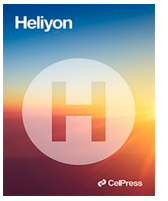
Heliyon 10 (2024) e23567

Contents lists available at ScienceDirect

Heliyon

journal homepage: www.cell.com/heliyon



DPro-SM – A distributed framework for proactive straggler mitigation using LSTM

Aswathy Ravikumar [a], Harini Sriraman [a,*]

[a] School of Computer Science and Engineering, VIT, Chennai, 600127, India

ARTICLE INFO

Keywords:
Distributed deep learning
Data parallel
Stragglers
MPI
LSTM
Proactive mitigation

ABSTRACT

The recent advancement in deep learning with growth in big data and high-performance computing is Distributed Deep Learning. The rapid rise in the volume of data and network complexity has led to significant growth in DDL. Distribution of the network leads to high communication and computation among the nodes, which leads to high training time and lower accuracy. The primary reason for the delay in communication is the presence of straggler nodes which causes the bottleneck in communication. Due to the enormous volume of parameter transfer, Distributed Deep Learning's data parallelism incurs substantial communication costs. The newly developed model-parallel methods may minimize the communication effort; however, this results in load imbalance and severe straggler issues: the proposed model DPro-SM, a distributed framework for proactive straggler mitigation using LSTM in distributed deep learning. DPro-SM uses LSTM to predict the completion time of each worker and proactively allocates resources to reduce the overall training time. The results show that DPro-SM can significantly reduce the training time and improve the scalability and efficiency of large-scale machine learning tasks.

1. Introduction

Utilization of deep learning on a large scale for picture analysis in various real-world applications. It is required to spread models over several nodes because of the rapid increase of the data and the models' size. The model's scalability, the amount of time it takes to train, and the amount of money it saves all increase thanks to distributed processing. However, the distribution could make calculation times longer when nodes in the network are no longer in use. Various factors, including communication-induced delay, network connection, resource sharing, and computational capability, can all influence the amount of time it takes for remote nodes to complete a calculation. The fundamental problem with distribution is that it might result in staleness among the worker nodes. It is hard to minimize the influence of stragglers in dispersed clusters because of their widespread distribution [1,2]. The most common reasons for stragglers are problems with the storage or the disks, workload imbalances, resource sharing, and other factors. Straggler nodes are responsible for the delay in synchronization during each iteration's aggregation step. The repetitive nature of SGD algorithms used in distributed deep learning leads to the generation of outliers, which in turn causes delays in throughput and accuracy. The calculation cycles wasted on stragglers need to be recovered so we may continue. In distributed deep learning, stragglers are often eliminated by utilizing node replication, extra resources, primary backup, and failing nodes' bypasses. Nevertheless, this strategy does not maximize resource efficiency and is expensive.

* Corresponding author.
E-mail addresses: aswathyravi2290@gmail.com (A. Ravikumar), harini.s@vit.ac.in (H. Sriraman).

https://doi.org/10.1016/j.heliyon.2023.e23567
Received 23 June 2023; Received in revised form 16 October 2023; Accepted 6 December 2023
Available online 10 December 2023

Deep neural networks (DNNs) are often more profound, more complicated, and trained on a more significant number of datasets to achieve high-performance levels. Due to rapidly expanding data volumes and model sizes, enormous amounts of computation have been required. This demonstrates that DNN learning is time-intensive and may take several days. Utilizing high-performance hardware isn't the only option; one potential alternative is to parallelize and provide DNN training operations on many nodes instead. Under these circumstances, the amount of work that each node contributes to the calculation is minimal at best [3–7]. Despite this, the communication delay is a critical obstacle in distributed training due to the frequent communication requirements for delivering vast volumes of data across multiple computing nodes. As the size of the complex has increased, the amount of overhead devoted to communications has soared. Since the bulk of training time has been spent on data transmission, this behavior drastically limits the advantage that may be gained via parallel training. Adopting high-performance hardware accelerators simply reduces the overhead associated with computing while keeping the overhead associated with communication unaltered [8–10]. This results in a more significant proportion of time being spent on communication.

Partly and distributed processing is frequently required to solve the learning and optimization problems that arise at the present size. Distributed implementations for possibly prohibitive computational and memory constraints [11]. On the other hand, networked implementations of this kind usually face issues at the system level. Communication that is too slow and computational nodes that need to be balanced are also issues. A few sluggish nodes are responsible for most distributed systems' poor performance. Their holdups, also known as stragglers or a handful of sluggish communication lines, place a tremendous load on the network.

Previous publications on distributed deep learning have covered various academic issues in their discussions. Ben et al. [2] conducted a comprehensive study of the concurrency of DNNs at different levels in their research. Chahal et al. [12] investigated various training frameworks and algorithms, including the most recent and cutting-edge ones. Mayer et al. [13]conducted recent research investigating the administrative issues related to deploying large-scale deep learning algorithms on a distributed network. Mittal et al.'s [14] disclosure of GPU architecture-level and system-level optimization methods can benefit deep learning applications. The article [15] discusses the difficulty of predicting the efficiency of manufacturing procedures in cyber-physical systems when the products undergo hundreds of operations and when the data collected during the manufacturing processes is insufficient to enable precise predictions and, consequently, to identify those operations that may produce low performance results.

The work [16] has developed and tested a novel system for predicting the spread of wildfires. The strategy combines approaches for data assimilation, error covariance adjustment, long-short-term memory recurrent neural networks, and reduced-order modeling. In many situations, it has been demonstrated that using machine learning surrogate models and less-Order Modeling (ROM) may considerably increase efficiency. The system may approximate complicated processes using these approaches with less computer resources, which lowers computational costs. Additionally, the use of data assimilation techniques makes it easier for the system to adjust to actual observations, improving its capacity to faithfully reflect and react to the environment's dynamic character. Three significant wildfires that happened in California between 2017 and 2020 were replicated and predicted using these methods. The computational efficiency of the deep-learning surrogate model outperforms that of ordinary Cellular Automata simulations by a factor of about 1000. Data assimilation uses daily fire perimeters acquired from satellites as input, allowing for real-time modifications to the fire forecast. Furthermore, errors in simulations and observations are estimated using an error covariance tuning technique in the condensed space. According to the results of our investigation, using covariance correction and data assimilation techniques significantly reduces root mean square error (RMSE) by around 50 %. This significant reduction in mistakes has a significant impact on predicting accuracy. This study represents the first development in the field of reduced-order wildfire spread forecasting and illustrates how machine learning models powered by data may improve the accuracy of fire forecasting for a variety of real-world uses.

High-dimensional dynamical systems use reduced-order modeling and low-dimensional surrogate models to improve algorithmic efficiency with machine learning methods. The work [17] develops a system that merges reduced-order surrogate models with a novel data assimilation (DA) technique to include real-time observations from several physical domains. In this work, we use local smooth surrogate functions to connect encoded system variables with current observations. This allows variational data assimilation with little computing. The generalized latent assimilation system may improve reduced-order modeling's computational efficiency and data assimilation precision. This paper also explains surrogate and original assimilation cost functions theoretically. Its upper restriction depends on the local training set size. A computationally expensive high-dimensional CFD program evaluates the proposed technique.

2. Background

When training neural networks, there are two types of parallelization techniques: model parallel and data parallel. Model parallelism necessitates "splitting" the learning model and distributing its "pieces" among many computer nodes to get the desired results. For instance, we might place the first section of the layers on a single GPU and the second portion on a different GPU. Both GPUs would be responsible for processing the layers. Alternatively, you could split the layers along the center and give each one to a different GPU [18]. This tempting method is only sometimes put into action since there is insufficient time for communication among the various devices. Each processing device has a copy of a worker or replica, which is also known as a replica of a learning model. After calculating the gradient on their own data fragments, the replicas integrate their results to update the model parameters. The main idea behind data parallelism is to raise the total throughput rate by replicating the model over numerous computers, where backpropagation may be run in parallel to collect more data about the loss function more quickly. This is done to enhance the overall throughput rate. The parallel processing of data is accomplished primarily through the following means: Every node in the cluster gets a copy of the most recent model first. Backpropagation is carried out in parallel by each node, with each node utilizing its data assignment. After that, the results are pooled, and the new model incorporates them. This is allowed in deep neural networks since most changes made to a training sample do not contain data from other samples. This is why it is legal for this to happen. It is common practice to cut the

number of computations and the number of intermediate tensors in half when decreasing the number of training samples. This speed up the backpropagation process and is beneficial for big models. Because it is necessary to send per-parameter gradients or model parameters between computers, the relationship between the size of the model and the bandwidth available on the network decides whether data parallelism may expedite the training process. Soon, problems linked to bandwidth may restrict the scalability of massive models.

The data input on the CPU and training on the GPU would happen simultaneously. Nevertheless, the CPU makes the process of encoding data more time-consuming. This is due to a combination of three different variables. CPU computing limitation. As was mentioned before, the central processing unit (CPU) is primarily responsible for loading data. If the CPU is sluggish, it cannot load data with the same efficiency as the GPU. Provides a more in-depth explanation of the CPU data loading sensitivity. Augmentation and large-scale picture decoding comprise most of the CPU operations the CNN model performs. The decoding stage of the data input pipeline ate up fifty percent of the available time. Large datasets size the successful training of models is only possible with datasets. Increases the likelihood that the dataset will only be stored partially in memory. In other words, it leads to varying data access costs for each dataset sample throughout the loading phase.

The impact of lagging in widely spaced DNN lessons in distributed training is that every graphics processing unit (GPU) has the information injector that it uses to service data from the partitioned dataset. The expenses of the data pipeline for each data injector vary when using partitioned dataset samples and randomized data augmentation. The straggler effect happens when waiting for other computers to synchronize the gradients on their displays. The total training throughput drops because the GPU's data processor with the lowest capacity recovers after a data loading surge. The current approaches and DNN platform need help to solve the problem of students falling behind in their coursework.

3. Stragglers

Numerous applications in the real world make substantial use of deep learning for picture analysis. It is essential to distribute models over a few different nodes due to the rapid increase of both the data and the models' size. The model's use of distributed processing improves its scalability, training time, and cost-efficiency. However, the distribution could result in longer calculation times when there are outdated nodes in the network. The amount of time it takes dispersed nodes to complete a calculation is determined by various parameters. Some of these elements include communication-related delay, network connection, resource sharing, and processing capability.

Regarding distribution, the most significant barrier to overcome is the level of obsolescence present in the worker nodes. When dealing with distributed clusters, it is hard to prevent the consequences of stragglers completely. Various factors can bring on stragglers, the most prominent of which are flaws in storage and drives, imbalanced tasks, and resource sharing. Because of stragglers, calculation durations might become significantly lengthier, and model performance can suffer. The amount of staleness is what decides whether the model will converge. The fact that the staleness of deeper models decays at a slower rate than that of shallower parts is the key factor that defines the rate at which an algorithm converges. The essential method for usage in deep learning is stochastic gradient descent, or SGD for short. When dealing with vast amounts of data, parallel SGD workers might get stale; hence, convergence and strategies for overcoming staleness and establishing stability are significant. Lagging nodes are the source of the synchronization delay during each iteration's aggregation step. In distributed deep learning, the repetitive nature of SGD algorithms creates outliers, which in turn causes a delay in throughput and precision [19]. There is a loss of computing cycles since the remains must be retrieved. In distributed deep learning, stragglers are often eliminated by utilizing node replication, extra resources, primary backup, and the bypassing of failing nodes; nevertheless, this strategy does not maximize resource efficiency and is costly.

Graphics Processing Units have developed into a very effective tool for speeding neural network models by adopting a kernel-based execution strategy and focusing on optimizing individual kernels for maximum resource consumption and high performance. This has enabled the GPU to become a highly effective instrument for accelerating neural network models. This approach launches kernels in sequential order, which results in efficient and adequate exploitation of the GPU's capabilities due to the limited space available for optimization within a single kernel. It is necessary to have a lightweight parallelization module that makes the most of the latest GPUs' capacity for concurrent kernel execution to get around this disadvantage.

4. Existing methods

Straggler mitigation is critical in distributed computing systems to improve overall efficiency and performance. Stragglers are nodes or workers that significantly lag others, causing delays and impacting the completion time of tasks or computations. Mitigating the impact of stragglers is essential to ensure the timely completion of distributed computing tasks.

Existing literature on straggler mitigation focuses on developing techniques and algorithms to address this challenge [19–21]. Various approaches have been proposed to detect, handle, and minimize the impact of stragglers in distributed systems. These approaches aim to optimize resource utilization, reduce computation time, and enhance system reliability. One common approach is task replication, where tasks are duplicated and assigned to multiple workers. The results from the first completed task are used, while the others are discarded. This approach helps mitigate the impact of slow workers but incurs additional computation and communication overhead. Speculative execution is another widely explored approach, where multiple copies of tasks are launched, and the result of the fastest completed task is used. This proactive technique aims to handle potential stragglers but can lead to resource wastage if predictions are inaccurate. Other approaches involve workload redistribution, adaptive task scheduling, and fault detection and recovery mechanisms. Workload redistribution strategies aim to balance the workload across workers by dynamically migrating tasks

Table 1
Existing straggler mitigation methods.

Methods	Details	Reference
Minimizing delay when stragglers occur using adaptive distributed SGD	• The approach involves waiting for the responses of the fastest k workers out of a total of n workers before updating the model. • The value of k determines the balance between the runtime of SGD (convergence rate) and the model's error. • An adaptive policy is proposed to optimize this trade-off by adjusting k based on an upper bound on the error derived from wall clock time. • The adaptive distributed SGD method utilizes a statistical heuristic for implementation.	[21]
Adaptive cache pre-forwarding policy for distributed deep	• A new cache mechanism called cache pre-forwarding decreases synchronization and network blocking times. • Cache pre-forwarding utilizes reinforcement learning to train a policy that increases the cache hit rate. • The adaptive nature of the policy makes it suitable for various computing environments.	[27]
Gradient Coding	• Gradient coding techniques on Amazon EC2 instances. • The proposed schemes introduce computation overhead while maintaining the same level of communication. • The advantage of this additional computation is enhanced fault tolerance, enabling recovery of full gradients even if certain machines fail to deliver their assigned work or experience delays. • Partial straggler schemes offer fault tolerance while allowing all machines to contribute partial work.	[28]
Iterative convergent parallel ML	• Use a flexible synchronization approach that enables workers to dynamically swap tasks with one another. • The model enables the redistribution of tasks among workers in real-time. • The flexibility of the synchronization model allows for efficient adaptation to changing workloads and resource availability. • Use a flexible synchronization approach that enables workers to dynamically swap tasks with one another.	[29]
Data Encoding	• Multiple encoding schemes are utilized to demonstrate the effectiveness of popular batch algorithms, including gradient descent and L-BFGS. • The algorithms are applied coding-oblivious, resulting in sample path linear convergence. • Convergence is achieved using a varying subset of nodes at each iteration, allowing for flexibility in the selection process. • The algorithms deterministically converge to an approximate solution of the original problem.	[30]
Near-Optimal Straggler Mitigation Methods	• The master coverage of calculated partial gradients is the main goal of the suggested BCC system. • Batching and coupon collection make up the two phases of BCC. • During the batching stage, the employees divide training samples at random into batches for regional processing. • The processing outcomes from the data batches are gathered at the master during the coupon collecting stage, which is similar to the well-known coupon collector problem.	[22]
Exploiting Data Dependency	• Based on the value of the data to other workers, parallel processes prioritize synchronization while considering data dependencies. • To increase computational efficiency, the Grid Graph load balancing and splitting approach is created. • Grid Graph minimizes the quantity of exchanged data and guarantees workload equality among employees by utilizing the spatial and connectivity aspects of the simulation environment.	[31]
Tuple Scheduling	• Hone introduces a tuple scheduler that minimizes the maximum queue backlog of tasks over time. • The scheduler utilizes an online Largest-Backlog-First (LBF) algorithm. • The LBF algorithm has a proven competitive ratio, ensuring efficient tuple scheduling. • Hone's tuple scheduler aims to optimize task management and minimize backlog efficiently.	[32]
Redundancy Techniques	• By encoding the dataset, the distributed optimization framework uses an over-complete representation with redundancy. • At each cycle, the algorithm treats errant nodes as missing or "erasures." • Embedded redundancy makes up for the loss stragglers cause. • Even when stragglers are disregarded, several optimization methods (including gradient descent, L-BFGS, and proximal gradient) working under data parallelism converge to a rough solution for quadratic loss functions. • The architecture allows for effective optimization even when there are drift nodes.	[33]
Collaborative Learning	• A collaborative learning-based approach is introduced for predicting stragglers using the alternate direction method of multipliers (ADMM). • The ADMM approach is resource-efficient and effectively mitigates stragglers without transferring data to a centralized location. • The framework facilitates information sharing among different models, enabling the utilization of larger training datasets and reducing training time by eliminating data transfer. • Rigorous evaluations are conducted on various datasets, demonstrating high accuracy results of the proposed method.	[34]

(*continued on next page*)

Table 1 (*continued*)

Methods	Details	Reference
	• The approach offers an efficient and accurate solution for straggler prediction in collaborative learning scenarios.	
Optimizing resources to mitigate stragglers through virtualization in run time	• Conducting statistical analysis of straggler-related metrics in Cloud computing systems. • Identifying the most suitable stragglers to prioritize for mitigation. • Utilizing modeling and prediction techniques to prevent straggler occurrences by assessing machine node performance. • Creating a specialized algorithm to handle scenarios where speculative execution is not feasible. • The focus is on analyzing, identifying, and proactively addressing straggler issues within Cloud computing systems.	[35]
Review and Analysis of Straggler Handling Techniques	• Longest Approximation Time to End Scheduling Algorithm: An algorithm that prioritizes tasks based on their estimated completion times, with a focus on the longest approximated time to completion. • Self-Adaptive MapReduce Scheduling Algorithm: An algorithm that dynamically adjusts the scheduling of MapReduce tasks based on the changing workload and system conditions. • Wrangler: A resource-efficient task scheduling approach that eliminates the need for task replication. It incorporates a confidence measure to address modelling errors and ensure reliable task scheduling.	[36]
Slack Squeeze Coded Computing	• Proposed Slack Squeeze Coded Computation (S2C2) as a dynamic workload distribution strategy for coded computation. • S2C2 efficiently utilizes the compute slack (overhead) inherent in coded computing frameworks. • Work is assigned to fast and slow nodes based on their respective speeds, without requiring data redistribution. • S2C2 optimizes workload distribution without the need for data redistribution, enhancing efficiency in coded computation.	[37]
Multiple Parallelism	• Introducing a novel methodology called "straggler projectionto thoroughly examine stragglers and provideng practical guidelines for addressing this issue. • Straggler projection focuses on two key aspects: • Controlling the training speed of each worker through elastic training parallelism control. • Transferring blocked tasks from stragglers to pioneers, maximizing the utilization of computational resources. • The methodology offers comprehensive insights and guidelines to tackle stragglers in distributed computing environments effectively.	[38]

from overloaded to underutilized workers. Adaptive task scheduling algorithms adjust task assignments based on worker performance characteristics, optimizing task allocation and mitigating stragglers. Fault detection mechanisms diagnose and eliminate machines with faulty hardware or reroute tasks away from underperforming nodes. Machine learning techniques have also been explored in straggler mitigation. Predictive models based on historical data or real-time monitoring can identify potential stragglers and take proactive measures such as task replication or workload redistribution.

Drop Stragglers [22]: This approach involves discarding the gradients from the slowest machines. However, this method can be ineffective as it reduces the adequate mini-batch size and increases gradient variance, leading to slower convergence.

Advantages: Simple to implement and can save computation time by discarding slow gradients. Smaller effective mini-batch sizes can lead to slower convergence due to increased gradient variance. More advanced techniques are needed to address the challenges posed by dropping stragglers, such as reducing the impact on convergence rate while improving overall efficiency.

Backup Workers [23]: This method involves having backup workers that can take over the tasks of slow workers. The gradients from the slowest b workers are dropped when they arrive. Enables the system to continue progressing by utilizing backup workers, mitigating the impact of stragglers on the overall computation. Requires additional resources in the form of backup workers, which can increase costs. It may also introduce additional communication overhead. Further research is needed to optimize the selection and management of backup workers and address the communication overhead introduced by this method.

Blacklisting [24]: This approach periodically diagnoses and eliminates machines with faulty hardware from the computing cluster.

Ensures the removal of machines with faulty hardware, preventing their impact on the overall computation. It relies on periodic diagnosis, which may not capture all instances of faulty hardware. The process of removing machines can disrupt the computation and introduce additional overhead. There is a need for more sophisticated fault diagnosis mechanisms and efficient strategies for removing faulty machines without disrupting the overall computation.

Speculative execution [25,26]: This method involves waiting and observing the relative progress rates of tasks and then launching copies of tasks that are predicted to be stragglers. Allows for proactive handling of potential stragglers, improving overall computation time by launching additional tasks. Speculative execution may lead to resource wastage if predictions of stragglers are inaccurate. It may not be well-suited for small jobs where the benefits of speculative execution are limited. More research is needed to develop accurate prediction models for identifying potential stragglers and refining the task-launching strategy to minimize resource wastage in speculative execution. Alternatively, approaches may need to be explored for small-scale jobs where speculative execution is less effective. The existing approaches for straggler mitigation are listed in Table 1.

The prevalence of small jobs, with 82 % of them containing fewer than ten tasks, necessitates the development of a new prediction

method for stragglers. These small jobs are often interactive and have strict latency constraints, making them highly vulnerable to the impact of stragglers. Predicting stragglers in small jobs is statistically challenging, requiring a more extended waiting period to achieve accurate predictions. Moreover, as small jobs execute all their tasks simultaneously, waiting for task completion can consume a significant portion of the job's duration.

5. Proposed model

The proposed model for straggler mitigation is designed based on the MPI parameter server data parallel model and LSTM model for proactive straggler mitigation as shown in Fig. 1. The proposed model DPro-SM Distributed framework for Proactive Straggler Mitigation using LSTM. The straggler node is proactively identified and replaced by the output of the n LSTM node. The proposed model prevents a vanishing gradient problem by utilizing the LSTM model—the LSTM node aids in proactive straggler mitigation by replacing the function of the straggler node. During the occurrence of a straggler node, a delay occurs, and the model detects the presence of the straggler, and the LSTM node substitutes the weights for the straggler node.

1. Experimental Setup

The experimental setup is made in AWS Sage Maker Notebook instance type ml.t2.2xlarge with Elastic Inference and volume size 5 GB EBS, ARN -arn: aws:sagemaker:us-east-1:174919681671:notebook-instance/cnn and Amazon Linux 2, Jupyter Lab 1(notebook-al2-v1) with 1 IMDS Version. The model was implemented using the MPI framework using the parameter server logic for the data-parallel model.

2. SGD

The stochastic gradient descent (SGD) technique is the foundation of most contemporary machine learning algorithms. Therefore, boosting the stability and convergence rate of SGD algorithms is crucial for making machine learning algorithms efficient and rapid [39,40]. SGD is typically executed serially on a single node. Running SGD serially on a single server may be prohibitively slow for big datasets [41]. Using the same settings for each node's forward propagation, transmit a small sample of distinct data to every node, calculate the gradient conventionally, and return the gradients to a root node. This stage is asynchronous since the speeds of the individual nodes vary somewhat. Once we get all of the gradients, perform synchronization, compute the mean of the gradients and utilize that value to update the parameters. Repeat the same steps for the subsequent iteration [42].

The PS model comprises four phases as described below:

- Initialization: Initially, all worker nodes receive the model weights from the centralized parameter server.
- Local Training: Each worker node trains its local model using its specific training data partition and generates local gradients. Subsequently, these local gradients are uploaded by each worker node to the centralized PS.
- Gradient Aggregation: After receiving all the gradients from the worker nodes, the parameter server aggregates them together.
- Parameter Update: Once the aggregated gradient has been computed, the parameter server utilizes this information to update the model's parameters on the centralized server.

3. LSTM Model

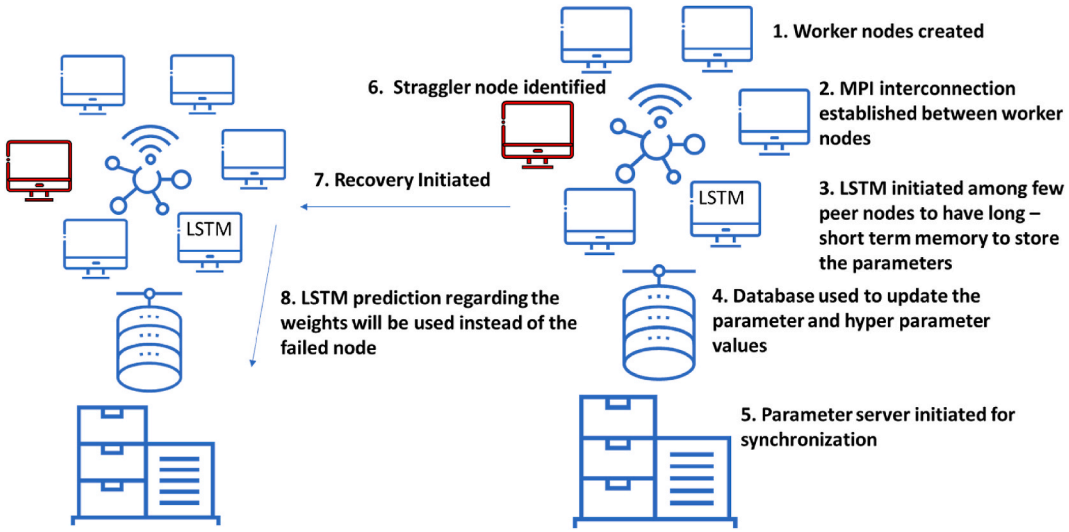


Fig. 1. Proposed model.

Vanishing gradient problems occur when training deep neural networks using stochastic gradient descent (SGD) or its variants, such as mini-batch gradient descent. It primarily affects deep networks with many layers, making it difficult for the model to learn and converge effectively. In the context of SGD, the vanishing gradient problem refers to the phenomenon where the gradients calculated during backpropagation become extremely small as they propagate backward through the network layers. This occurs when the network weights and biases are updated based on the gradients multiplied by the learning rate. The problem arises because of how gradients are calculated using the chain rule in backpropagation. When the network has many layers, the gradients are multiplied during the backward pass, leading to an exponential decrease in their values as they propagate from the output layer to the input layer. Consequently, the weights of the early layers receive only tiny updates during training significantly, slowing down their learning proceed. The vanishing gradient leads to stragglers in the parallel data model.

LSTM (Long Short-Term Memory) nodes are designed to overcome the problem of vanishing gradients in recurrent neural networks. One of the key features of LSTM nodes is their ability to selectively retain or forget information over long sequences, allowing them to capture long-term dependencies effectively. This property helps mitigate the vanishing gradients problem commonly occurring in recurrent neural networks. LSTM nodes achieve this by incorporating gating mechanisms that control the flow of information. These gates, including the input, forget, and output gates, regulate the information flow within the node. Using these gates, LSTM nodes can selectively propagate gradients through time and prevent them from vanishing or exploding during backpropagation. The gates provide a way to learn which information is relevant to retain and propagate through time steps, enabling the LSTM node to capture long-range dependencies and preserve important context.Additionally, the LSTM architecture includes a memory cell that stores and updates the information over time. This memory cell helps alleviate the vanishing gradients problem by providing a stable path for gradient flow. The LSTM node for straggler handling is shown in Fig. 2.

The LSTM model is essential for improving the distributed deep learning process in the DPro-SM framework. Recurrent neural networks (RNNs) of the LSTM variety are particularly well suited for jobs requiring sequence prediction because they can effectively capture long-term relationships in sequential input. The LSTM model is used in this architecture to forecast individual worker completion times inside the distributed deep learning system.Based on two main sources of data—their previous completion times and the condition of the distributed system—the LSTM model in DPro-SM is trained to predict the completion time of each worker. To do this, a dataset of historical worker completion times and related snapshots of the system's state is used to train the model. Through training, the LSTM model may discover complex patterns and correlations that affect the estimated completion time for each worker's duties.This proactive quality of this forecasting skill is important. The DPro-SM framework may forecast which employees may become stragglers, or those who are likely to take longer to complete their given tasks, by applying the LSTM-based predictions. With this information, DPro-SM may proactively deploy extra resources or tasks to reduce the effect of stragglers on the total training duration. The reduction of training time and improvement of the scalability and effectiveness of large-scale machine learning systems depend critically on proactive resource allocation and straggler mitigation.

6. Result analysis

The proposed model DPro-SM using the LSTM node is implemented in the dataset [43]. The dataset comprises 5863 X-Ray images labeled as either Pneumonia or Normal. The proposed model is compared with the existing straggler mitigation methods. The cost analysis of the proposed method DPro-SM is done and compared with the existing state of art methods and shown in Table 2. The proposed method has a high-speed medium delay and proactive mitigation with low cost per hr.

The straggler mitigation percentage of the proposed model was compared with existing works and shown in Fig. 3. The proposed model was found to have around 98 % straggler mitigation. Fig. 4 shows the proposed model's e accuracy and training time concerning straggler nodel.

Fig. 5 shows the timing analysis of the proposed model with the different straggler stages. The early mitigation using DPro-SM reduces the time in the presence of a straggler compared to straggler detection and mitigation approaches. The DPro-SM

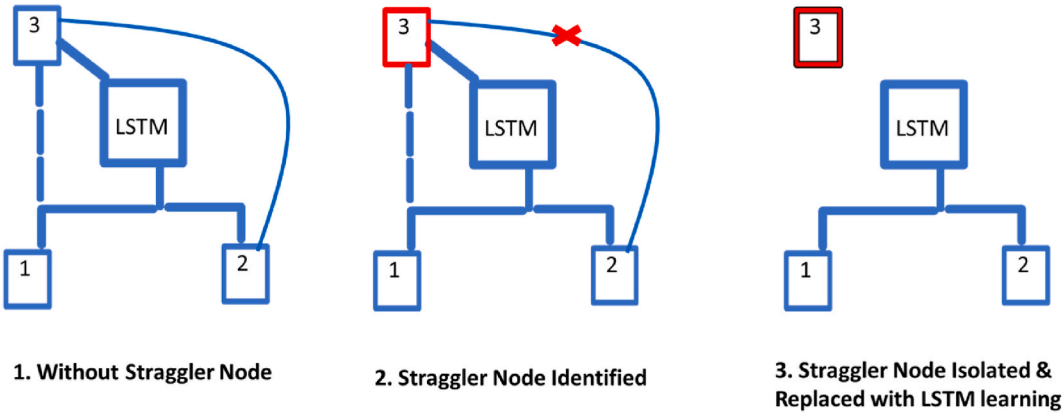


Fig. 2. LSTM node for straggler handling.

Table 2
State of Art methods.

Model	Standby nodes	Disk Usage	Memory Usage	Speed	Delay	Methods	Features	Cost
GRASS Straggler Handling Mechanism [44]	Few nodes	No	Medium	Low	High	Reactive	Speculation as a means of mitigating stragglers reactively. Two algorithms, one for resource-aware scheduling and the other for greedy speculation	$2.17 per hr
Dolly Straggler Handling Mechanism [45]	Multiple nodes (High Replication)	No	High	Good for a few tasks only	No	Proactive – cloning	Forks duties into numerous clones that are executed concurrently within their budgetary constraints. Using the CPU utilization of tasks and the Upper-Confidence-Bound, the number of clones is calculated.	$4.896 per hr
Stochastic Gradient Coding [42]	Multiple Standby (High Replication)	No	High	Low	High	Proactive	Distributed gradient calculation utilizing a pair-wise balancing scheme for cloned task execution.	$4.896 per hr
LATE [52] (Last Approximate Task to Execute)	T nodes (Medium Replication)	No	Medium	Low	High	Reactive	LATE consumes more time on superfluous suppositions. Prioritize speculative tasks, choose rapid nodes to execute on, and limit speculative tasks to avoid thrashing.	$2.17 per hr
Wrangler [46]	Thread rescheduling (No Replication)	High	Low	Medium	High	Proactive	By deferring the start of tasks predicted as straggler heartbeats, this linear modelling approach reduces the use of excess resources.	$0.399 per hr
Proposed Model D-Pro-SM	1 node (Minimum Replication)	No	High	High	Medium	Proactive	Proactive Mitigation LSTM-based model. Overcome the vanishing gradient problem	$0.399 per hr

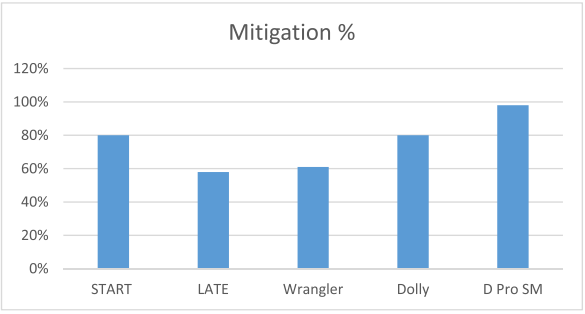


Fig. 3. Mitigation % of DPRo-SM.

framework employs the LSTM technique in order to forecast the estimated duration for each worker's task completion. The LSTM is a specific form of architecture within the broader category of recurrent neural networks (RNNs). It is specifically designed to effectively simulate sequences and time series data. Within the framework of DPro-SM, the LSTM model is employed to generate forecasts for the anticipated completion times of individual workers' assignments or computations.

The selection of LSTM is motivated by its capability to effectively capture dependencies included in sequential data, as well as its proficiency in handling long-range dependencies. This attribute proves to be particularly advantageous when making predictions regarding the completion durations of workers operating inside a distributed computing or parallel processing framework. Through the examination of historical data and the observation of worker behaviors, LSTM models has the capability to acquire knowledge on patterns and trends that serve as indicators for estimating the duration required by a certain worker to complete their designated duties.

The utilization of LSTM-based predictions for worker completion times might enhance the efficiency of load balancing and resource allocation in DPro-SM. This framework enables informed decision-making on task assignment to employees, taking into account their anticipated completion times. This phenomenon has the potential to enhance resource allocation and enhance the overall efficiency of systems in the context of large-scale machine learning and data processing activities. Graph load balancing and partitioning methods are commonly employed in the field of distributed computing and parallel processing to enhance computational efficiency while dealing with extensive graphs. The objective of these strategies is to achieve a balanced distribution of the workload among numerous

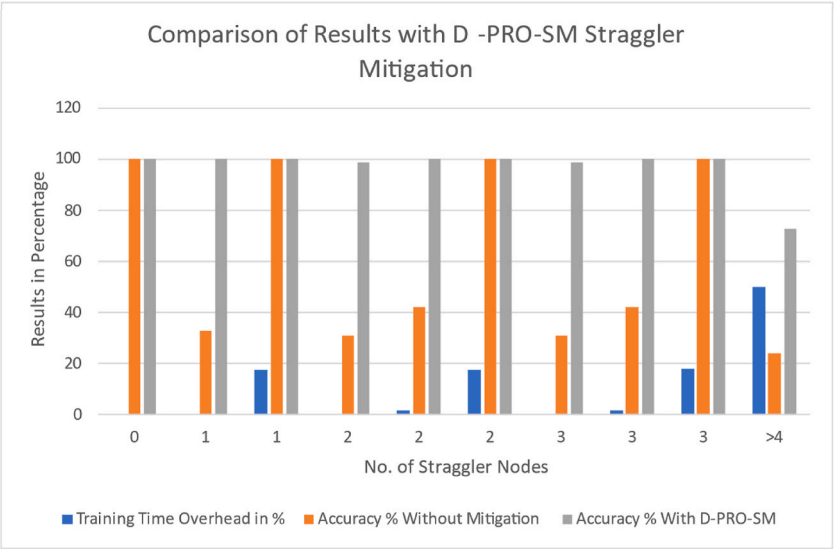


Fig. 4. Mitigation % concerning accuracy and training time overhead.

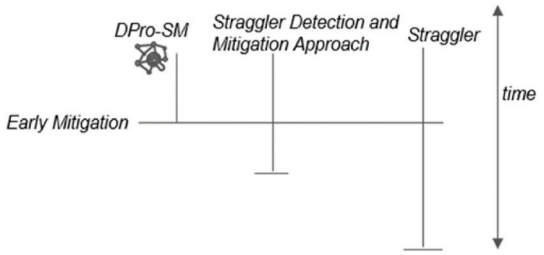


Fig. 5. Timing Analysis of DPro- SM with existing methods.

processing units, such as CPUs or nodes in a cluster, in order to optimize parallelism and reduce periods of inactivity.The utilization of load balancing and partitioning techniques is crucial in effectively handling extensive graphs within distributed and parallel computing settings. Graph-based algorithms and applications benefit from the usage of resources, as it enables them to operate effectively. Additionally, the reduction in processing times contributes to better computational efficiency. Moreover, the scalability and fault tolerance of these algorithms and applications are enhanced, further enhancing their efficiency.The approach of load balancing and partitioning in graphs enhances computing performance by taking into account data dependencies and prioritizing synchronization depending on the significance of the data. The process of reducing the quantity of transferred data and ensuring task balancing among workers is implemented. The approach utilizes the spatial and connectivity characteristics of the simulation environment.

7. Conclusion and future scope

DPro-SM is a promising framework for mitigating the issue of straggler nodes in distributed deep learning. By using LSTM to predict the completion time of each worker, DPro-SM can proactively allocate resources and reduce the overall training time. DPro-SM framework's LSTM model makes use of its capacity to simulate long-term dependencies to forecast worker completion times based on historical data and present system circumstances. With the help of this predictive capabilities, DPro-SM can make well-informed judgments, which optimize resource allocation and greatly enhance the effectiveness and scalability of distributed deep learning applications, eventually resulting in a quicker and more efficient model training process. The Graph load balancing and partitioning method improves computational efficiency by considering data dependencies and prioritizing synchronization based on data importance. While there is still room for improvement, DPro-SM shows excellent potential in addressing the challenges of distributed deep learning and improving the scalability and efficiency of large-scale machine learning tasks. There are several potential avenues for future research and development of DPro-SM. One area of focus could be on improving the accuracy of the LSTM-based straggler prediction model, potentially by incorporating additional features or using more advanced machine learning techniques. Another potential direction is to explore the use of DPro-SM in other types of deep learning tasks beyond image classification, such as natural language processing or speech recognition. Additionally, further investigation could be done on the scalability and performance of

DPro-SM in more extensive and complex distribution systems. Finally, there is also potential for integrating DPro-SM with other existing straggler mitigation methods to further improve the efficiency and effectiveness of distributed deep learning. This methodology enhances the effectiveness of resource allocation and job distribution by the proactive identification of workers who are anticipated to require more time to complete their tasks. The DPro-SM system has the potential to bring novel methodologies for the dynamic allocation of computer resources, using real-time forecasts. The implementation of dynamic allocation plays a crucial role in optimizing the usage of resources and reducing the time required for training. This invention holds great potential for enhancing large-scale distributed deep learning processes. DPro-SM primarily emphasizes the development of scalable solutions for distributed deep learning, hence facilitating the management of bigger datasets and models. The achievement of scalability can be facilitated by the use of effective strategies for partitioning, scheduling, and resource management.

In the future, the paper's suggested research and development directions for the DPro-SM framework cover a number of important fields. These include evaluating DPro-SM's scalability and performance in more complex distribution systems, improving the accuracy of the LSTM-based straggler prediction model through advanced techniques and the incorporation of additional features, expanding the applicability of DPro-SM beyond image classification to domains like speech recognition and natural language processing, and combining DPro-SM with existing straggler mitigation techniques to create a comprehensive approach to efficiency.

Data availability statement

Data associated with this study has been into a publicly available repository https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia.

Additional information

No additional information is available for this paper.

CRediT authorship contribution statement

Aswathy Ravikumar: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Harini Sriraman:** Conceptualization, Data curation, Formal analysis, Methodology, Software, Supervision, Validation, Visualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors appreciate the efforts of the anonymous reviewers toward this publication. The support from Vellore Institute of Technology University, Chennai is also deeply appreciated.

References

[1] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, A.Y. Ng, Large Scale Distributed Deep Networks, (n.d.) 11.
[2] T. Ben-Nun, T. Hoefler, Demystifying Parallel and Distributed Deep Learning: an In-Depth Concurrency Analysis, 2018. http://arxiv.org/abs/1802.09941. (Accessed 18 December 2022).
[3] A. Ravikumar, H. Sriraman, P.M. Sai Saketh, S. Lokesh, A. Karanam, Effect of neural network structure in accelerating performance and accuracy of a convolutional neural network with GPU/TPU for image analytics, PeerJ. Comput. Sci. 8 (2022) e909, https://doi.org/10.7717/peerj-cs.909.
[4] A. Ravikumar, H. Sriraman, A novel mixed precision distributed TPU GAN for accelerated learning curve, Csse 46 (2023) 563–578, https://doi.org/10.32604/csse.2023.034710.
[5] H.K. Omar, A.K. Jumaa, Distributed big data analysis using spark parallel data processing, Bullet. Electrical Eng. Informatics 11 (2022) 1505–1515, https://doi.org/10.11591/eei.v11i3.3187.
[6] A. Ravikumar, H. Sriraman, Real-time pneumonia prediction using pipelined spark and high-performance computing, PeerJ. Comput. Sci. 9 (2023) e1258, https://doi.org/10.7717/peerj-cs.1258.
[7] S. Harini, A. Ravikumar, Effect of parallel workload on dynamic voltage frequency scaling for dark silicon ameliorating, in: 2020 International Conference on Smart Electronics and Communication, ICOSEC), 2020, pp. 1012–1017, https://doi.org/10.1109/ICOSEC49089.2020.9215262.
[8] A. Ravikumar, H. Sriraman, Staleness and stagglers in distibuted deep image analytics, in: 2021 International Conference on Artificial Intelligence and Smart Systems, ICAIS), 2021, pp. 848–852, https://doi.org/10.1109/ICAIS50930.2021.9395782.
[9] A. Ravikumar, Non-relational multi-level caching for mitigation of staleness & stragglers in distributed deep learning, in: Proceedings of the 22nd International Middleware Conference: Doctoral Symposium, ACM, Québec city Canada, 2021, pp. 15–16, https://doi.org/10.1145/3491087.3493678.
[10] S. Harini, A. Ravikumar, D. Garg, VeNNus: an artificial intelligence accelerator based on RISC-V architecture, in: N. Chaki, J. Pejas, N. Devarakonda, R.M. Rao Kovvur (Eds.), Proceedings of International Conference on Computational Intelligence and Data Engineering, Springer, Singapore, 2021, pp. 287–300, https://doi.org/10.1007/978-981-15-8767-2_25.
[11] A. Ravikumar, H. Sriraman, S. Lokesh, P. Maruthi Sai Saketh, Identifying pitfalls and solutions in parallelizing long short-term memory network on graphical processing unit by comparing with tensor processing unit parallelism, in: S. Smys, K.A. Kamel, R. Palanisamy (Eds.), Inventive Computation and Information Technologies, Springer Nature Singapore, Singapore, 2023, pp. 111–125.

[12] K. Chahal, M.S. Grover, K. Dey, A Hitchhiker's Guide on Distributed Training of Deep Neural Networks, 2018, https://doi.org/10.48550/arXiv.1810.11787.
[13] R. Mayer, H.-A. Jacobsen, Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools, 2019. http://arxiv.org/abs/1903.11314. (Accessed 15 July 2022).
[14] S. Mittal, S. Vaishay, A survey of techniques for optimizing deep learning on GPUs, J. Syst. Architect. 99 (2019), 101635, https://doi.org/10.1016/j.sysarc.2019.101635.
[15] D. Moldovan, I. Anghel, T. Cioara, I. Salomie, Time series features extraction versus LSTM for manufacturing processes performance prediction, in: 2019 International Conference on Speech Technology and Human-Computer Dialogue (SpeD), 2019, pp. 1–10, https://doi.org/10.1109/SPED.2019.8906653.
[16] S. Cheng, I.C. Prentice, Y. Huang, Y. Jin, Y.-K. Guo, R. Arcucci, Data-driven surrogate model with latent data assimilation: application to wildfire forecasting, J. Comput. Phys. 464 (2022), 111302, https://doi.org/10.1016/j.jcp.2022.111302.
[17] S. Cheng, J. Chen, C. Anastasiou, P. Angeli, O.K. Matar, Y.-K. Guo, C.C. Pain, R. Arcucci, Generalised latent assimilation in heterogeneous reduced spaces with machine learning surrogate models, J. Sci. Comput. 94 (2022) 11, https://doi.org/10.1007/s10915-022-02059-4.
[18] A. Ravikumar, H. Sriraman, Computationally efficient neural rendering for generator adversarial networks using a multi-GPU cluster in a cloud environment, IEEE Access (2023) 1, https://doi.org/10.1109/ACCESS.2023.3274201, 1.
[19] A. Harlap, H. Cui, W. Dai, J. Wei, G.R. Ganger, P.B. Gibbons, G.A. Gibson, E.P. Xing, Addressing the straggler problem for iterative convergent parallel ML, in: Proceedings of the Seventh ACM Symposium on Cloud Computing, Association for Computing Machinery, New York, NY, USA, 2016, pp. 98–111, https://doi.org/10.1145/2987550.2987554.
[20] E. Ozfatura, S. Ulukus, D. Gündüz, Straggler-aware distributed learning: communication–computation latency trade-off, Entropy 22 (2020), https://doi.org/10.3390/e22050544.
[21] S.K. Hanna, R. Bitar, P. Parag, V. Dasari, S. El Rouayheb, Adaptive distributed stochastic gradient descent for minimizing delay in the presence of stragglers, in: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, Barcelona, Spain, 2020, pp. 4262–4266, https://doi.org/10.1109/ICASSP40776.2020.9053961.
[22] S. Li, S.M.M. Kalan, A.S. Avestimehr, M. Soltanolkotabi, Near-Optimal Straggler Mitigation for Distributed Gradient Methods, 2017. http://arxiv.org/abs/1710.09990. (Accessed 18 June 2023).
[23] G. Xiong, G. Yan, R. Singh, J. Li, Straggler-Resilient Distributed Machine Learning with Dynamic Backup Workers, 2021, https://doi.org/10.48550/arXiv.2102.06280.
[24] X. Ouyang, C. Wang, J. Xu, Mitigating stragglers to avoid QoS violation for time-critical applications through dynamic server blacklisting, Future Generat. Comput. Syst. 101 (2019) 831–842, https://doi.org/10.1016/j.future.2019.07.017.
[25] J. Cipar, Q. Ho, J.K. Kim, S. Lee, G.R. Ganger, G. Gibson, K. Keeton, E. Xing, Solving the Straggler Problem with Bounded Staleness, (n.d.).
[26] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, Improving MapReduce Performance in Heterogeneous Environments, (n.d.).
[27] S.-T. Cheng, C.-W. Hsu, G.-J. Horng, C.-H. Lin, Adaptive cache pre-forwarding policy for distributed deep learning, Comput. Electr. Eng. 82 (2020), 106558, https://doi.org/10.1016/j.compeleceng.2020.106558.
[28] R. Tandon, Q. Lei, A.G. Dimakis, N. Karampatziakis, Gradient coding: avoiding stragglers in distributed learning, in: Proceedings of the 34th International Conference on Machine Learning, PMLR, 2017, pp. 3368–3376, in: https://proceedings.mlr.press/v70/tandon17a.html. (Accessed 18 June 2023).
[29] A. Harlap, H. Cui, W. Dai, J. Wei, G.R. Ganger, P.B. Gibbons, G.A. Gibson, E.P. Xing, Solving the Straggler Problem for Iterative Convergent Parallel ML, (n.d.).
[30] C. Karakus, Y. Sun, S. Diggavi, W. Yin, Straggler Mitigation in Distributed Optimization through Data Encoding, 2018, https://doi.org/10.48550/arXiv.1711.04969.
[31] E. Bin Khunayn, S. Karunasekera, H. Xie, K. Ramamohanarao, Exploiting data dependency to mitigate stragglers in distributed spatial simulation, in: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Association for Computing Machinery, New York, NY, USA, 2017, https://doi.org/10.1145/3139958.3140018.
[32] W. Li, D. Liu, K. Chen, K. Li, H. Qi, Hone, Mitigating stragglers in distributed stream processing with tuple scheduling, IEEE Trans. Parallel Distr. Syst. 32 (2021) 2021–2034, https://doi.org/10.1109/TPDS.2021.3051059.
[33] C. Karakus, Y. Sun, S. Diggavi, W. Yin, Redundancy Techniques for Straggler Mitigation in Distributed Optimization and Learning, 2018, https://doi.org/10.48550/arXiv.1803.05397.
[34] S. Deshmukh, K. Thirupathi Rao, M. Shabaz, Collaborative learning based straggler prevention in large-scale distributed computing framework, Secur. Commun. Network. (2021), e8340925, https://doi.org/10.1155/2021/8340925, 2021.
[35] P. Garraghan, X. Ouyang, R. Yang, D. McKee, J. xu, Straggler Root-Cause and Impact Analysis for Massive-Scale Virtualized Cloud Datacenters, IEEE Transactions on Services Computing, 2016, p. 1, https://doi.org/10.1109/TSC.2016.2611578, 1.
[36] A. Bhandare, J. George, S. Deshpande, Y. Karle, Review and Analysis of Straggler Handling Techniques, vol. 7, 2016.
[37] K.G. Narra, Z. Lin, M. Kiamari, S. Avestimehr, M. Annavaram, Slack squeeze coded computing for adaptive straggler mitigation, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1–16, https://doi.org/10.1145/3295500.3356170.
[38] Q. Zhou, S. Guo, H. Lu, L. Li, M. Guo, Y. Sun, K. Wang, Falcon: addressing stragglers in heterogeneous parameter server via multiple parallelism, IEEE Trans. Comput. 70 (2021) 139–155, https://doi.org/10.1109/TC.2020.2974461.
[39] S. Zhang, A.E. Choromanska, Y. LeCun, Deep learning with elastic averaging SGD, in: Advances in Neural Information Processing Systems, Curran Associates, Inc., 2015, in: https://proceedings.neurips.cc/paper/2015/hash/d18f655c3fce66ca401d5f38b48c89af-Abstract.html. (Accessed 24 June 2022).
[40] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, K. He, Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, 2018. http://arxiv.org/abs/1706.02677. (Accessed 15 July 2022).
[41] J. Chen, X. Pan, R. Monga, S. Bengio, R. Jozefowicz, Revisiting distributed synchronous SGD. https://doi.org/10.48550/arXiv.1604.00981, 2017.
[42] R. Bitar, M. Wootters, S.E. Rouayheb, Stochastic Gradient Coding for Straggler Mitigation in Distributed Learning, 2019. http://arxiv.org/abs/1905.05383. (Accessed 18 June 2023).
[43] Chest X-ray images (pneumonia), (n.d.). https://kaggle.com/paultimothymooney/chest-xray-pneumonia. (Accessed 5 January 2022).
[44] G. Ananthanarayanan, M.C.-C. Hung, X. Ren, I. Stoica, A. Wierman, M. Yu, GRASS: Trimming Stragglers in Approximation Analytics, (n.d.).
[45] G. Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica, Effective Straggler Mitigation: Attack of the Clones, 2013, pp. 185–198. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/ananthanarayanan. (Accessed 18 June 2023).
[46] N.J. Yadwadkar, G. Ananthanarayanan, R. Katz, Wrangler: predictable and faster jobs using fewer resources, in: Proceedings of the ACM Symposium on Cloud Computing, ACM, Seattle WA USA, 2014, pp. 1–14, https://doi.org/10.1145/2670979.2671005.