Hindawi
Security and Communication Networks
Volume 2021, Article ID 8340925, 9 pages
https://doi.org/10.1155/2021/8340925

Research Article

Collaborative Learning Based Straggler Prevention in Large-Scale Distributed Computing Framework

Shyam Deshmukh,[1] **Komati Thirupathi Rao**,[1] **and Mohammad Shabaz**[2]

[1]*Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur 522502, AP, India*
[2]*Arba Minch University, Arba Minch, Ethiopia*

Correspondence should be addressed to Mohammad Shabaz; mohammad.shabaz@amu.edu.et

Received 7 April 2021; Revised 9 May 2021; Accepted 13 May 2021; Published 24 May 2021

Academic Editor: Manjit Kaur

Modern big data applications tend to prefer a cluster computing approach as they are linked to the distributed computing framework that serves users jobs as per demand. It performs rapid processing of tasks by subdividing them into tasks that execute in parallel. Because of the complex environment, hardware and software issues, tasks might run slowly leading to delayed job completion, and such phenomena are also known as stragglers. The performance improvement of distributed computing framework is a bottleneck by straggling nodes due to various factors like shared resources, heavy system load, or hardware issues leading to the prolonged job execution time. Many state-of-the-art approaches use independent models per node and workload. With increased nodes and workloads, the number of models would increase, and even with large numbers of nodes. Not every node would be able to capture the stragglers as there might not be sufficient training data available of straggler patterns, yielding suboptimal straggler prediction. To alleviate such problems, we propose a novel collaborative learning-based approach for straggler prediction, the alternate direction method of multipliers (ADMM), which is resource-efficient and learns how to efficiently deal with mitigating stragglers without moving data to a centralized location. The proposed framework shares information among the various models, allowing us to use larger training data and bring training time down by avoiding data transfer. We rigorously evaluate the proposed method on various datasets with high accuracy results.

1. Introduction

Any organization that depends on a cloud computing environment majorly focuses on factors like CPU usage, memory, I/O and Network for performance optimization. However, all these parameters are susceptible to performance degradation and may result in suboptimal quality of service (QoS). The Google cluster's trace study is a milestone toward the analysis of workloads in a cloud environment with multiple servers as studied in Dean and Ghemawat [1]; Chen et al. [2]; Reiss et al. [3]. This provides the analysis of workload data recorded on Google cluster trace. The important contribution is the analysis of many tasks and jobs that offer an efficient allotment of the resources for new upcoming tasks to the cloud data center, thereby increasing a throughput of the data center. Owing to the inherent nature of a parallel execution in distributed computing systems, sometimes, it experiences the slow-running tasks known as stragglers, potentially resulting in a delayed job execution. Cloud computing and high-performance computing frameworks typically monitor task completion status and launch backup tasks for stragglers during job execution. Such redundant approaches incur huge operational and financial costs. Even with this, they do not provide postevent analyses to diagnose the causes of the stragglers and their proactive prevention. Typical straggler identification is performed in two modes: (1) reactive (online) and (2) proactive (offline). Reactive techniques typically use a criterion of comparing the task execution time with a threshold calculated based on the median value within all the tasks [4].

Monitoring data may not be always accessible from the user side since the monitoring tools are hard to install and tune. Hence, some studies focus on the offline strategy by analyzing logs instead of monitoring Lu et al.[5]. Cluster managers, e.g., YARN in Vavilapalli et al. [6], Isard et al. [7], Verma et al. [8], have different focuses. They provide resource isolation and allocation based on usages, job priorities, and fairness. They do not provide answers to which tasks are stragglers within a job or to why those tasks are slower.

On the other hand, proactive methods analyze dynamic features like resource utilization, node performance, and heterogeneity that change over time. Using ML, it is possible to build models for previously unknown values using training data that can predict the future and identify straggler [9]. Straggler detection and analysis using ML can be categorized under proactive approaches. Javadpour et al. [10] propose a dynamic method that applies neural networks for identifying straggler tasks to increase the efficiency. Another method of straggler-identification compares the task's execution time (or progress) with a threshold calculated based on the median value within all the tasks. Moreover, there are a breed of techniques of straggler identification based on CPU utilization. It has been identified that there is a strong correlation between high system CPU utilization and straggler occurrence as examined in Reiss et al. [3]; Shen and Li [11]. The reason for this occurrence is resource contention. This is further compounded due to Head-of-Line blocking (HOL blocking), task interference during execution, busy locks, queue issues, hazard rates of task execution, and launching additional speculative replicas, which requires additional time for execution.

The state-of-the-art proactive models as studied analyze the workload and compute nodes as a separate straggler estimation task with independent models. One of the motivations for pursuing a separate ML model for each workload and node independently is because there exists a wide variety of resources allocation ranging from node to node and workload to workload. Consequently, a wide variety of straggler patterns arise because of such heterogeneity. This was demonstrated by Yadwadkar et al. [4]. Thus, a separate ML model training deemed to be necessary. However, such models face a couple of major challenges: (1) independent node and workload that need a set of new training leading to increased time for data gathering, and (2) data scarcity that might arise for a given workload for respective node yielding suboptimal ML models. This set of challenges can be effectively addressed by the ML model that learns the straggler prediction task collaboratively. In such approaches, the node, where sufficient training data would not be available, would get the data, while it was executing other workloads, or from other nodes running the same workload. This can be achieved in practice using multitask learning (MTL) as demonstrated by Yadwadkar et al. [9]. Another approach mentioned in Deshmukh et al. [12] tried to avoid straggler occurrence through data parallelism techniques like MPI-libraries.

Developing a distributed machine learning approach, which distributes large scale data efficiently, is challenging. Standard ML techniques need the training data to be gathered at a centralized location, i.e., on one machine or in a data center. Such data collection and analysis might be difficult to conduct in practice because of resource constraints. In a distributed setting, multiple nodes collaboratively work toward a common optimization objective through an interactive process of local computation and communication, which ideally should result in all models converging to a global optimum.

To alleviate problems, in this paper, we propose a Collaborative Learning-based (CL) formulation for learning predictors that are highly accurate and generalize better than multiple independent models. This is based on the alternate direction method of multipliers- (ADMM-) based support vector machine (SVM), proposed by Boyd et al. [13]. The proposed model enables the nodes to collectively learn a shared prediction model while keeping all the training data on nodes, decoupling the ability to do ML from the need to store the data in the centralized manner. CL allows for smarter models, lower latency, and less power consumption, all while ensuring privacy. There exists a subtle difference between parallel variants of traditional ML models and the CL-based ones; traditional ones have single instruction multiple data (SIMD) architecture, while the latter have decentralized/distributed optimization of model parameters. The local models make predictions on the nodes by bringing the model training to the node as well.

In CL, there exist two types of nodes: (1) a common handler that shares the model updates with other nodes, and (2) independent nodes that are the members of the data center. Independent node downloads the current model, improves it by learning from data on node itself, and then performs the model parameter changes as an update. Only this update to the model is sent to the common node, where it is immediately processed with other node updates to improve the shared model. All the training data remains on the node, and no individual updates are stored in the common node. Consequently, no data transfer takes place among the nodes making it highly resource-efficient and quick. In case of straggler identification, each independent node would be trained on the local data, and thus, forming a local model (A) for straggler identification, and the parameters of all such nodes are aggregated (B) to form a consensus change. Note here that all data reside on local nodes, while only ML model parameters are shared. The consensus change form (B) is reflected on the global straggler identification model (C), owing to the decentralization property of collaborative filtering. Finally, a copy of (C) is made available on each of (A) for straggler prediction. To that end, our key contributions are as follows:

(1) A novel CL-based technique of the straggler identification problem that is resource-efficient and captures the heterogeneous resource contention patterns across workloads and nodes.

(2) A rigorous evaluation of the proposed system for predicting and avoiding stragglers in both generated data and real-world production cluster traces.

(3) The robust CL-based formulation for straggler detection even with a small number of stragglers, thus tackling class imbalance problems, a phenomenon that frequently occurs in ML problems because of lack of sufficient training examples.

In what follows, we first give some background on stragglers in Section 2. We then describe the proposed CL-based straggler detection framework in Section 3. In Section 4, we empirically evaluate our formulations on the various workloads. In Section 5, we describe the results substantiating claims proposed in this article. We conclude with an outlook of improvement and discussion of the proposed work.

2. Related Work

Considering the dynamic nature of the cloud environment including nonreliable resources, heterogeneous workload, and quality of service (QoS) requirements, a static resource management solution may not work. Therefore, a static resource manager is extended with a monitoring module, which collects the valuable information on the performance of the application along with the resource utilization of system components about the state of the system. On the other hand, advances in machine-learning-based (ML) methods offer all the behavioral patterns and interesting changes of monitored components. Obtained knowledge about nonconforming patterns, which is often referred to as an outlier induced for a variety of reasons, helps improve the system performance. Parallel computing frameworks that follow the MapReduce by Dean and Ghemawat [1] paradigm are widely used in real-world big data applications to handle batch and streaming data. Among these, Zaharia et al. [14] have recently gained wide adoption. Different from the Hadoop framework as in Manikandan and Ravi [15], Vavilapalli et al.[16], Spark supports a more general programming model, in which an in-memory technique, called Resilient Distributed Dataset (RDD), Zaharia et al. [17], is used to store the input and intermediate data generated during computation stages. Spark is an implementation of the MapReduce model that outperforms Hadoop by packing multiple operations into single tasks, and by utilizing the RAM memory for caching intermediate data. We target Apache Spark, because it is a widely used, efficient, state-of-the-art platform for data analytics, and it is currently the fastest-growing such open-source platform, Zaharia et al. [14].

Apache spark is an open-source cluster computing engine for large data processing. One of the most important factors in processing large datasets is the speed achieved through running computations in memory. At its core, Spark is a 'computational engine' that is responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines, or a computing cluster. Spark is designed to efficiently scale up from one-to-many thousands of compute nodes. To achieve this while maximizing flexibility, Spark can run over a variety of cluster managers, including Hadoop YARN, and a simple cluster manager included in Spark itself called the Standalone Scheduler. The Spark context connects to the Spark cluster manager, which then allocates resources across the worker nodes for the application. The cluster manager allocates executors across the cluster worker nodes. It copies the application jar file to the workers, and finally it allocates tasks.

LATE by Zaharia et al. [18] uses progress score to enhance the performance as compared to speculative execution. But it exerts pressure on other running tasks by competing for the resources and presumes that tasks make development at a roughly constant rate, which is not always the case. Mantri proposed by Ananthanarayanan et al. [19] focuses more on saving computing resources of a cluster, i.e., task slots. If the backup job has an extremely large probability to finish early, Mantri will stop the initial task while the cluster is active (kill-restart method). However, the kill-restart method may not guarantee that the new task will be completed earlier than the original one. In all reactive techniques, the problem gets even worse when some tasks start straggling when they are well into their execution. Cloning mechanism like Dolly proposed by Ananthanarayanan et al. [20] is proactive but focuses only on interactive jobs and is replicative in nature, incurring additional resources.

A detailed survey of load balancing strategies using Hadoop queue scheduling and virtual machine migration was proposed by Dey and Gunasekhar [21]. A method was proposed by Sravanthi and Rao [22], which is a dynamic, processing aware job scheduler, a technique that performs load allotment work to nodes based on their prior performance. Similarly, a method was proposed by Naresh et al. [23] performing optimal resource discovery and dynamic resource allocation. It is based on improved particle swarm optimization and cuckoo search algorithms. Load balancing is the process of adapting to increase and decrease in the workload with associated resource consumption in data centers that enhance the overall performance of the system achieving client satisfaction. An effective measure was studied by Talasila et al. [24] for tackling the load balancing phenomenon for efficient traffic handling in the public cloud. Another method based on Ant colony swarm optimization based on performance analysis of load balancing techniques in cloud centers was studied by Reddy et al. [25], to prevent the latency in real-time stream processing engines like Apache Spark streaming, with an additional technique like dolly retreat mechanism to avoid stragglers and process data efficiently, as studied in Srikanth and Reddy [26]. Radha and Rao [27] offered a comprehensive review of techniques to increase MapReduce performance in heterogeneous cloud environments by partitioning data locality through intermediate data at the reducer side. By applying the delayed scheduling by enhancing the data locality in MapReduce, Radha and Rao [28] have shown the performance improvement in slot Utilization and Hadoop cluster. Praveen et al. [29] proposed an effective resource allocation using a social group optimization algorithm in conjunction with the scheduling of tasks by application of shortest-job-first scheduling technique for minimizing the makespan time and maximizing throughput.

Many researchers have attempted to avoid stragglers through machine learning approaches. The poor performing nodes are identified and blacklisted [30, 31] during the task scheduling phase. These techniques again lead to resource wastage, as they are not able to participate in the execution as stragglers are mainly nonpersistent. Mao et al. [32], Du et al. [33], and Zhang et al. [34] have applied a reinforcement learning approach for mitigating stragglers, which reduce job completion time, but the preciseness of identification of stragglers may not be optimum. Existing approaches used in decentralizing data consists of Alternating Direction Method of Multipliers (ADMM) based algorithms like [35–39].

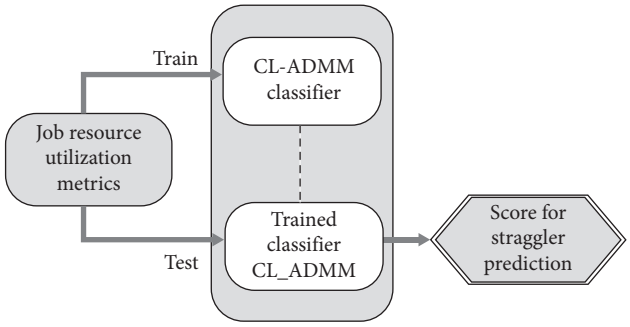

Figure 1: Workflow of proposed straggler detection framework.

3. Proposed Work

3.1. Framework. We introduce a novel framework to identify stragglers, as illustrated in Figure 1 which is based on two main stages. The first stage consists of two parts: (1) extraction of feature vectors from various job resources utilization metrics of nodes; (2) training a global classifier with the help of multiple independent local models as described in the current and next sections as depicted in Figure 2. The second stage consists of testing workloads from the validation or unseen environment by applying the learned model. The feature designing from the test data is the same as mentioned above. Testing at nodes is performed by copying the global model to a node.

The training of the proposed framework takes place in multiple stages as depicted in Figure 2. It shows learning phase of distributed SVM via ADMM, in which individual worker trains SVM model concurrently and separately. In the beginning, each worker's local SVM will be different, but after exchange of model parameters with global model, it becomes more similar in each iteration. The global model will aggregate the local model parameters and generate the consensus model.

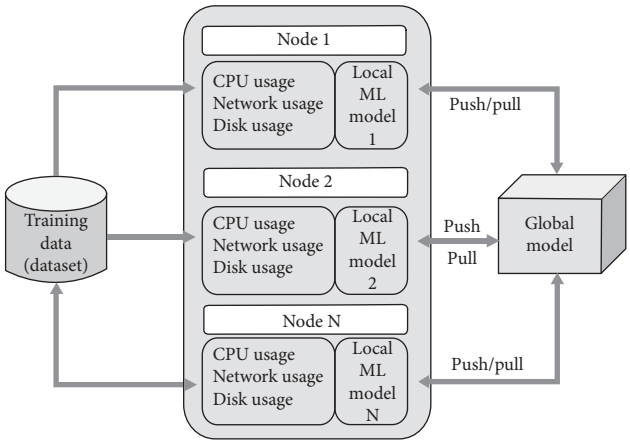

Figure 2: Training phase of the proposed architecture.

3.2. ADMM-Based Collaborative Learning. We consider a set of n nodes and a central aggregator. Each node $i \in n$ has an independent training dataset. $D_i := (a_{i,j}, b_{i,j})$: $\forall j \in m_i$ where m_i is the number of training samples in the dataset D_i, $a_{i,j} \in R^d$ is the d – dimensional feature vector of the j-th training sample, and $b_{i,j} \in R^p$ is the corresponding p-dimensional data label. In this paper, we consider a star network topology, where each node can communicate with the central aggregator, and the aggregator is responsible for message passing and aggregation. The goal of straggler identification is to train a supervised learning model on the segregated dataset $D_i, i \in n$ from n nodes. This enables predicting a label for any new data feature vector of job utilization metrics. The learning objective can be formulated as the following regularized empirical risk minimization problem:

$$\min_{w} \sum_{i=1}^{n} \sum_{j=1}^{m_i} \frac{1}{m_i} l\left(a_{i,j}, b_{i,j}, w\right) + \lambda R(w), \tag{1}$$

$w \in R^{d \times p}$ is the trained global ML model. $l(\cdot)$: $R^d \times R^p \times R^{d \times p} \longrightarrow R$ is the loss function used to measure the quality of the trained model, $R(\cdot)$ refers to the regularizer function introduced to prevent overfitting, and $\lambda > 0$ is the regularizer parameter controlling the impact of regularization. Casting Equation (1) can be cast into the loss function of binary logistic regression classifier as follows:

$$l\left(a_{i,j}, b_{i,j}, w\right) = \ln\left(1 + \exp\left(-b_{i,j} w^T a_{i,j}\right)\right). \tag{2}$$

To apply ADMM, we reformulate Equation (1) as

$$\min_{\{w_i\}_{i \in n}} \sum_{i=1}^{n} \left(\sum_{j=1}^{m_i} \frac{1}{m_i} l\left(a_{i,j}, b_{i,j}, w_i\right) + \frac{\lambda}{n} R(w_i) \right), \tag{3}$$

such that $w_i = w, \quad i = 1, \ldots, n.$

In standard ADMM, the augmented Lagrangian function associated with the problem (3) is

$$L_\rho(w, \{w_i\}_{i\in n}, \{\gamma_i\}_{i\in n}) = \sum_{i=1}^{n} L_{\rho,i}(w_i, w, \gamma_i), \quad (4)$$

where

$$L_{\rho,i}(w_i, w, \gamma_i) = \sum_{j=1}^{m_i} \frac{1}{m_i} l\left(a_{i,j}, b_{i,j}, w_i\right) + \frac{\lambda}{n} R(w_i) - \gamma_i, w_i - w + \frac{\rho}{2}\|w_i - w\|^2, \quad (5)$$

$\{\gamma_i\}_{i\in n} \in R^{d\times p\times n}$ are the dual variables associated with the constraints, and $\rho > 0$ is the penalty parameter. The standard ADMM solves the problem in Equation (3) in a Gauss-Seidel manner by minimizing Equation (4) with respect to $\{w_i\}_{i\in n}$ and **w** alternatively followed by a dual update of $\{\gamma_i\}_{i\in n}$. The formulation is based on the work presented in [13].

3.3. Straggler Prediction Model Using Probabilistic Classification. The training dataset $D = \{(a_i, b_i)|a_i \in R^d, b_i \in (-1, +1)\}_{i=1}^{m}$ is either −1 or +1, indicating the class to which data point ai belongs. The objective of probabilistic classification using logistic regression as mentioned above is to learn the class-posterior probability $p(\mathbf{b} \mid \mathbf{a})$ of the training samples dataset D. Based on the class-posterior probability, classification of a new sample a_{test} can be carried out $b_{\text{test}} := \max \in \{-1, +1\} p(b|a)$ with confidence $p(b|a)$. Let $b \in \{-1, \overset{b}{+}1\}$ represent the nonstraggler and straggler class, respectively. The task of straggler detection is to assign the value of the estimate $\widehat{p}(a)$ for test data, given training data and model. The conditional probability of straggler is given by $\widehat{p}(a, \theta)$, where θ is the vector of parameters learned in Section 3.*2 - w, w_i, ρ and γ* respectively.

4. Experimental Study

4.1. Configurations. The various configuration parameters are mentioned in Table 1.

4.2. Cluster Setup. We have a network of nodes in a Hadoop Cluster as per the configurations as shown in Table 1. We have built the Hadoop Cluster of five nodes to estimate the proposed solution for discovering straggler nodes. One of the nodes is picked as a master node, and it runs the Hadoop Distributed File System (Name-node) and MapReduce run time (Resource manager). The remaining four nodes are slave nodes (Data-nodes and Node-managers). The regular block size in Hadoop is 128 MB. When a larger file is inserted into HDFS, it will be broken down into 128 MB pieces and divided between data nodes. All systems in the multinode setup use Ubuntu v16.04 operating system, JDK 1.7, and Hadoop 2.7.1 version for performance.

4.3. Workload. We executed two different types of jobs on intensive Hadoop memory and intensive CPU utilization. Memory intensive tasks such as machine learning-based K-Nearest neighbors and image-processing were performed. CPU intensive tasks were created by kernel Support vector machines and similar algorithms. Some network intensive tasks using heavy uploads and downloads were also created in conjunction with the first two types of load creation mechanism.

4.4. Dataset

4.4.1. Features. We have used 22 features, most of them related to CPU utilization (e.g., CPU idle time, user time, system, CPU wait, I/O and CPU speed, etc.), disk utilization (e.g., amount of free space, local read/write statistics from the data nodes, maximum percent used for all partitions, etc.), memory utilization (e.g., Amount of buffered, cached, shared, free and total amount of available memory, etc.), network utilization (e.g., packets in and out per second, etc.), and system-level features (e.g., total number of processes, total number of running processes, total amount of swap memory, amount of available swap memory, etc.). The job history server traces job execution time through start time, finish time, task execution time, read data in bytes, write data in bytes, and elapsed time that are also obtained. We have not used any feature reduction technique as the number of features is already lower in number, and performance demonstrated using the proposed method in section 5 does not seem to be affected by the number of features.

4.4.2. Dataset Generation. For constructing the prediction models, we require a labelled dataset consisting of {feature, label} pairs. We have used Ganglia-based node-monitor by Massie et al. [40] to capture resource utilization metrics of nodes. We get the features related to jobs from Hadoop. We select a subset consisting of five features, that is, execution time, average CPU utilization ratio, memory usage, disk I/O time, and cycles per instruction, empirically using the proposed ADMM. The metric used for deciding the straggler is normalized duration (execution) time suggested by Yadwadkar et al. [4]:

$$n\,d(t) = \left\lceil \frac{\text{task execution time}}{(\text{amount of work bytes read/written by task } t)} \right\rceil. \quad (6)$$

An i^{th} task t of job J is called a straggler if $n\,d(t_i) > (\beta \times \text{median}\{n\,d(t_i)\})$, where β is the threshold

Table 1: Hardware and software configurations used.

Attributes	Values
Hadoop cluster installation mode	Fully distributed
Number of cluster nodes	5
RAM at nodes 1, 2, 3, and 4	4 GB
Network topology	Star with master-slave
Hard disk space	500 GB
Master node	Has a job follower
Slave node	Data node and task follower
File block size	128 MB
Clock frequency	2.7 GHz

coefficient, taken as 1.3, as a rule of thumb. However, we see the variation of performance metrics across various values of β.

4.5. Experimental Setup. With labelling the dataset, we evaluate the performance of straggler prediction on all the workloads using ADMM-based SVM. First, each node builds its local classification model by collecting data on that node. To get the features related to the straggler node, we have overloaded each node alternatively and then captured its features. This process of capturing the dataset for straggler and nonstraggler in the training phase requires little time, and we incrementally increased the number of stragglers in the system. The standard feature normalized data is fed to the ADMM SVM written in the Spark environment by Dhar et al. [41]. This reduces the model building time with a small amount of model parameter transfer. This completes the model training phase. This global model would reside on each node for the classification.

In this experiment, we have chosen a binary classification method, where +1 is the label for straggler and -1 for nonstraggler. For ADMM-SVM for logistic regression, logistic loss is used. The practical implementation of ADMM-LR is referred to in [34]. For ADMM-SVM with least square formulation, the loss function is least square for both methods, and the regularization parameter is elastic net. The parameters λ and ρ are set to 1. For MPI logistic regression from Scikit-learn by Pedregosa et al. [42], we use *L2* penalty, with regularization constant C being set to 1.

We consider a 5-fold cross validation method to determine the performance metrics. Here, we provide the results of ADMM-SVM with logistic regression and least-squares SVM and centralized parallel (message passing interface) SVM (LIBLINEAR) from Pedregosa et al. [42] and Fan et al. [43] and then evaluate them using the following scenarios:

(1) Classification accuracy when there is sufficient data

(2) Classification accuracy when sufficient data is not available We also provide the performance across various β. Overall, we have 724 stragglers and 21000 nonstraggler records.

5. Results and Discussions

5.1. Performance Evaluation Metrics. We use Precision, Recall, and $F1$-Score (denoted as $F1$) to evaluate the performance of all models: the true positives (TP) are the true straggler detected by the system. False positives (FP) are the nonstraggler data points detected as stragglers. True negatives (TN) are the correct nonstragglers detected by the system, and false negatives (FN) are stragglers detected as nonstragglers by the system. With this set of definitions,

$$\begin{aligned} \text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\ \text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\ F1 \text{ score} &= \frac{2 \times \text{Precision Recall}}{\text{Precision} + \text{Recall}}. \end{aligned} \tag{7}$$

5.2. Evaluation. We report the quantitative improvement for identification of stragglers: Figure 3 presents the F1 score (harmonic mean of precision and recall) of straggler detection averaged across the 5-fold with 80/20 ratio of train and test. All data points on the plot are a 5-fold quantity average. Figure 3 reports values of F1 score for various values of β. From the figures, our approaches outperform the MPI-based methods. We have an extremely high F1-score of more than 98% for beta values 1.6 to 1.8. The benchmark method has a lower performance. A potential reason for MPI-based SVM to perform slightly worse is because it is not easily scalable. Besides, the class imbalance between stragglers and nonstragglers is problematic for most supervised learning methods. Our framework alleviates these problems by including a training dataset estimating correct data distributions of each class. Figure 4 represents the classification accuracy when sufficient data is not available. It represents the accuracy of 5-fold straggler detection average with 80/20 ratio of train and test. All data points on the plot are a 5-fold quantity average. With the increase in the number of straggler class examples available for training, the straggler detection improves. Again, ADMM-LR-SVM performs best, while its variant ADMM-LS-SVM is not far from it. With just 183 sets of straggler examples, our framework achieves more than 94% accuracy. The performance of both of these methods remains constant with increased inclusion of straggler records.

MPI based SVM performs relatively poorly because of class imbalance examples. Similarly, Figure 5 represents the F1 score computed against various numbers of straggler records. With increasing the number of straggler examples, F1-score of straggler detection improves. Again, ADMM-LR-SVM performs best, while its variant ADMM-LS-SVM is not far from it. With just 183 sets of straggler examples, our framework achieves F1-score more than 98%. As seen in

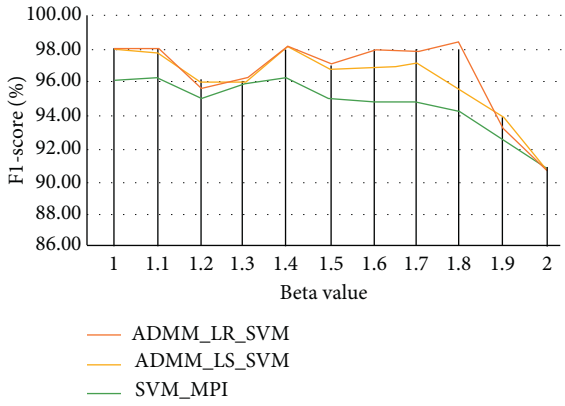

Figure 3: The F1-score variation across various values of β.

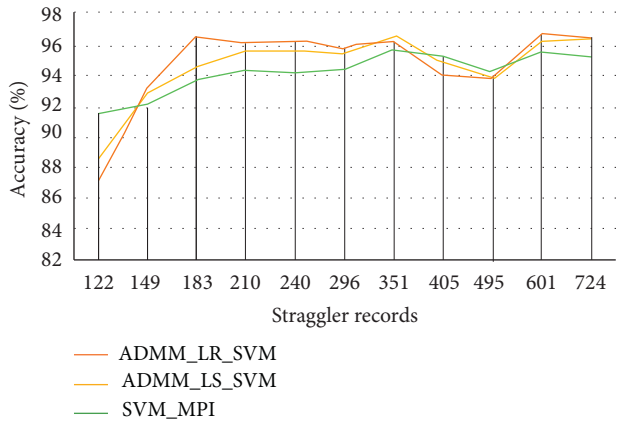

Figure 4: Variation of accuracy with increasing number of stragglers.

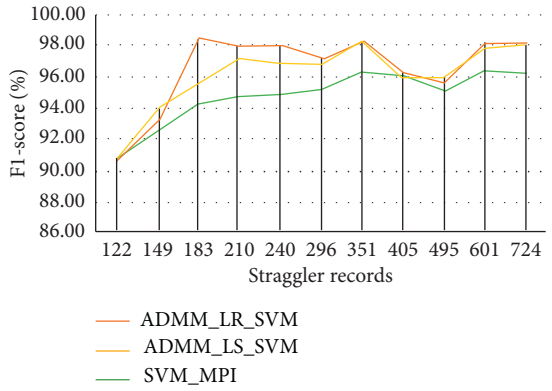

Figure 5: Variation of F1-score with increasing number of stragglers.

Figures 3–5, there is considerable improvement of lesion detection, thanks to the proposed framework.

6. Conclusion

We have introduced a novel method for straggler detection based on support vector machine variants of alternating directions of method of multipliers. The efficacy of our method was evaluated through rigorous evaluation on straggle data. We have demonstrated that our method achieves better performance compared to the benchmark method: MPI-based SVM. Our formulation can achieve better accuracy with only a third of the training data and can generalize better than other approaches for learning tasks with little or no data. Thus, the class imbalance problem is tackled naturally. Our methodology is more suitable for straggler analysis because of its ability to capture heterogeneous distribution of stragglers correctly. This performance suggests that it can provide valuable assistance in detecting the stragglers in production with high reliability. The proposed framework is generic in nature and can be extended to various types of workloads, e.g., workloads across various data centers, independent of big data computing frameworks. The framework described here allows for exploration of additional information with node and job utilization resources. For example, one can consider infusing distribution of node utilization metrics with task utilization metrics and thus can help in further management of scheduling of jobs. The adaptation of ADMM-SVM investigated for learning a comprehensive predictor with better accuracy and reduced job completion along with improved data privacy as no data movement from client site is required for sensitive applications.

Data Availability

The data are available upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] Q. Chen, C. Liu, and Z. Xiao, "Improving mapreduce performance using smart speculative execution strategy," *Institute of Electrical and Electronics Engineers Transactions on Computers*, vol. 63, no. 4, pp. 954–967, 2013.

[3] C. Reiss, J. Wilkes, and J. L. Hellerstein, *Google Cluster-Usage Traces: Format Schema*, Google Inc, White Paper, , pp. 1–14, 2011.

[4] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz, "Wrangler: Predictable and faster jobs using fewer resources." in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 1–14, Seattle WA USA, November 2014.

[5] S. Lu, X. Wei, B. Rao et al., "LADRA: log-based abnormal task detection and root-cause analysis in big data processing with Spark," *Future Generation Computer Systems*, vol. 95, pp. 392–403, 2019.

[6] V. K. Vavilapalli, A. C. Murthy, C. Douglas et al., "Apache hadoop yarn: yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, pp. 1–16, Santa Clara, CA, USA, October 2013.

[7] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pp. 261–276, Big Sky, MT, USA, October 2009.

[8] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with borg," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, April 2015.

[9] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, and R. Katz, "Multi-task learning for straggler avoiding predictive job scheduling," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 3692–3728, 2016.

[10] A. Javadpour, G. Wang, S. Rezaei, and K. C. Li, "Detecting straggler mapreduce tasks in big data processing infrastructure by neural network," *The Journal of Supercomputing*, vol. 2020, 25 pages, 2020.

[11] H. Shen and Li C. Zeno, "A straggler diagnosis system for distributed computing using machine learning," in *Proceedings of the International Conference on High Performance Computing*, pp. 144–162, Springer, Pune, India, December 2020.

[12] S. Deshmukh, J. Aghav, K. T. Rao, and B. T. Rao, "Avoiding slow running nodes in distributed systems, Lecture Notes in Networks and Systems," in *Proceedings of the Computer Communication, Networking and Internet Security*, Springer, Berlin, Germany, pp. 411–420, 2017.

[13] S. Boyd, N. Parikh, and E. Chu, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*, Now Publishers Inc., Delft, Netherlands, 2011.

[14] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," *Osdi*, vol. 8, no. 7, 2016.

[15] S. G. Manikandan and S. Ravi, "Big data analysis using Apache hadoop," in *Proceedings of the 2014 International Conference on IT Convergence and Security (ICITCS)*, pp. 1–4, IEEE, Beijing, China, October 2014.

[16] V. K. Vavilapalli, A. C. Murthy, C. Douglas et al., "Apache hadoop yarn: yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud*, Santa Clara, CA, USA, October 2013.

[17] M. Zaharia, R. S. Xin, P. Wendell et al., "Apache spark: a unified engine for big data processing," *Communications of the ACM"*, vol. 59, no. 11, pp. 56–65, 2012.

[18] M. Zaharia, M. Chowdhury, T. Das, and A. Dave, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation R12*, pp. 15–28, Renton, WA, USA, April 2008.

[19] G. Ananthanarayanan, S. Kandula, A. G. Greenberg et al., "Reining in the outliers in map-reduce clusters using mantri," *Osdi*, vol. 10, p. 24, 2010.

[20] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: attack of the clones," in *Proceedings of the Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation*, vol. 13, pp. 185–198, Berkeley, CA, USA, April 2013.

[21] N. S. Dey and T. Gunasekhar, "A comprehensive survey of load balancing strategies using hadoop queue scheduling and virtual machine migration," *Institute of Electrical and Electronics Engineers Access*, vol. 7, pp. 92259–92284, 2019.

[22] S. Sravanthi and K. Rao, "Efficient big data analytics with optimized parallel processing," in *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, vol. 11, pp. 312–318, Phoenix, AZ, USA, May 2016.

[23] T. Naresh, A. Lakshmi, and V. Reddy, "An efficient resource allocation strategy based on improved particle swarm optimization (ipso)," *Pakistan Journal of Biotechnology*, vol. 14, pp. 125–128, 2017.

[24] S. Talasila, V. Havisha, S. Koushik, M. Deep, and V. Reddy, "Load balancing techniques for efficient traffic management in cloud environment," *Inter- National Journal of Electrical and Computer Engineering (IJECE)*, vol. 6, p. 963, 2016.

[25] V. Reddy, K. Surya, M. Praveen, B. Lokesh, A. Vishal, and K. Akhil, "Performance analysis of load balancing algorithms in cloud computing environment," *Indian Journal of Science and Technology*, vol. 9, 2016.

[26] B. V. S. Srikanth and V. Krishna Reddy, "Efficiency of stream processing engines for processing BIGDATA streams," *Indian Journal of Science and Technology*, vol. 9, no. 14, 2016.

[27] K. Radha and D. B. Rao, "A review on enhancing map reduce performance with data locality in heterogeneous environment," *International Journal of Control Theory and Applications*, vol. 9, pp. 8463–8472, 2016.

[28] K. Radha and B. T. Rao, "Slot utilization and performance improvement in hadoop cluster," *Advances in Intelligent Systems and Computing*, vol. 72, pp. 49–62, 2016.

[29] s Praveen, T. R. Komati, and B. Janakiramaiah, "Effective allocation of re- sources and task scheduling in cloud environment using social group optimization," *Arabian Journal for Science and Engineering*, vol. 43, 2017.

[30] N. J. Yadwadkar and W. Choi, *Proactive Straggler Avoidance Using Machine Learning*, Univ. California, Berkeley, CA, USA, White Paper, 2012.

[31] X. Ouyang, C. Wang, and J. Xu, "Mitigating stragglers to avoid QoS violation for time-critical applications through dynamic server blacklisting," *Future Generation Computer Systems*, vol. 101, Article ID 831842, 2019.

[32] H. Mao, M. Schwarzkopf, S. Bojja Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," 2018, http://arxiv.org/abs/1810.01963.

[33] P. Lubell-Doughtie and J. Sondag, "Practical distributed classification using the alternating direction method of multipliers algorithm," in *Proceedings of the 2013 IEEE International Conference on Big Data*, pp. 773–776, Silicon Valley, CA, USA, October 2013.

[34] H. Du, S. Zhang, P. Han, K. Zhang, and B. Xu, "Cheetah: A dynamic performance optimization approach on heterogeneous big data analytics cluster," in *Proceedings of the 5th International Conference on Big Data Computing and Communications (BIGCOM)*, pp. 169–177, QingDao, China, August 2019.

[35] H. Du and S. Zhang, "Hawkeye: adaptive straggler identification on heterogeneous spark cluster with reinforcement learning," *Institute of Electrical and Electronics Engineers Access*, vol. 8, pp. 57822–57832, 2020.

[36] E. Wei and A. Ozdaglar, "Distributed alternating direction method of multipliers," in *Proceedings of the 2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 5445–5450, Wailea, HI, USA, December 2012.

[37] J. Bhola, S. Soni, and G. K. Cheema, "Genetic algorithm based optimized leach protocol for energy efficient wireless sensor networks," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 3, pp. 1281–1288, 2019.

[38] Q. Ling and A. Ribeiro, "Decentralized linearized alternating direction method of multipliers," in *Proceedings of the Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 5447–5451, IEEE, Florence, Italy, May 2014.

[39] Q. Ling, Y. Liu, W. Shi, and Z. Tian, "Weighted admm for fast decentralized network optimization," *Institute of Electrical and Electronics Engineers Transactions on Signal Processing*, vol. 64, no. 22, pp. 5930–5942, 2016.

[40] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[41] S. Dhar, C. Yi, N. Ramakrishnan, and M. Shah, "Admm based scalable machine learning on spark," in *Proceedings of the 2015 IEEE International Conference on Big Data (Big Data)*, pp. 1174–1182, IEEE, Santa Clara, CA, USA, November 2015.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort et al., "Scikit- learn: machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[43] R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin, "Liblinear: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.