l^t), \forall l \in L$
    ▷ *Gradients calculation:*
    $\mathbf{g}_r^t = \frac{1}{B}\sum_n \nabla\ell(\mathbf{Y}_{r,n}, f(\mathbf{X}_{r,n}, \boldsymbol{\theta}_r^t))$
    $\mathbf{G}^t = \frac{1}{R}\sum_{r=1}^{R} \mathbf{g}_r^t$ ▷ *G_group*
    ▷ *Parameters update:*
    $\boldsymbol{\theta}_r^{t+1} = \boldsymbol{\theta}_r^t - \mu_t \mathbf{G}^t$
  end for
end for

Algorithm 2 Training Procedure for the Proposed Methodology. The Algorithm Comprises Forward Propagation, Loss and Gradient Calculation, and Parameter Update

for $\forall e \in [1, E]$ **do**
  for $\forall i \in [1, K]$ **do**
    $t = i + K(e-1)$
    ▷ *Batch processing:*
    $\mathcal{B}_i \subset \{\mathcal{X}_r, \mathcal{Y}_r\} = \{\mathbf{X}_{r,n}, \mathbf{Y}_{r,n}\}_{n=1+B(i-1)}^{n+B-1}$
    $\mathbf{X}_{r,n}^{(l)} = \sigma(\mathbf{X}_{r,n}^{(l-1)},\ \boldsymbol{\theta}_l^t), \forall l \in L$
    ▷ *Gradients calculation:*
    $\mathbf{g}_r^t = \frac{1}{B}\sum_n \nabla\ell(\mathbf{Y}_{r,n}, f(\mathbf{X}_{r,n}, \boldsymbol{\theta}_r^t))$
    $\mathbf{G}_m^t = \frac{1}{R_m}\sum_{r=1}^{R_m} \mathbf{g}_r^t$ ▷ *g_group*
    ▷ *Inter-node consensus:*
    if $i == K'$ **then**
      $\boldsymbol{\Theta}_R^t = \frac{1}{R}\sum_{m=1}^{M}\sum_{r=1}^{R_m} \boldsymbol{\theta}_r^t$ ▷ *G_group*
      $\boldsymbol{\theta}_r^t = \boldsymbol{\Theta}_R^t$
    end if
    ▷ *Parameters update:*
    $\boldsymbol{\theta}_r^{t+1} = \boldsymbol{\theta}_r^t - \mu_t \mathbf{G}^t$
  end for
end for

well-known communication latency problem for data parallelism schemes. The experimental evaluation of the proposal encompasses a wide range of studies, including speedup analysis, classification tasks on deep models, ViTs, evaluation of optimizers, fine-grained classification, and image generation. Standard all-reduce implementation is considered as baseline.

A. Experimental Datasets

Next, the description of the studied datasets in the presented study is provided.

1) The **MNIST** [46] dataset is composed of a training set with 60 000 samples and a test set with 10 000 samples. Images represent a digit from 0 to 9 with size of 30 × 30. Image colors are black and white.
2) The **CIFAR-10** and **CIFAR-100** [47] datasets are composed of 60 000 colored images of size 32 × 32 with 6000 and 600 images per class, respectively. Both datasets are divided in training and test data, using 50 000 for training and 10 000 for evaluation. The main difference between both datasets is that CIFAR-10 has 10 independent classes assigned for classification. Conversely, CIFAR-100 has 100 sub-classes grouped into 20 classes.
3) Four fine-grained datasets are considered. Firstly, **Stanford Cars** [48] is composed of 16 185 colored images of size 360 × 240 from the rear of 196 types of cars and divided in half between training and test. Secondly, **Stanford Dogs** [49], contains 20 580 images of 120 dogs classes, which are divided into 12 000 and 8580 for training and testing, respectively. Thirdly, **CUB-200-2011** [50] contains 11 788 birds images of 200 classes divided into 5994 for training and 5794 for testing. Each image is composed of 15 part locations, 312 binary attributes, and 1 bounding box. Finally, **FGVC Aircraft** [51] is composed of 10 200 aircraft images of 102 classes. Each image has a tight bounding box and a hierarchical model label.
4) The **ImageNet-1K** [52] dataset comprises 1000 mutually exclusive classes and contains around 1 281 167 training images, 50 000 validation images and 100 000 test images. Each image is of size 256 × 256 × 3 resized to 224 × 224 × 3. Images contain animals, landscape, or fungible objects, among others.

B. Experimental Settings

Experiments have evaluated the behavior of both baseline and proposed schemes. The training runs averages are calculated from five Monte Carlo executions for all employed metrics, that is, training time, accuracy, and loss. Two platforms are used in order to test the proposed implementation in different architectures. The first platform is the *Modular Supercomputer Architecture* (MSA) developed by the European project Dynamical Exascale Entry Platform-Extreme Scale Technologies (DEEP-EST) [53]. This platform is composed of three main modules: cluster module (CM), data analytics module (DAM), and extreme scale booster (ESB). These modules are constituted of 50 CPUs nodes, 16 GPU+CPU nodes, and 74 GPU+CPU nodes, respectively. In the performed experiments, the ESB module is used for a maximum of 20 nodes due to its suitable design for the training of deep learning algorithms. Each node is composed of an NVIDIA V100 Tesla GPU and Intel Xeon "Cascade Lake" Silver 4215 CPU running at 2.50 GHz. Network between nodes is a 100 Gb/s Infiniband. The second platform is the Ceta-Ciemat supercomputer, where four nodes, named Tesla accelerators (TAs), have been selected for the experimentation with four NVIDIA V100 Tesla GPU per node and Intel CPU "Cascade Lake" 6240 running at 2.50 GHz. Both modules are shown in Fig. 4.

TABLE II

EXPERIMENTAL MODELS DESCRIPTION. IN THE MODELS, MP AND AVG DENOTE A MAXPOOLING OR AVERAGEPOOLING LAYER, RESPECTIVELY. THE PARAMETERS FOR MP ARE KERNEL SIZE AND STRIDE. PADDING AND STRIDE ARE DEFINED AS 2 AND 2 FOR THE IMAGENET DATASET, AND 1 AND 1 FOR CIFAR. *C* REFERS TO CIFAR AND *I* TO IMAGENET. IN VIT MODELS, *B* AND *S* REFERS TO BASE AND SMALL MODELS, RESPECTIVELY. *T* IS THE ACRONYM USED FOR TRANSFER, LR REFERS TO LEAKYRELU AND UP IS UPSAMPLE WITH A SCALE FACTOR OF 2. IN GAN MODELS, IMAGE SIZE IS HEIGHT × WIDTH × DEPTH, VALIDITY REFERS TO FALSE OR TRUE (0, 1) AND LABEL (0, 9) THE CLASS NUMBER. FINALLY, BN DENOTES BATCHNORM LAYER AND DP REFERS DROPOUT

Model	**Conv1**	**Conv2x**	**Conv3x**	**Conv4x**	**Conv5x**	**Tail**
ResNet18	C: $3\times3\times64$ I: $7\times7\times64$ + MP(3,2)	$2\times\begin{bmatrix}3\times3\times64\\3\times3\times64\end{bmatrix}$	$2\times\begin{bmatrix}3\times3\times128\\3\times3\times128\end{bmatrix}$	$2\times\begin{bmatrix}3\times3\times256\\3\times3\times256\end{bmatrix}$	$2\times\begin{bmatrix}3\times3\times512\\3\times3\times512\end{bmatrix}$	AVG + FC(512)
ResNet34	C: $3\times3\times64$ I: $7\times7\times64$ + MP(3,2)	$3\times\begin{bmatrix}3\times3\times64\\3\times3\times64\end{bmatrix}$	$4\times\begin{bmatrix}3\times3\times128\\3\times3\times128\end{bmatrix}$	$6\times\begin{bmatrix}3\times3\times256\\3\times3\times256\end{bmatrix}$	$3\times\begin{bmatrix}3\times3\times512\\3\times3\times512\end{bmatrix}$	AVG + FC(512)
ResNet50	C: $3\times3\times64$ I: $7\times7\times64$ + MP(3,2)	$3\times\begin{bmatrix}1\times1\times64\\3\times3\times64\\1\times1\times256\end{bmatrix}$	$4\times\begin{bmatrix}1\times1\times128\\3\times3\times128\\1\times1\times512\end{bmatrix}$	$6\times\begin{bmatrix}1\times1\times256\\3\times3\times256\\1\times1\times1024\end{bmatrix}$	$3\times\begin{bmatrix}1\times1\times512\\3\times3\times512\\1\times1\times2048\end{bmatrix}$	AVG + FC(2048)
ResNet101	C: $3\times3\times64$ I: $7\times7\times64$ + MP(3,2)	$3\times\begin{bmatrix}1\times1\times64\\3\times3\times64\\1\times1\times256\end{bmatrix}$	$4\times\begin{bmatrix}1\times1\times128\\3\times3\times128\\1\times1\times512\end{bmatrix}$	$23\times\begin{bmatrix}1\times1\times256\\3\times3\times256\\1\times1\times1024\end{bmatrix}$	$3\times\begin{bmatrix}1\times1\times512\\3\times3\times512\\1\times1\times2048\end{bmatrix}$	AVG + FC(2048)

Shufflenet_V2_1x	**Conv1+MP**	**Stage2**	**Stage3**	**Stage4**	**Conv5**	**Tail**
	$3\times3\times24$ + MP(3,2)	$4\times[3\times3\times116]$	$8\times[3\times3\times232]$	$4\times[3\times3\times464]$	$1\times1\times1024$	AVG + FC(1000)

Transformers	**Width ‖ Depth**	**Heads**	**Path Size**	**Parameters** ($\times10^3$)	**Epochs**	**Pretrained**
ViT-B (Timm-T)	384 ‖ 12	12	16	8686	200	True
ViT-S (Timm-T)	224 ‖ 6	8	8	2166	50	True

GAN	**Latent**	**FC**	**Conv1**	**Conv2**	**Conv3**	**Output**
Generator	128	$(size//4)^2$ + BN	UP + $3\times3\times128$	UP + $3\times3\times64$	$3\times3\times32$	Image
Discriminator		$3\times3\times16$	$3\times3\times32$	$3\times3\times64$	$3\times3\times128$	Validity, Label

Recurrent	**Unit**	**Classifier**
	(Simple / LSTM / GRU) + BN + DP(0.25)	FC(32) + DP(0.5) + FC(10)

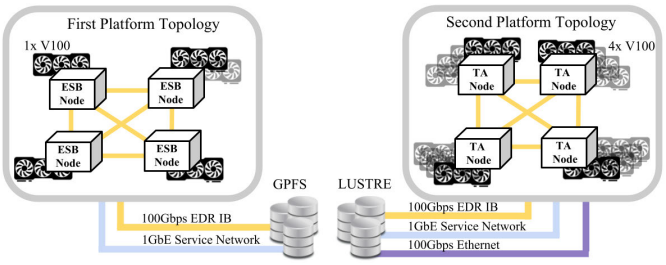


Fig. 4. Cluster configurations used in the experimentation. The data is provided to the modules using high-performance clustered file system software (GPFS) for ESB nodes or Lustre TA nodes.

Regarding the neural networks, evaluated models vary in depth (**ResNet18**, **ResNet50** and **ResNet101**) to provide different levels of feature extraction. These models are trained using a scheduler of the learning rate 0.1 with SGD momentum of 0.9, weight decay $5e^{-4}$ and batch size of 128. The implemented scheduler is *CosineAnnealingLR* with a maximum number of iterations K_max equal to the number of epochs $E = 400$. On the other hand, transformer models demonstrated to perform very well on image classification using sequences of image patches [54]. Indeed, in the experimentation, pretrained ViT models [55] (**ViT-B** and **ViT-S**) are evaluated using Adam optimizer [56] with those hyperparameters provided by Zhu et al. [54] for 200 and 50 epochs, respectively. In addition, for fine-grained classification, 100 epochs have been conducted using the *MultiStepLR* scheduler with milestones $50, 80$ and batch size 48. Milestones indicate the epoch number where the learning rate is decayed. For fine-grained datasets, the training is performed using the **ResNet50** model with SGD, momentum of 0.9, learning rate of $1e^{-2}$ and γ 0.1. Next, **GAN** is trained with a batch size of 32 for the MNIST dataset with Adam during 300 epochs, using a value of learning rate of $2e^{-4}$ with betas $B1$ and $B2$ decay of 0.5 and 0.999 for both optimizers, generator, and discriminator, respectively. Batch normalization is used for all training models. Models are described in Table II. The scheduler used for GAN is *CosineAnnealingLR* with a total of steps equal to the number of epochs. Finally, ImageNet is trained with learning rate $2e^{-3}$ for the proposed method and $1e^{-3}$ for the baseline. This requirement is set to avoid local minima in complex training scenarios, where the proposed method requires bigger learning steps to maintain the convergence. This is also implemented to train MNIST with RNNs. For these last two experiments, the batch size is set to 64 per replica and $K' = 50$. Rest of the hyperparameters are extracted from AdaBelief [57] for ImageNet. On the contrary, RNNs are trained with Adam.

C. Experimental Discussion

1) First Experiment: This experiment evaluates different node distributions in order to examine the speedup over the ResNet models. Also, an accuracy comparison between the baseline and proposed methodologies is performed using the CIFAR-10 and CIFAR-100 datasets. Obtained results are presented in Table III. In the obtained results, it can be observed how different factors can affect the speedup, such as the number of iterations performed by each replica, the number of parameters of the network and the channel congestion. Moreover, the speedup will increase until the point that intranode communications are significantly reduced. Also, the acceleration gain is slightly increased depending on the number of parameters. MA denotes the proposal accuracy evolution until the baseline maximum accuracy is reached. Meanwhile, MA-A denotes the acceleration produced by the

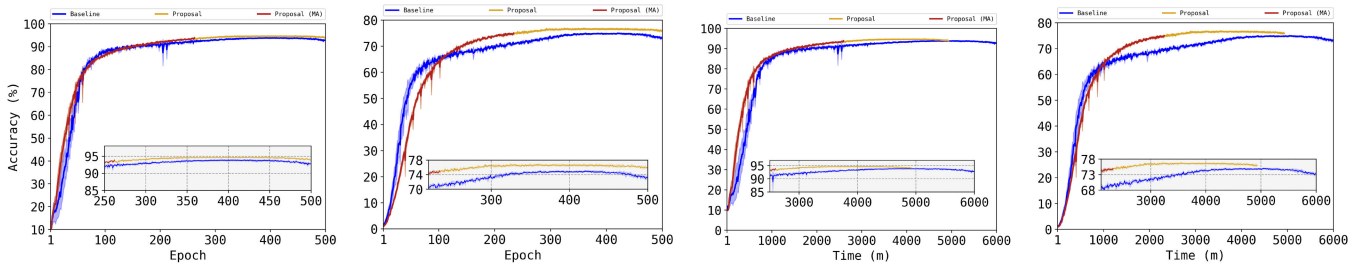


Fig. 5. Experiment 1: accuracy evolution for both schemes using ResNet101. Minutes are denoted as min for the y-axis of (c) and (d). MA denotes the proposal until the baseline maximum accuracy is reached. (a) CIFAR-10. (b) CIFAR-100. (c) CIFAR-10 (time). (d) CIFAR-100 (time).

TABLE III

EXPERIMENT 1: ACCURACY (%) AND SPEEDUP MEASUREMENTS. SPEEDUP IS SHOWN IN PERCENTAGE (%). @ INDICATES A NEARBY VALUE

Network	Nodes	CIFAR-10			CIFAR-100			Speedup
		Baseline	Proposed	MA-Acceleration	Baseline	Proposed	MA-Acceleration	
ResNet18	4	94.26±0.25	**94.32±0.01**	1.06	74.16±0.18	**74.44±0.34**	1.12	@1
	8	93.35±0.09	**93.43±0.18**	1.14	73.36±0.12	**73.62±0.17**	1.23	4.97
	12	**94.14±0.07**	94.13±0.10	1.15	73.89±0.34	**74.09±0.12**	1.70	15.12
	16	93.19±0.29	**93.77±0.12**	1.88	73.54 ±0.24	**73.54±0.05**	1.15	5.62
	20	93.40±0.17	**93.77±0.03**	1.65	**72.64±0.21**	73.62±0.24	1.92	1.55
ResNet50	4	93.23±0.41	**93.84±0.18**	1.22	75.55±0.75	**76.67±0.40**	1.28	@1
	8	91.85±0.53	**92.12±0.47**	1.44	72.64±0.76	**74.94±0.15**	1.47	5.77
	12	93.08±0.20	**93.69±0.13**	2.00	74.35±0.78	**76.33±0.44**	2.40	17.99
	16	90.81±0.30	**92.84±0.19**	2.25	73.34±0.45	**75.36±0.10**	2.22	11.02
	20	91.03±0.22	**92.75±0.05**	2.35	71.97±0.17	**75.05±0.40**	2.61	6.11
ResNet101	4	94.83±0.22	**94.85±0.06**	1.07	76.26±0.21	**77.12±0.51**	1.26	@1
	8	93.14±0.38	**93.51±0.26**	1.27	74.38±0.50	**75.76±0.52**	1.44	7.24
	12	93.92±0.22	**94.80±0.11**	2.30	75.11±0.20	**76.92±0.27**	2.57	21.47
	16	93.17±0.23	**94.54±0.23**	2.34	73.50±0.44	**76.42±0.39**	2.98	12.93
	20	92.72±0.61	**94.14±0.06**	2.46	72.46±0.58	**76.28±0.29**	3.01	7.15

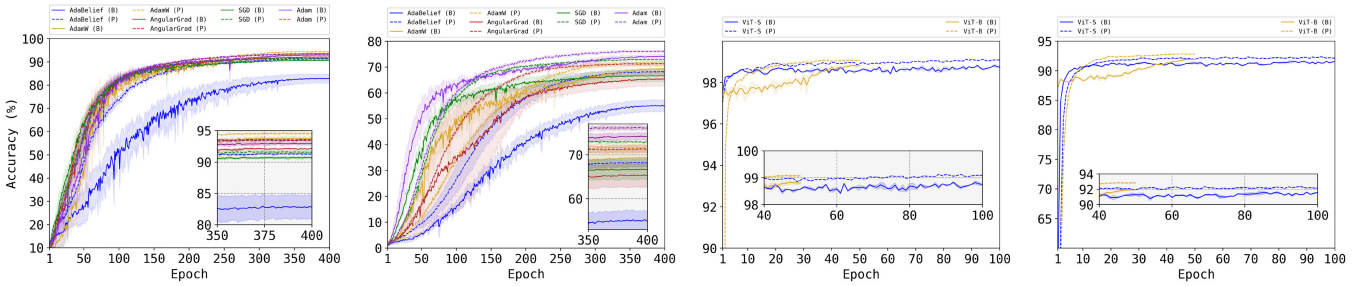
proposal to reach the maximum accuracy obtained in the baseline method.

Focusing on the analysis reliability, those implementations with the proposed communication scheme obtains better accuracy results for both datasets. The highest two accuracy values are obtained using 4 and 12 nodes for all models. As a summary of CIFAR-10 results, for ResNet18, the proposed method obtains an accuracy of 94.34% with an improvement of +0.06% compared with the baseline method for 4 nodes, whilst for 12 the difference is residual. Keeping constant the analysis on 4 and 12 nodes for the rest of the models, the obtained gains are +0.61% for both optimal node distributions in ResNet50, and +0.02% and +0.88% in ResNet101. Moreover, for CIFAR-100, notable improvements are shown for all models. Similar to CIFAR-10, best results are obtained using 4 and 12 nodes. In this regard, 74.44% and 74.09% for ResNet18, 76.67% and 76.33% for ResNet50, 77.12% and 76.92% for ResNet101. In addition, the gains obtained in comparison with the baseline method are +0.28% and +0.20% for ResNet18, +1.12% and +1.98% for ResNet50, +0.86% and +1.81% for ResNet101, respectively, for both optimal node distributions. The proposed method shows a clear improvement improving the accuracy for the different distribution alternatives. According to the number of nodes, the baseline method obtains accuracy losses as the number of nodes increases. Meanwhile, the proposal aims to maintain stable the accuracy for all node distributions. Table III shows how the proposed method always aims to reach the best accuracy for all datasets and models.

The graphic representation for the accuracy values from Table III are shown using its evolution by epoch and time in Fig. 5(a) and (b) and (c) and (d), respectively. Note that the 12 distribution is selected as the best trade-off between speedup and accuracy. Additionally, Fig. 5 also shows the accuracy evolution until the point where the proposed method reaches the same accuracy than the baseline. This evolution is denoted as *Proposed* (MA) from Table III. Moreover, speedup from Table III is shown with respect to the *Proposed* and *Baseline* plots. As a consequence, in Fig. 5(c) and (d), the rest of the time aims to increase the accuracy performance.

Last, the speedup values show that the proposal benefits from a higher number of nodes until the point that the number of iterations performed in each of them is significantly reduced. At that point, the number of intranode communications still benefit when compared to the baseline. As they are reduced, their improvement over time is affected. In these cases, even that the speedup is less, the proposal still obtains better accuracy for all cases. Indeed, expanding the parameter count within the model yields a proportionally increased acceleration.

2) Second Experiment: The objective of the second experiment is to determine the behavior of the proposal using different optimization algorithms to demonstrate its robustness. Therefore, multiple classification algorithms have



(a) CIFAR-10. (b) CIFAR-100. (c) CIFAR-10. (d) CIFAR-100.

Fig. 6. (a) and (b) Experiment 2: accuracy evolution for different optimization algorithms. *P* and *B* denotes the proposed and the baseline methodologies, respectively. (c) and (d) Experiment 3: obtained results of ViT models. *P* and *B* denotes the proposed and the baseline methodologies, respectively.

TABLE IV

EXPERIMENT 2: ACCURACY (%) RESULTS OF RESNET50 USING 12 NODES CONSIDERING BOTH CIFAR-10 AND CIFAR-100 DATASETS

Optimizer	CIFAR-10		CIFAR-100	
	Baseline	Proposed	Baseline	Proposed
SGD	90.81±0.16	**91.78±0.24**	66.86±2.19	**73.17±0.15**
Adam	93.08±0.20	**93.69±0.13**	74.35±0.78	**76.33±0.44**
AdamW	93.89±0.26	**94.67±0.14**	69.50±1.94	**71.92±0.85**
AngularGrad	92.25±0.03	**93.55±0.18**	65.53±2.68	**71.51±0.75**
AdaBelief	83.03±1.92	**91.41±0.09**	55.28±2.20	**68.37±0.95**

TABLE V

EXPERIMENT 3: ACCURACY RESULTS OF TRANSFORMER VIT MODELS USING 12 NODES. ADAM IS USED AS OPTIMIZER. BOTH DATASETS CIFAR-10 AND CIFAR-100 ARE STUDIED

Network	CIFAR-10		CIFAR-100	
	Baseline	Proposed	Baseline	Proposed
ViT-B	98.83±0.06	**99.10±0.01**	92.09±0.05	**92.68±0.05**
ViT-S	99.05±0.01	**99.18±0.01**	91.92±0.08	**92.85±0.04**

been evaluated. These algorithms are *Adam*, *AdamW* [58], *AdaBelief* [57], *AngularGrad* [59], and *SGD* [60]. Optimizers are configured used the optimal hyperparameters provided by the literature. The experiments have been conducted using the ResNet50 model with a 12-node distribution. The selection of the node distribution is determined by the previous experiment, where the best trade-off of accuracy and speedup has been demonstrated for 12 nodes. Accuracy results are shown in Table IV. The proposed method aims to obtain major accuracy improvements for the evaluated optimizers. As a resume, the best accuracy for CIFAR-10 is obtained with AdamW with a value of 94.67% and 93.89% for the proposal and baseline, respectively. Highlight the difference obtained between both methodologies with a gain of +0.78% at high-precision levels. Others optimizers such as AngularGrad also obtain notable improvements. Meanwhile, for CIFAR-100, the highest accuracy is obtained with Adam, with a value of 76.33% and 74.35%, aiming a gain of +1.98%. Note that Adam values are obtained from the previous Experiment 1. Accuracy results from Table IV are shown in Fig. 6(a) and (b).

3) Third Experiment: This experiment analyzes both baseline and proposed schemes for ViT models. Therefore, models from Table II ViT-B and ViT-S are used. Accuracy results are included in Table V. Moreover, the results are as positive as in the previous experiments. Accuracy is improved for both CIFAR-10 and CIFAR-100 datasets. Specifically, for the base model ViT-B a gain of +0.37 and +0.59 is obtained for the evaluated datasets, respectively. For the small model ViT-S, the

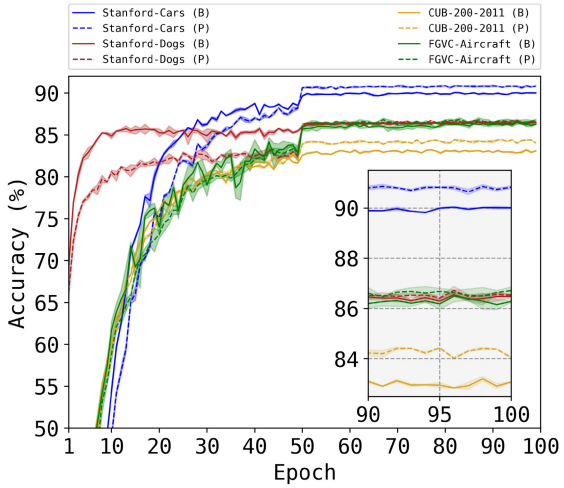


Fig. 7. Experiment 4: accuracy evolution for both schemes using ResNet50 for fine-grained datasets. *P* and *B* denotes the proposed and the baseline methodologies, respectively.

gains are +0.13 and +1.07, for both datasets. As can be seen, the accuracy percentages are very high, all of them exceeding 90%. As a consequence, obtaining accuracy improvements is difficult. Last, results from Table V are clearly shown in Fig. 6(c) and (d), where the gains and the positive trend of the proposed method can be appreciated.

4) Fourth Experiment: The fourth experiment has been conducted to evaluate the performance of pretrained models for fine-grained classification. Obtained results from the studied datasets Stanford Cars, Standford Dogs, CUB-200-2011, and FGVC Aircraft are shown in Table VI. The proposed method obtains better classification results for all datasets. Specially, for Stanford Cars and CUB-200-2011, the accuracy improvements are +0.84% and +1.17%, respectively. Fig. 7 shows how the proposal starts the training with lower accuracy values. Subsequently, as the training is maintained over time, the proposal manages to overcome the baseline results. In this way, we can determine that the proposal manages to improve training for fine-grained classification algorithms with pretrained values.

5) Fifth Experiment: This experiment performs the training for GAN to generate MNIST images with good resolution. Since previous experiments focus on classification, the main objective of this experiment is to demonstrate the effectiveness of the proposed method for a different task. The proposal

TABLE VI
EXPERIMENT 4: OBTAINED ACCURACY (%) FOR FINE-GRAINED CLASSIFICATION TASK USING 12 NODES

Stanford Cars		Stanford Dogs		CUB-200-2011		FGVC Aircraft	
Baseline	Proposed	Baseline	Proposed	Baseline	Proposed	Baseline	Proposed
90.10±0.01	**90.94±0.06**	86.60±0.09	**86.80±0.05**	83.29±0.02	**84.46±0.01**	86.59±0.15	**86.94±0.16**

TABLE VII
EXPERIMENT 6: RESULTS FOR IMAGENET-1K CLASSIFICATION USING BOTH METHODOLOGIES WITH $R = 16$ AND $M = 4$

ResNet18				ResNet34				ShuffleNet_V2			
Baseline	Proposed	Speedup	MA-A	Baseline	Proposed	Speedup	MA-A	Baseline	Proposed	Speedup	MA-A
66.93	**66.97**	1.10	1.17	70.37	**70.96**	2.34	2.52	63.37	**64.11**	1.03	1.22

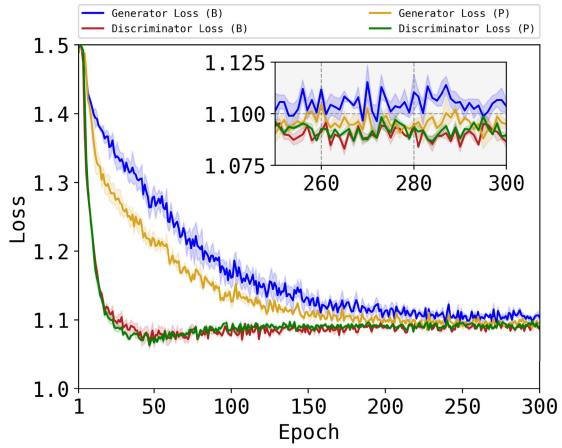


Fig. 8. Experiment 5: loss evolution of both schemes for GAN networks. Generator and discriminator are shown for each method. P and B denotes the proposed and the baseline methodologies, respectively.

(a) (b)

Fig. 9. Experiment 5: generated MNIST images for both methodologies. (a) Baseline. (b) Proposed.

aims to reduce the communication for the discriminator and generator during training. Networks are described in Table II.

The evolution of the loss functions for both networks, discriminator and generator, are shown in Fig. 8 for both baseline and proposed methods. The graphical evolution shows that the proposed method reaches lower loss values. Hence, the generation error is less for the proposal. Additionally, in Fig. 9, the generated images for the baseline and proposed methods are shown. As demonstrated, the proposal aims to generate more realistic images, clearly appreciated for the digits 5 and 8. In this context, the detection of digit borders is a crucial aspect that the proposal focuses on, aiming to improve its efficiency with an overall acceleration of 10.1%.

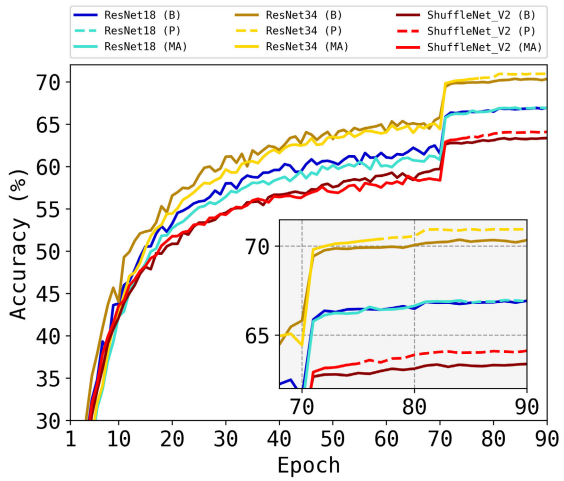


Fig. 10. Experiment 6: accuracy evolution of ResNet and ShuffleNet models for ImageNet-1K using both methodologies. P and B denotes the proposed and the baseline methodologies, respectively. MA denotes the proposal until the baseline maximum accuracy is reached.

TABLE VIII
EXPERIMENT 7: ACCURACY RESULTS OF RECURRENT-BASED MODELS FOR THE MNIST DATASET USING $R = 4$ AND $M = 2$

LSTM		GRU		RNN	
Baseline	Proposed	Baseline	Proposed	Baseline	Proposed
98.67±0.11	**98.85±0.04**	98.70±0.06	**98.90±0.08**	**96.41±0.27**	96.27±0.12

6) Sixth Experiment: This experiment evaluates the accuracy and speedup of ResNet18, ResNet34, and ShuffleNet_V2 models on the ImageNet-1K dataset. As in previous experiments, the proposal is compared against the baseline all-reduce implementation. It is observed in Fig. 10 and Table VII that the proposal obtains better performance in terms of accuracy and speedup for all models. As the complexity of the model increases, the robustness of the proposed approach is demonstrated by its ability to maintain or even improve the accuracy. The proposed method achieves the maximum accuracy earlier than the baseline approach.

7) Seventh Experiment: The last experiment aims to evaluate the behavior of different RNN-based models. Concretely, these models are long short-term memory (LSTM), gated recurrent unit (GRU), and classic RNN. Results are shown in Fig. 11 and Table VIII for both proposed and baseline communication schemes. Obtained speedup is +15.09%, +4.11%, and +4.34%, respectively. As can be observed, the accuracy values remain consistently stable for all approaches, with

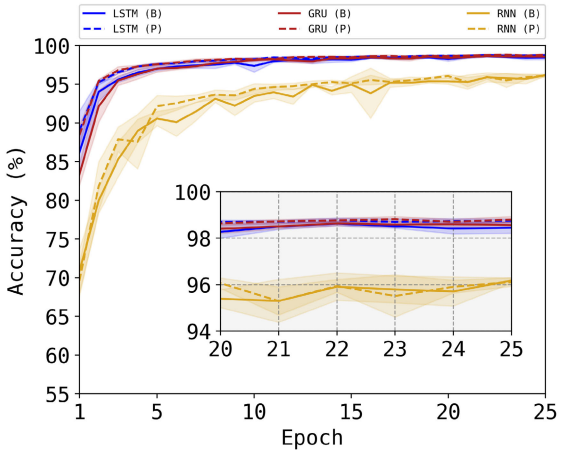


Fig. 11. Experiment 7: accuracy evolution for RNN models using MNIST. *P* and *B* denotes the proposed and the baseline methodologies, respectively.

light improvements observed in the LSTM and GRU models benefiting the proposal.

V. Conclusion

In this work, we proposed a novel node-based communication scheme, which exploits all-reduce collectives for handling replica gradients and parameters updating. In particular, two types of communications named intranode and internode have been implemented, depending on the location of the replicas in an HPC platform. Then, the number of global communications is reduced by discriminating the most expensive communications. As a result, the proposed method obtains significant speedups with respect to the baseline method extracted from the literature. The proposed communication scheduling outperforms the baseline communications in most of the experimental cases. The proposed method capitalizes on the utilization of large-scale datasets and complex neural architectures, while also extending its applicability beyond these specific cases. This is attributed to the substantial volume of data that needs to be exchanged between internode replicas. As a result, the performance achieved by the proposed method exhibits scalability in such environments. Finally, the proposed method has been tested for multiple applications, such as image classification and generation, and obtaining better overall results. The proposal also demonstrates that average the combined replica parameters helps to improve the reliability of the models for the proposed scheme. We can conclude that the proposed communication scheme is more effective and successfully manages to improve the applications performance. With an efficient communication design, we avoid the detrimental continuous exchange of gradients between replicas that increase the execution time and slow down the convergence of the model.

Acknowledgment

This work was supported by the computing facilities of the Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), Trujillo, Spain, funded by the European Regional Development Fund (ERDF). CETA-CIEMAT belongs to CIEMAT and the Government of Spain.

References

[1] G. Litjens et al., "A survey on deep learning in medical image analysis," *Med. Image Anal.*, vol. 42, pp. 60–88, Dec. 2017.

[2] J. M. Haut et al., "Distributed deep learning for remote sensing data interpretation," *Proc. IEEE*, vol. 109, no. 8, pp. 1320–1349, Aug. 2021.

[3] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 604–624, Feb. 2021.

[4] S. Moreno-Álvarez, J. M. Haut, M. E. Paoletti, J. A. Rico-Gallego, J. C. Díaz-Martín, and J. Plaza, "Training deep neural networks: A static load balancing approach," *J. Supercomput.*, vol. 76, no. 12, pp. 9739–9754, Dec. 2020.

[5] *MPI: A Message-passing Interface Standard, Version 3.1; June 4, 2015*. MPI Forum, High-Perform. Comput. Center Stuttgart, Univ. Stuttgart, Stuttgart, Germany, 2015.

[6] H. Sun, Z. Gui, S. Guo, Q. Qi, J. Wang, and J. Liao, "GSSP: Eliminating stragglers through grouping synchronous for distributed deep learning in heterogeneous cluster," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2637–2648, Oct. 2022.

[7] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, Jun. 2015, pp. 1–9, doi: 10.1109/CVPR.2015.7298594.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[9] K. Han et al., "A survey on visual transformer," 2020, *arXiv:2012.12556*.

[10] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2017.

[11] I. Goodfellow et al., "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[12] L. R. Medsker and L. C. Jain, "Recurrent neural networks," *Design Appl.*, vol. 5, pp. 64–67, Dec. 2001.

[13] S. Li et al., "PyTorch distributed: Experiences on accelerating data parallel training," 2020, *arXiv:2006.15704*.

[14] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, "Collective communication: Theory, practice, and experience: Research articles," *Concurr. Comput. Pract. Exper.*, vol. 19, no. 13, pp. 1749–1783, Sep. 2007.

[15] C. Renggli, S. Ashkboos, M. Aghagolzadeh, D. Alistarh, and T. Hoefler, "SparCML: High-performance sparse communication for machine learning," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.* New York, NY, USA: Association for Computing Machinery, Nov. 2019, doi: 10.1145/3295500.3356222.

[16] S. Shi et al., "A distributed synchronous SGD algorithm with global top-k sparsification for low bandwidth networks," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 2238–2247.

[17] M. E. Paoletti, J. M. Haut, T. Alipour-Fard, S. K. Roy, E. M. T. Hendrix, and A. Plaza, "Separable attention network in single- and mixed-precision floating point for land-cover classification of remote sensing images," *IEEE Geosci. Remote Sens. Lett.*, vol. 19, pp. 1–5, 2022.

[18] M. E. Paoletti, X. Tao, J. M. Haut, S. Moreno-Álvarez, and A. Plaza, "Deep mixed precision for hyperspectral image classification," *J. Supercomput.*, vol. 77, no. 8, pp. 9190–9201, Aug. 2021.

[19] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, "Collective communication: Theory, practice, and experience," *Concurrency Computation: Pract. Exper.*, vol. 19, no. 13, pp. 1749–1783, 2007.

[20] S. Li and T. Hoefler, "Near-optimal sparse allreduce for distributed deep learning," in *Proc. 27th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.* New York, NY, USA: Association for Computing Machinery, Apr. 2022, pp. 135–149, doi: 10.1145/3503221.3508399.

[21] S. Shi, X. Chu, K. Chun Cheung, and S. See, "Understanding top-k sparsification in distributed deep learning," 2019, *arXiv:1911.08772*.

[22] R. Rabenseifner, "Optimization of collective reduction operations," in *Proc. Int. Conf. Comput. Sci.*, vol. 3036, 2004, pp. 1–9.

[23] P. Sanders, J. Speck, and J. L. Träff, "Two-tree algorithms for full bandwidth broadcast, reduction and scan," *Parallel Comput.*, vol. 35, no. 12, pp. 581–594, Dec. 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819109000957

[24] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. H. Yum, and E. J. Kim, "Communication algorithm-architecture co-design for distributed deep learning," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 181–194.

[25] D. Clarke, Z. Zhong, V. Rychkov, and A. Lastovetsky, "FuPerMod: A framework for optimal data partitioning for parallel scientific applications on dedicated heterogeneous HPC platforms," in *Parallel Computing Technologies*. Berlin, Germany: Springer, 2013, pp. 182–196.

[26] A. Lastovetsky, I.-H. Mkwawa, and M. O'Flynn, "An accurate communication model of a heterogeneous cluster based on a switch-enabled Ethernet network," in *Proc. 12th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, 2006, p. 6.

[27] J. A. Rico-Gallego, S. Moreno-Álvarez, J. C. Díaz-Martín, and A. L. Lastovetsky, "A tool to assess the communication cost of parallel kernels on heterogeneous platforms," *J. Supercomput.*, vol. 76, no. 6, pp. 4629–4644, Jun. 2020.

[28] J. A. Rico-Gallego, J. C. Díaz-Martín, C. Calvo-Jurado, S. Moreno-Álvarez, and J. L. García-Zapata, "Analytical communication performance models as a metric in the partitioning of data-parallel kernels on heterogeneous platforms," *J. Supercomput.*, vol. 75, no. 3, pp. 1654–1669, Mar. 2019, doi: 10.1007/S11227-018-2724-8.

[29] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, and Q. V. Le, "Mesh-TensorFlow: Deep learning for supercomputers," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4235–4245.

[30] D. Coquelin et al., "Accelerating neural network training with distributed asynchronous and selective optimization (DASO)," *J. Big Data*, vol. 9, no. 1, p. 14, Dec. 2022.

[31] B. E. Woodworth, K. K. Patel, and N. Srebro, "Minibatch vs local SGD for heterogeneous distributed learning," in *Advances in Neural Information Processing Systems*, vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. Red Hook, NY, USA: Curran Associates, 2020, pp. 6281–6292.

[32] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[33] A. Khaled, K. Mishchenko, and P. Richtárik, "Tighter theory for local SGD on identical and heterogeneous data," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 4519–4529.

[34] L. Zhang, S. Shi, X. Chu, W. Wang, B. Li, and C. Liu, "Decoupling the all-reduce primitive for accelerating distributed deep learning," 2023, *arXiv:2302.12445*.

[35] H. Zhang et al., "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *Proc. USENIX Annu. Tech. Conf.*, vol. 1, 2017, pp. 1–2.

[36] M. Cho, U. Finkler, and D. Kung, "Blueconnect: Novel hierarchical all-reduce on multi-tired network for deep learning," in *Proc. 2nd SysML Conf.*, 2019, pp. 1–5.

[37] A. Sergeev and M. D. Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," 2018, *arXiv:1802.05799*.

[38] S. Li et al., "Colossal-AI: A unified deep learning system for large-scale parallel training," 2021, *arXiv:2110.14883*.

[39] N. Bogoychev, K. Heafield, A. F. Aji, and M. Junczys-Dowmunt, "Accelerating asynchronous stochastic gradient descent for neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.* Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 2991–2996. [Online]. Available: https://aclanthology.org/D18-1332

[40] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, 2018, pp. 1306–1316.

[41] N. Ivkin et al., "Communication-efficient distributed SGD with sketching," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–11.

[42] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, "Communication-efficient distributed deep learning: A comprehensive survey," 2020, *arXiv:2003.06307*.

[43] S. Ouyang, D. Dong, Y. Xu, and L. Xiao, "Communication optimization strategies for distributed deep neural network training: A survey," *J. Parallel Distrib. Comput.*, vol. 149, pp. 52–65, Mar. 2021.

[44] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," 2016, *arXiv:1609.04836*.

[45] M. Zinkevich, M. Weimer, L. Li, and A. Smola, "Parallelized stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 23, 2010, pp. 1–9.

[46] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.

[47] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. TR-2009, 2009.

[48] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3D object representations for fine-grained categorization," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, Dec. 2013, pp. 554–561, doi: 10.1109/ICCVW.2013.77.

[49] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, "Novel dataset for fine-grained image categorization: Stanford dogs," in *Proc. CVPR Workshop Fine-Grained Vis. Categorization (FGVC)*, vol. 2, no. 1, 2011, pp. 1–2.

[50] P. Welinder et al., "Caltech-UCSD birds 200," California Inst. Technol., Pasadena, CA, Tech. Rep. CNS-TR-2010-001, 2010.

[51] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, "Fine-grained visual classification of aircraft," 2013, *arXiv:1306.5151*.

[52] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[53] E. Suarez, N. Eicker, and T. Lippert, "Modular supercomputing architecture: From idea to production," in *Contemporary High Performance Computing*. Boca Raton, FL, USA: CRC Press, 2019.

[54] M. Zhu, Y. Tang, and K. Han, "Vision transformer pruning," 2021, *arXiv:2104.08500*.

[55] A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.

[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[57] J. Zhuang et al., "AdaBelief optimizer: Adapting stepsizes by the belief in observed gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1–12.

[58] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017, *arXiv:1711.05101*.

[59] S. K. Roy et al., "AngularGrad: A new optimization technique for angular convergence of convolutional neural networks," 2021, *arXiv:2105.10190*.

[60] B. Léon, "Stochastic gradient learning in neural networks," *Proc. Neuro-Nimes*, vol. 91, no. 8, p. 12, Jan. 1991.