IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 22, 2021, accepted April 5, 2021, date of publication April 12, 2021, date of current version April 20, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3072609

Distributed Tree-Based Machine Learning for Short-Term Load Forecasting With Apache Spark

AMEEMA ZAINAB[1,2], (Member, IEEE), ALI GHRAYEB[2], (Fellow, IEEE), HAITHAM ABU-RUB[2], (Fellow, IEEE), SHADY S. REFAAT[2], (Senior Member, IEEE), AND OTHMANE BOUHALI[3,4], (Member, IEEE)

[1]Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843, USA
[2]Department of Electrical and Computer Engineering, Texas A&M University at Qatar, Doha 5825, Qatar
[3]Research Computing, Texas A&M University at Qatar, Doha 5825, Qatar
[4]Qatar Computing Research Institute, Hamad Bin Khalifa University, Doha 5825, Qatar

Corresponding author: Ameema Zainab (azain@tamu.edu)

This work was supported in part by the National Priorities Research Program (NPRP) Grant from the Qatar National Research Fund (a member of Qatar Foundation) under Grant NPRP10-0101-170082, in part by the Internal Seed Grant from Texas A&M University at Qatar, in part by IBERDROLA Qatar Science and Technology Park (QSTP) LLC, and in part by the Open Access Funding by the Qatar National Library.

ABSTRACT Machine learning algorithms have been intensively applied to perform load forecasting to obtain better accuracies as compared to traditional statistical methods. However, with the huge increase in data size, sophisticated models have to be created which require big data platforms. Optimal and effective use of the available computational resources can be attained by maximizing the effective utilization of the cluster nodes. Parallel computing is demanded to allow for optimal resource utilization in dealing with smart grid big data. In this paper, a master-slave parallel computing paradigm is utilized and experimented with for load forecasting in a multi-AMI environment. The paper proposes a concurrent job scheduling algorithm in a multi-energy data source environment using Apache Spark. An efficient resource utilization strategy is proposed for submitting multiple Spark jobs to reduce job completion time. The optimal value of clustering is used in this paper to cluster the data into groups to be able to reduce the computational time additionally. Multiple tree-based machine learning algorithms are tested with parallel computation to evaluate the performance with tunable parameters on a real-world dataset. One thousand distribution transformers' real data from Spain for three years are used to demonstrate the performance of the proposed methodology with a trade-off between accuracy and processing time.

INDEX TERMS Apache spark, concurrent computing, load forecasting, parallel processing, resource management.

I. INTRODUCTION

With the development of the smart infrastructure in the electrical grids, the data collected from various units and locations over time have begun to receive the attention of grid operators and research centers. Data centers usually collect 15-minutes to one-hour frequency of grid data, which creates enormous amounts of data streams. The power grid operators are looking forward to creating data analytics solutions to benefit from these enormous amounts of collected data. Processing large amounts of data and deriving insights will help in the purpose of knowledge discovery and better decision making. Machine learning (ML) techniques help in the decision-making processes and big data provides power in better decision making. For an hourly load forecast, if the algorithm takes few hours to calibrate the hour-ahead forecasting model, then it is a big data problem [1]. The computing time can not be longer than the lead time for any business application.

To manage big data and perform ML with big data, most of the researchers focused on one of the important challenges, i.e., handling large size of data stored historically in the data centers [2]. Several ML algorithms were designed with the assumption that the entire dataset fits in the memory. This assumption negatively affects the ML algorithms while impeding their performance. For instance, a support vector machine (SVM) has a space complexity of $O\left(m^2\right)$ and a training complexity of $O\left(m^3\right)$ [3], where m indicates the number

The associate editor coordinating the review of this manuscript and approving it for publication was Lin Zhang.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

of samples. Traditional ML algorithms were built with the assumption that the data can fit into memory, but with the era of big data, it becomes challenging for ML models to adapt to the deluge of data. Big data due to its nature of velocity also imposes the challenge of all the data being available during training. Therefore, as the size of the data increases, distributed processing frameworks, parallel data structures, data reuse, and data partitioning become important characteristics. Resilient distributed datasets (RDDs) implemented in a Spark cluster computing framework exhibit in-memory characteristics [4]. This leads to the use of the typical architecture which can accommodate both the cluster computing framework and machine learning capabilities. To improve the performance of the big data machine learning algorithms, manipulations in terms of the way ML algorithms execute and the processing infrastructure is necessary. Among the various ML paradigms in big data, this paper focuses on tree-based methods and ensemble learning techniques. Splitting a deluge of data into multiple datasets to perform training with the ML models has gained significant improvement in the learning process in terms of the big data context. For example, the authors in [5] applied ensemble learning to subsamples of big data improving learning accuracy and simultaneously decreased the computation time.

The multi AMI infrastructure mostly concentrates on forecasting the load of all the distribution transformers (DT's) at the same time. In this paper, a novel scheduling technique with the help of the Apache spark platform is proposed to short-long term forecast the load of all the one thousand transformers simultaneously. The spark cluster submits big data analytics tasks as spark jobs and the computational resources are allocated optimally to these spark jobs. The amount allocated to these jobs is customizable by the user which affects the Job Completion Time (JCT) significantly. This paper utilizes ML algorithms such as Spark Random Forest and Spark Gradient boosted regression trees for training and forecasting the load. The proposed method performs load forecasting by submitting multiple jobs concurrently on the data sets utilizing the cluster resources optimally.

The main contributions of this paper can be summarized as follows:

1) Proposing an optimal scheduling algorithm to perform load forecasting with parallel and distributed execution in a multi-AMI environment on the smart grid big data.
2) Tuning the ML models to gain high accuracy along with measures to combat overfitting.
3) Testing the proposed methodology on all the one thousand transformers' data without grouping, then a comparison is made against the proposed grouping technique to show its merits.

The performance of the proposed method is tested on real big data from an industry partner Iberdrola to validate its effect on ascended performance.

The paper is organized as follows. Section II discusses the related research in the field of load forecasting using big data platforms. Section III describes the novel scheduling algorithm proposed to perform load forecasting on multiple datasets utilizing apache spark. Section IV describes the experimental setup of the optimal scheduling algorithm along with the results obtained by implementing the proposed methodology on real big data. Section V concludes the paper.

II. RELATED WORK

Many papers have proposed benchmarking results with the use of ML for load forecasting, but in this section, the essence of big data smart grid load forecasting using spark is outlined. The widely installed smart meters collect huge amounts of load data for each of the grid's distribution transformers. Many computing frameworks [6]–[9] have been developed for the analysis of big data but, MapReduce [6] is the most famous one because of its features of fault-tolerance, parallel computation, and flexibility. Apache Spark [10] proposed by the Zaharia *et al.* team emerged to overcome the drawbacks in MapReduce. It is an open-source framework and is 100 times faster than Hadoop MapReduce [11]. Spark can execute over several cluster managers such as Hadoop YARN [12], Apache Mesos [13], and spark's standalone scheduler. Spark can also interface with a variety of data storage repositories like Hadoop Distributed File System (HDFS) [14], Hive [15], Hbase [16], to name a few. However, spark supports distributed computing resulting in a communication overhead increase. Previous research has observed that by only increasing the computational capability, JCT reduces but then starts increasing in communication overhead [17]. Hence the proposed scheduling algorithm in this paper focuses to utilize the available computation capability and still be able to submit multiple jobs with the help of an optimal scheduling algorithm and not losing on communication overhead.

Highly cited algorithms for forecasting smart grid data include linear regression, SVM and its variants [18], and artificial neural networks (ANN) [19] [20]. A pooling-based deep recurrent neural network (DRNN) was proposed to learn the spatial information, which outperformed Support Vector regressor (SVR), Auto-Regressive Integrated Moving Average (ARIMA), and the classical deep recurrent neural network (RNN) [21]. In [22], Happy et al, proposed a statistical approach for load forecasting using quantile regression random forest, risk assessment index, and probability map. In [23], a backpropagation approach was utilized to perform short-term load forecasting utilizing weather data. In [24], Wei et al performed midterm load forecasting of power supply unit (PSU) considered as a collection of distribution transformers. The authors have utilized a dynamic based network (DBN), with a peak load of all the distribution transformers within a PSU summed. All of the summed load values are utilized to train and forecast the load using sparks standalone cluster. However, the use of complete data to train instead of summed load values can result in better training accuracies but will require optimized scheduling method which is achieved in this paper.

This paper focuses on an hourly day-ahead load forecast with the use of spark ML tree-based algorithms. The models are trained with the spark.ml Application Programming Interface (API) of spark which is data frame based facilitating ML pipelines and feasible feature transformations [25].

III. PROPOSED LOAD FORECASTING METHODOLOGY FOR OPTIMIZED COMPUTATION WITH APACHE SPARK

The Spark ML library supports tree-based models namely spark ml decision trees and ensemble models namely spark ml random forest and spark ml gradient boosted regression trees [26]. Spark session connects to the master node to submit jobs, where each job is split into stages, and stages are further split into tasks. Adding more tasks to a single job if possible is recommended as compared to starting new jobs to avoid start-up costs. In the case of data from multiple transformers, each dataset can be assigned as a job. To reduce the execution time of the load forecasting models, multiple DTs' load forecasting is performed simultaneously with the help of parallel job submission in spark. Moreover, the shortest job submitted may consume fewer resources as compared to the other jobs submitted. To overcome this, python's thread pool concurrency feature in addition to the spark fair scheduler can be used. A solution is to decompose the complete dataset into a cluster of transformers IDs and use multiple computing nodes to train the clustered model with an added sequential step to test the model of each of the transformers within the clusters. However, it is necessary to train clustered models first and then test the individual models within the clusters. This attempts to add multiple layers of parallel processes executed sequentially as iterated in Fig. 1. For n clusters, the number of iterations to train the clustered data is n/j, where j is the number of jobs submitted simultaneously. As the n clusters are accessed repeatedly by the processes, the training data pertaining to n clusters is cached into memory. Similarly for t transformers belonging to a cluster, the number of iterations in total to test the holdout data of each of the transformers (TF) is $n * (t/j)$. As the value of $(n * (t/j))$ is larger than n/j in all cases of t, the time in the previous case (with n clusters) is much less than without clustering, provided the data size for each of the jobs in both cases is the same. The proposed parallel and sequential approach of the tree-based ensemble model is deployed on the spark. Fig. 2 is an illustration of the employed master-slave parallel computing paradigm where a single master and multiple slaves are used. To incorporate the proposed methodology parallelism in datastore and training are discussed further.

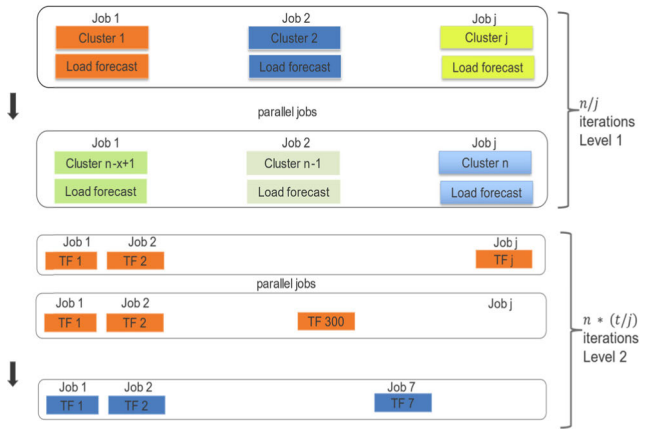

FIGURE 1. Nested parallelism with spark (sequential and parallel runs).

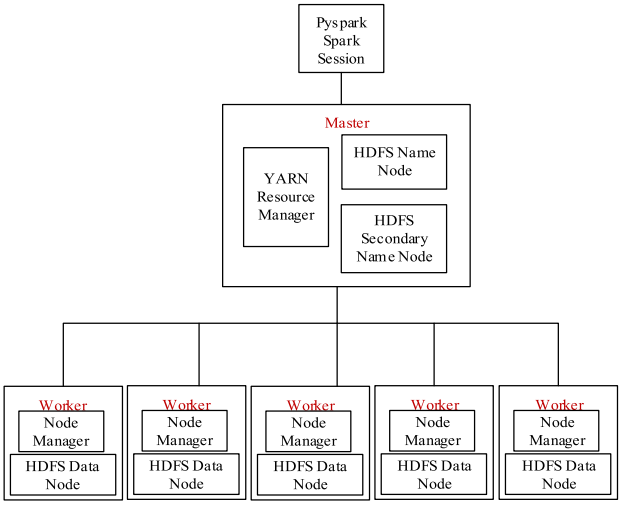

FIGURE 2. Experimental setup of spark framework for load forecasting.

A. DATASTORE PARALLELISM

The big data of the transformers load values with the timestamp is stored in the HDFS with replication factor 3. The resulting load data partitions are constructed into RDDs and stored in the corresponding data nodes. The number of partitions is automatically set by spark as one partition for a block of file, however, the data is repartitioned to 20 which is equal to the number of cores in each of the nodes using the pyspark programming interface. Spark by default supports Kyro serializer, that is almost 10x faster than the default java serializer. Kyro serializer is a graph serialization framework which is efficient and fast, and performs direct copy from object to object rather than a transition phase of bytes in the middle.

B. TRAINING PARALLELISM

The data from HDFS is read into the spark data frame for the analysis. By using the data frame API only, all the physical execution is compiled in native spark using Java Virtual Machine (JVM), while only the logical plan is constructed in pyspark [27]. The use of data frame API in pyspark results in efficient execution as it avoids the creation of key-value pairs that occur in scala. Data frames in spark are immutable like RDD and are conceptually similar to a pandas data frame or a relational database. However, the important difference is the execution of transformations and actions in spark. The spark's

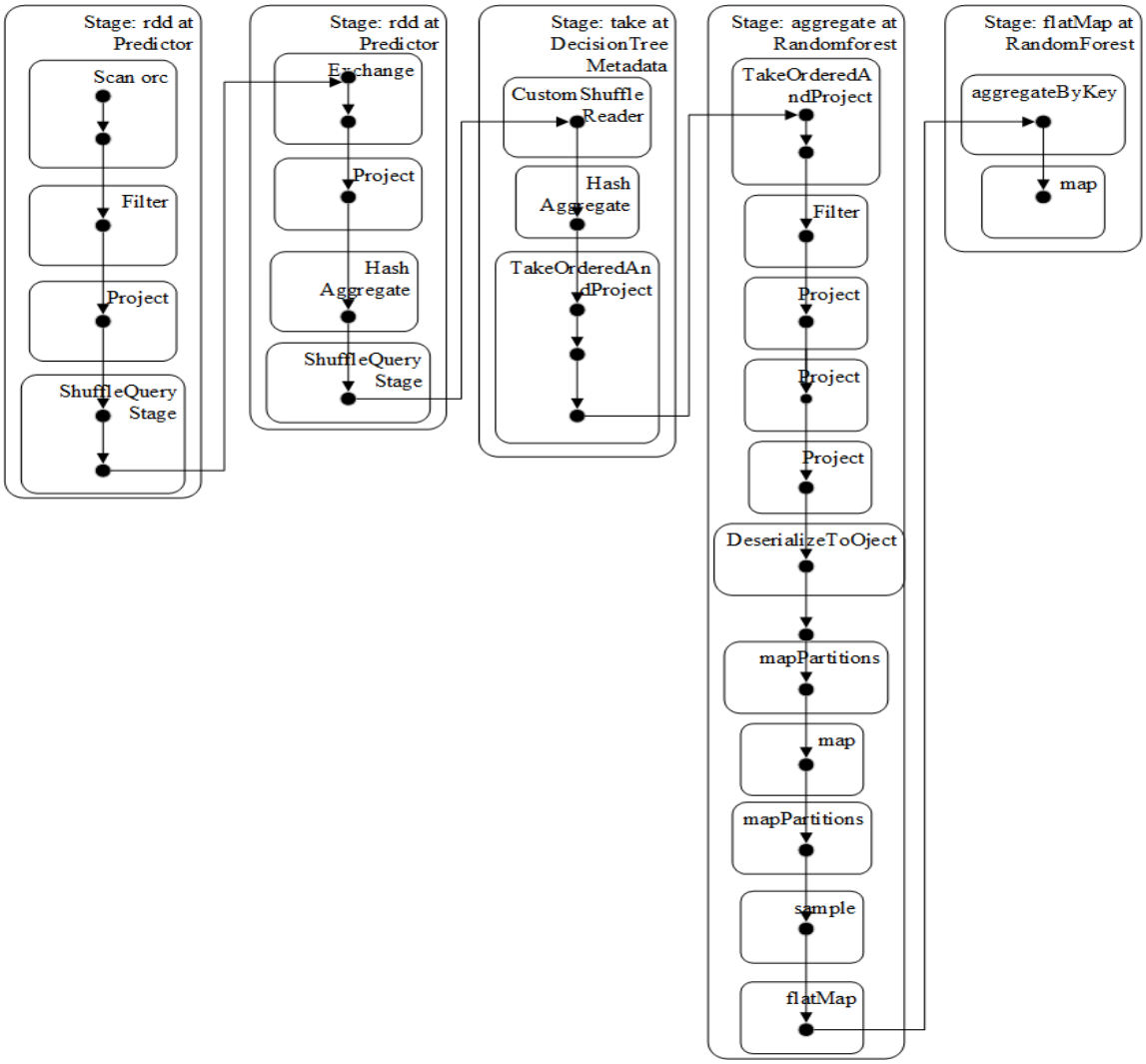

FIGURE 3. Spark-based DAG visualization for random forest regressor.

catalyst optimizer creates an optimized logical plan before sending an instruction to the spark driver. As the catalyst optimizer functions are the same across all the language APIs, data frames provide equivalent performance to all the spark API. Once a logical plan is created, it visualizes it as a Directed Acyclic Graph (DAG) as shown in Fig. 3, and is distributed among all the tasks in a job to be able to perform each of the stages concurrently.

Considering the merits of spark, it is used as the big data processing platform in our application for two of the main computing tasks:

1) Average load matrices calculation: the elements of the average load matrix consist of load averaged for 1 lag day, 7 lag days, etc. The data is inputted into the matrix calculation from the historical data stored in HDFS and the computations are carried out in pyspark.
2) Simultaneous training of DTs' load forecasting models with the help of thread pools in python and multiple jobs in spark utilizing a fair scheduler.

IV. OPTIMAL SCHEDULING ALGORITHM

Scheduling jobs considering the available resources is challenging. An optimal scheduling algorithm is necessary to schedule jobs to be able to reduce the execution time. As per the requirement of load prediction of multiple transformers at the same time, two scheduling algorithms are leveraged in this paper. In this section, the solutions of optimal scheduling when communication costs are ignored and when considered are discussed.

A. IGNORING COMMUNICATION COSTS

Considering w workers available and M jobs to be executed, three cases can be obtained where $(w < M)\,\&(wx = M)$, $(w < M)\,\&(wx < M)$, and $w \geq M$ where x is a multiple of M resulting in $wx = M$. The algorithm in this section is structured as follows.

Step 1: Submit the array of tasks to the w workers
Step 2: w jobs are submitted to the available w workers

Step 3: Whenever a processor becomes available, assign it the unexecuted ready job with the highest priority.

Submitting the jobs with the help of a thread pool as discussed in section III, w jobs are submitted at the same time. Considering the three cases, the algorithm flow can be elaborated for three of the cases as below:

Case I :

$$pool_{\omega}\,(train, [1, 2, \ldots .w])$$
$$(w < M)\,\&(wx = M)$$
$$T_m^1 T_m^2 T_m^3 T_m^4 .. T_m^w$$
$$.$$
$$xtimes$$
$$.$$
$$T_m^1 T_m^2 T_m^3 T_m^4 .. T_M^w$$

Case II:

$$pool_{\omega}\,(train, [1, 2, \ldots .w])$$
$$(w < M)\,\&(wx < M)$$
$$T_m^1 T_m^2 T_m^3 T_m^4 .. T_m^w$$
$$.$$
$$x - 1 times$$
$$.$$
$$T_m^1 T_m^2 T_m^3 .. T_m^{M-(x-1)w}$$

Case III:

$$pool_{\omega}\,(train, [1, 2, \ldots .w])$$
$$(w \geq M)$$
$$T_m^1 T_m^2 T_m^3 T_m^4 .. T_M^w$$

where T_m is the time taken for individual job execution and is assumed to be the same. In case III, the computational capabilities are not as high most of the time when the number of jobs to be submitted is in the thousands. The total execution time in all three cases can be summarized in (1) as:

$$T_{totaltime} = \begin{cases} xT, & where\ (w < M)\ and\ (wx = M) \\ xT, & where\ (w < M)\ and\ \&(wx < M) \\ T, & where\ w \geq M \end{cases} \quad (1)$$

Because of the way the number of concurrent jobs is submitted, w workers are assigned for each step of parallel runs. Although at the last step of execution $wx < M$ still takes the same amount of time, as w workers are assigned to perform the job. Algorithm 1 details the overall proposed load forecasting methodology based on the optimal scheduling algorithm discussed in section III.

Algorithm 1 Proposed Optimal Scheduling Algorithm

Input:
j : the number of batches
T : an array consisting of each of the *meterID*'s
w : number of workers (indicates the number of cores in an executor)
D : data filtered as per *meterID*
tfs: transformer allocated to a cluster with *clusterID*
csv: an empty csv file to accumulate all the results

Initialize:
def *Cluster*(*clusterID*):
$D^{clusterID} = clusterArray\,[clusterID]$;
Cache $D^{clusterID}$ into memory for repeated access;
Create the train and holdout data from $D^{clusterID}$;
Perform ML modeling on grouped train data by performing hyperparameter tuning;
Chose the hyperparameters with least error and store the model $M^{clusterID}$;
end def
def *forecast*(*n*, D^t):
Create the train and holdout data from D^t;
Chose the hyperparameters with least error and store the model M;
Perform testing on holdout data with $M^{clusterID}$;
Use model M to predict the holdout data;
Test the accuracy of the predicted model;
Read the results in the *csv* file;
Update *csv* file with training accuracy and holdout dataset accuracy along with the *meterID*;
end def
Output:
csv: the accuracy of holdout data of all the T models.

1. groupBy D with *meterID* and *timestamp*
2. Perform clustering on optimal value of k to obtain the group of clusters as *clusterlist*
3. Split D into an array of dataframes based on the *clusterlist* as *clusterArray* $[D^1, D^2, \ldots .D^n]$ where n is the number of clusters
4. call pool.map function with *cluster* function and *clusterID* as varibles;
   The function *cluster* is called n times in j batches resulting in n/j iterations. If any processor is available the available job is assigned to the processor. The results are updated simultaneously and any point is equal to the number of forecast processes completed execution;
5. Close pool;
6. Call join function after all the n/j iterations are completed;
7. Open a csv file to store results;
8. **for** n:
9. $tfs = clusterlist\,[n]$;
10. Split D^n into an array of dataframes based on the *tfs* as *tfArray* $[D^1, D^2, \ldots .D^t]$ where t is the number of transformers belonging to cluster n.
11. call pool.map function for *forecast* function with n and D^t as varibles;
    The function *forecast* is called t times in x batches. If any processor is available the available job is assigned to the processor. The results are updated simultaneously and any point is equal to the

number of forecast processes completed execution;
12. close pool;
13. call join function after all the t/j iterations are completed;
14. **end for**
15. **return** *csv*

B. CONSIDERING COMMUNICATION COSTS

The main idea of this scheduling task is to augment the scheduling with new precedence relations to be able to compensate for the communication time. By clustering the jobs into C clusters and submitting them to the same worker, the overall communication between clusters will be minimized. If $\tilde{T}$ is the time taken by a cluster including the communication costs, and y is a multiple of the total number of clusters resulting in $wy = C$, $y\tilde{T}$, is the time taken for all the jobs where $y < x$.

C. OBJECTIVE FUNCTION

This section attempts to create the theoretical functions for parallel and sequential training approaches and to propose an implementation solution based on the spark platform. The collected transformers power data is denoted as D where D^1, $D^2, D^3, \ldots D^M$ denote the data for meter m. The data D^m consists of F features namely month, day, year, etc. Therefore, the chunk of data for a meter ID can be expressed by D^m as following:

$$D^m = [X_1^m, X_2^m, \ldots..X_F^m] \tag{2}$$

$$D = \bigcup_{m=1}^{M} D^m = \bigcup_{m=1}^{M} \bigcup_{n=1}^{N^m} D_n^m \tag{3}$$

where X_f^m is the feature f of the chunk of the data for a meter ID m; N^m is the size of the m^{th} dataset. This chunk of data is trainable input to the machine learning model. Additionally, based on the data decomposition shown in (2), the mean square error (MSE) for regression of the parallel training of the ML model is represented as

$$\begin{aligned} RMSE^{OOB} &= \min \frac{1}{\mathrm{N}} \sum_{m=1}^{M} J^m \\ &= \min \frac{1}{\mathrm{N}} \sum_{m=1}^{M} \sum_{n=1}^{N^m} J_n^m \end{aligned} \tag{4}$$

And, the loss function J_n^m of the sample n in data with a subset, m is given by (5)

$$J_n^m = \sqrt{\frac{1}{N \sum_{n=1}^{N} \left\| y_n^m - \hat{y}_n^{OOB}\left(X_n^m\right) \right\|^2}} \tag{5}$$

where J^m in (6) is the loss function of the m^{th} data set

$$J^m = \sum_{n=1}^{N^m} J_n^m \tag{6}$$

y_n^m and $\hat{y}_n^m$ are the observed and the predicted load values, respectively, of sample n in data subset m; and N is the dimension of each of the output samples. The ML model training is performed to minimize the $RMSE^{OOB}$ in (4) and obtain the trees using the dataset D. Similar procedures are performed for the subset dataset D^m concerning the data subset m for transformer level load forecasting.

V. CASE STUDY

Firstly, the experimental setup is introduced in this section. Secondly, the performance of the proposed scheduling algorithms is evaluated. Finally, the results are presented and discussed.

A. EXPERIMENTAL SETUP

1) CLUSTER CONFIGURATION

The apache-spark platform, where all the computations are performed, consists of one master node and 5 slave nodes as shown in Fig. 2. Each of the 5 compute nodes is Linux-based and contains 24 physical CPU cores – 2 processor sockets with 12 cores per socket – and 128GB of RAM. The interconnect is comprised of the Cray Aries network, which is employed both for MPI as well as for storage traffic [28]. Hadoop 2.8.0 and spark 3.0.0 are installed on both the master and slave nodes. The load forecasting algorithm is implemented in Python 3.6.4.

2) DATA COLLECTION, STORAGE, & PREPROCESSING

In used experiments, the dataset consists of load value and timestamp of 1000 transformers meters of the Iberdrola network [29]. The data is split into 90% (Jan 2017 to Jun 2019) of training and 10% (July 2019 to September 2019) holdout dataset. The total dataset counts to around ∼24000000. The data was collected from the utility company in an Optimized Row Columnar (ORC) format and was stored in the HDFS storage on 5 data nodes and replicated 3 times. Currently, spark supports timestamp input with the help of flint time series as flint context and not flint session. As a consequence of this limitation, the timestamp is split into the year, month, day, and hour.

Fig. 4 shows the power consumption pattern for all the 3 years in the top left, data with large load values on the top right, and the frequency of the load values in the bottom left and bottom right graphs. It can be noted that the bottom left graph is right-skewed, and after log normalization, the spread of the data is more diverse comparatively but still not normally distributed. The bottom right graph also has log+1 normalization as the data consists of load values of 0. It can be noticed that the data is right-skewed.

In the short-term load forecasting (STLF) scenario of this work, the load value of 1000 DTs needs to be forecasted at the same time. The data profile of each of the DT ranges from January 2017 to September 2019. Based on the above information, an ideal load forecasting model for STLF requires

1) The time series of the historical data for the load profile.
2) The parameters of the trained ML model to accelerate the load forecasting.
3) The trained model of a single DT consisting of simple parameters, yet accurate.
4) Model executed and realized efficiently on a parallel processing platform i.e., apache-spark.

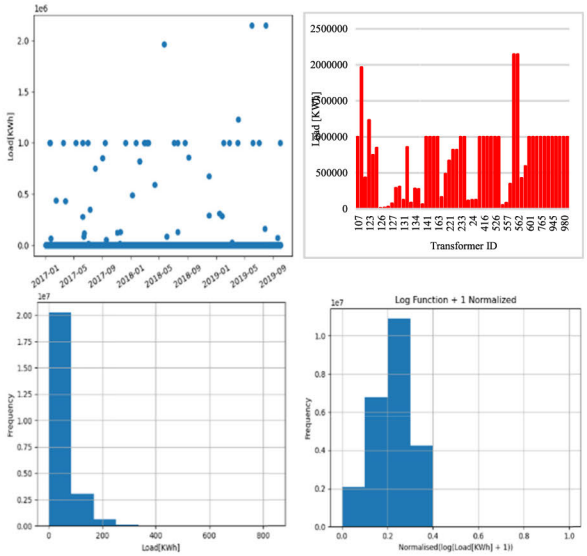

FIGURE 4. Top left - Load distribution across all the three years (The vertical axis indicates the load value in kWh and the x axis indicates the time stamp). Top right – Data with large load values greater than 1000 kWh (The vertical axis indicates the transformer id the data belongs to and the x axis indicates the load value in kWh. Bottom left – Frequency of the load distribution limiting to 1000 kWh. Bottom right – Frequency of log normalized load plus 1.

Spark tree models support both continuous and categorical features, partitioning data by rows, and distributed training. Algorithms available in spark ml are used for performance comparison, which includes the spark decision trees (Spark DT) and tree ensembles i.e., spark parallelized random forests (Spark RF), and spark gradient boosted trees (Spark GBTs).

B. PERFORMANCE EVALUATION

1) AVERAGE RMSE

The objective of future load consumption is to predict the load with high precision and speed to have near real-time processing ability. Root mean square error (RMSE) is used as the error metric because of its wide use. To evaluate the predictive performance, the training dataset is separated from the holdout dataset (data never used for training). All the models are built on the training data and optimized to obtain as low $RMSE_{train}$ as possible and predicted on the holdout dataset to note the $RMSE_{holdout}$. Moreover, to evaluate the performance on all the holdout datasets for different transformers, the average $RMSE$ ($ARMSE$) is calculated as described below:

$$ARMSE = \frac{1}{M}\sum_{i=1}^{M} RMSE_{holdout}\, 1 < i < M \qquad (7)$$

The ARMSE shows how well the ML model learns the data for all the distribution transformers. The reason for choosing ARMSE to have high average accuracy across all the distribution transformers and not just one or a few.

2) EXECUTION TIME

An important objective of choosing the proposed methodology is to reduce the processing time of the transformer's data. To improve the performance in terms of execution time, total time $T_{totaltime}$ is first measured by submitting individual jobs and $\tilde{T}_{totaltime}$ by considering a cluster of jobs. The time is compared in both cases to choose the methodology with the lowest execution time and still retain the skewed distribution of the multiple meters data.

3) SPARK OPTIMIZATION

Besides spark being an in-memory computing framework, it runs on top of the Java Virtual machines (JVMs). Hence tuning the JVM parameters is necessary to improve the performance of spark. In this paper, the authors have identified three key spark parameters that impact the utilization of resources to reduce the workload execution time. The paper has also focused on the right choice of parameters that impact the memory serialization, data compression, caching, and repartitioning of data. Compressing serialized RDDs helps in saving substantial space at the expense of some extra CPU time. Compression of RDD in shuffle operations has a great advantage due to it random read/write and multiple times read/write. Compression of spark RDD is achieved with the help of codec. Experiments are conducted considering: i) various combinations of several executors, ii) the number of cores per executor, and iii) the amount of memory for each of the executors. If CO is the total number of cores in the configuration then

$$CO = E * COperE \qquad (8)$$

where E is the total number of executors assigned and $COperE$ is the number of cores assigned per executor in the spark configuration. The distribution of total memory in the spark configuration is given as follows.

$$MEM = (0.9 * MEMperE * E) + (0.1 * MEMperE * E) \qquad (9)$$

where $MEMperE$ is the memory assigned per executor. The second term in (9) is the overhead memory allocated to each of the executors which accounts for virtual machine overheads or other native overheads. The additional memory and is usually chosen as either 10% of the executor memory or a minimum of 384MB by the spark cluster computing system [30]. Further, the $MEMperE$ is divided into two fractions, one for memory and the other for storage. The memory fraction handles the data structures, out-of-memory error and the storage fraction handles the cached blocks of data. The values of CO and MEM can vary and are very specific to the cluster used for configuring spark. Choosing a larger value of E results in reducing the $COperE$ to balance the CO. Similarly, choosing a larger value of E reduces the $MEMperE$ to balance the MEM.

C. RESULTS AND DISCUSSION

In this section, the metrics discussed in the previous section are evaluated on the dataset to showcase the benefits of the optimal scheduling algorithm. The $ARMSE$ and the execution

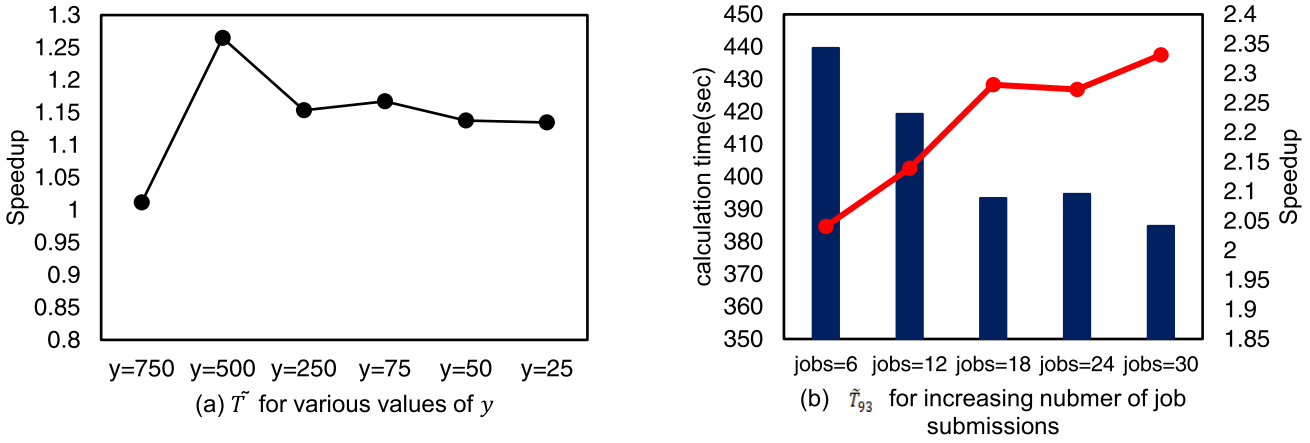

FIGURE 5. Performance evaluation. (a) shows the speedup for various cluster sizes for a concurrent job submission size of 18 and (b) presents the speedup of increasing the number of jobs. A value of $y = 93$ is chosen for all the job submission values.

times are noted under experiments to determine the robustness of load forecasting methodology using spark.

1) VALIDATION OF EXECUTION TIME

In this experiment, the proposed optimal scheduling method is validated in terms of the training time and the forecasting time. The total time T and $\tilde{T}$ are measured for both cases of x and y number of jobs submitted. The proposed scheme is tested for x using the K-means clustering algorithm to group the data in order to obtain clusters with higher accuracy. To validate the proposed method, various chunks of y values are considered and compared against the time taken for x number of chunks of data. For the given data as the value of x is 1000, values ranging from 750 to 25 are chosen as shown in Fig. 5(a). The speedup is calculated for the various combinations by performing $T/\tilde{T}$. As the value of y increases, the size of the data is distributed among the y chunks which also affects the processing time for different sizes of y. For varying values of y the speedup is increasing, indicating the time $\tilde{T}$ is always less than T for all the values of y. Choosing a lesser value of y and still not losing on speedup is recommended, as in practice it will help in reducing the execution time in cases of performing representative clustering. Hence the proposed optimal scheduling algorithm stated in section IV improves the performance by reducing the time to perform the analysis. similarly, for a y value of 93, varying values of the thread pool are performed to analyze the speedup as shown in Fig. 5(b). The choice value of $y = 93$ is based on the kmedoids clustering approach proposed in [31]. Hence the value of y is chosen as 93 to group data closest to each other, rather than choosing a random value of y. Compared with a single job submission, the calculation time tends to decrease gradually as the number of concurrent jobs submitted increases based on the left axis in Fig. 5(b). Massive jobs are distributed across the slave nodes, which reduces the computational load. The spark computing platform captures the intermediates results to memory resulting in the inefficiency of iterative processing where each data frame is called

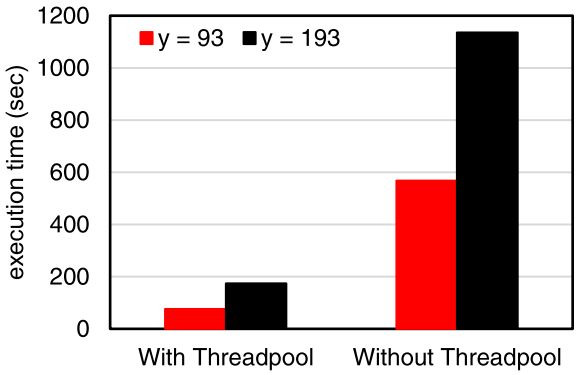

FIGURE 6. Performance benefit in terms of execution time with scheduling that can be achieved by using thread pool.

multiple times for various processing stages. As shown in Fig. 5 (b), based on the right y-axis, the speedup is approximately increasing linearly up to a value close to a number of cores and makes it less linear after a value of 18-20. When the number of jobs submitted increases above the threshold of a possible number of concurrent threads that can be submitted, the data transfer among the processes increases communication overhead which eventually increases the parallel management overhead. Hence a trend of less linearity can be observed clearly after a value of jobs $= 18$. Fig. 6, shows the benefits in terms of scheduling that can be achieved by using thread pool. The experiments are performed y $= 93$ and a randomly chosen group value of y $= 193$ to compare the performance. The execution time is lesser in both the cases as proposed for a value $y < x$. The training time for $y\tilde{T}$, for $y = 93$ executes 2.26 times, and for $Y = 193$, 2 times faster for a pool value of 18, as compared to without threadpool.

Additionally, the clustering time, training time, and testing time for a value of k $= 93$ is estimated based on the clustering performed to choose the best cluster number value as shown in Fig. 7(a). The total execution time (includes the training time of grouped clusters, testing time of individual

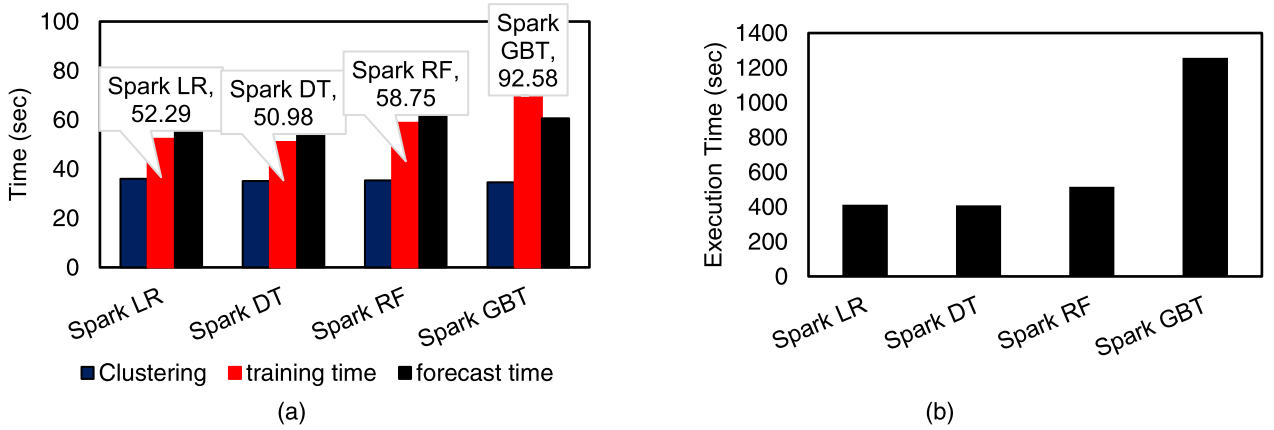

FIGURE 7. Comparison of compute time at various stages of load forecasting. (a) Results obtained for the time taken to perform clustering, training time and testing time on the holdout dataset for SLR(spark LR), SDT, SRF and SGBT. (b) The execution time involves clustering, training of grouped data, testing on clustered data, training on individual transformers and testing on individual transformers for the spark ML models.

transformers with clustering, the training time of the individual transformers, and the testing time for individual transformers) for the 1000 models is shown in Fig. 7(b). The time taken by the gradient boosted algorithm is the highest compared to the other spark ml algorithms. Although both random forest and gradient boosted trees are ensemble models, the random forest takes noticeably lesser time as compared to random forest. Inference out of this observation is that gradient boosted is a boosting algorithm that is quite sequential and is intended to take more execution time whereas multiple trees in the random forest can be run parallelly across the nodes to speed up the execution. The times observed in Fig. 7(a) show the lowest training time for spark decision tree regressor. It can be noted that the time taken to perform testing is almost close to the training time. This is evident from the proposed methodology which states that performing analysis on grouped data is preferred over individual transformers data. However, as the testing has to be performed on all the DT's datasets, grouping cannot be performed to reduce execution time.

To compare the results with different previous works, the comparison has been done with datasets of similar sizes and computational capacities utilized have been documented. The comparison has been done with methodologies that have utilized distributed ML modeling with Apache Spark. The comparison with previous works is presented in Table 1. It can be observed that although the proposed methodology consists of a dataset size of ∼24 million records is superior in terms of execution time as compared to previous works in performing distributed machine learning on big data.

2) VALIDATION OF SPARK OPTIMIZATION

To validate the use of an optimal number of *COperE*, experiments are conducted based on various combinations of *COperE* and *E* which in turn affects the *MEMperE*. Fig. 8 displays the comparison of run-time for various combinations of executors and cores per executor. The combination

TABLE 1. Comparison of performance of ML model in terms of execution time with previous works.

#	Reference	Techniques applied	Number of records	Total Memory	Total number of cores	Execution time (s)
1	[32]	Spark MLib	$2*10^4$	256GB	64	~500
2	[33]	DCPELM (Spark)	$5*10^5$	-		2.50
3	[5]	Spark - RSEM	$10*10^6$	16GB	2	1374
4	[34]	Spark - RF	$24*10^6$	150GB	32	~1600
5	Proposed	Spark - RF	$24*10^6$	600GB	100	58.75

RSEM - Reservoir Sampling based Ensemble Method
DCPELM - Distributed Computing Process Extreme Learning Machine

with the largest number of cores per executor shows the lowest run time as per the secondary y-axis in Fig. 8. As the job submission computes multiple jobs at the same time more number of workers helps in the distribution of the jobs to more number of workers. However, choosing more *E* and less *COperE* is not expected to be efficient as the work will be distributed across more executors resulting in larger transfer of data across the executors. The choice of executors less than 5 is not possible as the number of nodes in the configuration is 5, each node consists of a total of 120GB memory. Reducing the number of executors will result in each executor to contain more than 120GB which exceeds the threshold and is practically not possible. Hence a choice of 5 executors and 20 cores per executor is decided as an optimized combination of the spark configuration. It is worth to mention also that as the number of executors are increased, the *MEMperE* is reduced as it is distributed among the executors, to sum up to *MEM*.

Other than time, communication overhead and data transfer is also a concern in distributed computing. Other than time, communication overhead and data transfer is also a concern in distributed computing. By increasing the depth of

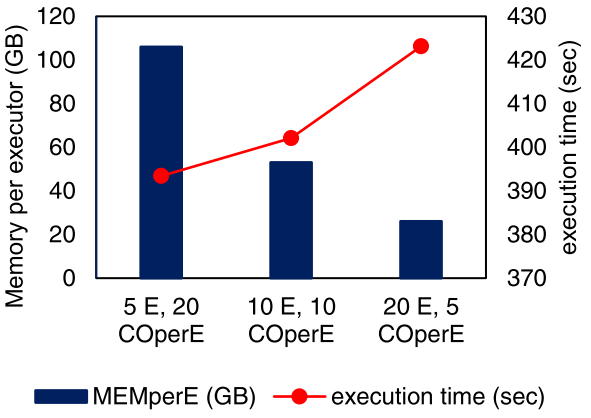

FIGURE 8. Run time comparison for various spark optimization parameters.

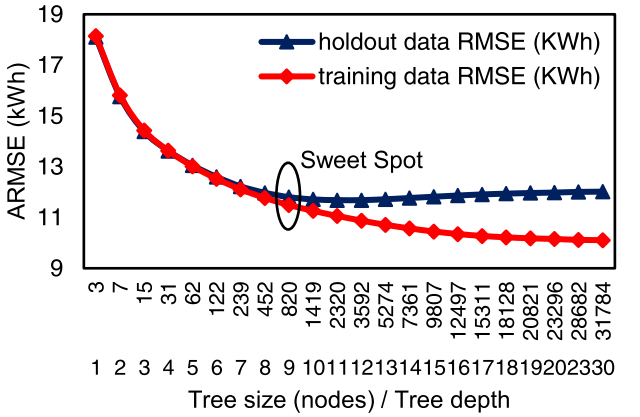

FIGURE 9. ARMSE of training and holdout dataset for spark decision tree. The spot above 820 nodes result in overfitting of the datasets.

TABLE 2. Performance of ML model in terms of RMSE and training time to monitor the effect of deep networks.

Max depth	Training accuracy (kWh)	Holdout accuracy (kWh)	Time (sec)
2	5.850740	9.205257	28.711398
4	5.422266	9.738556	25.833542
6	4.759868	11.30912	24.382604
8	4.401291	17.37286	25.479490
10	4.153886	12.44175	26.977820
12	4.083622	12.50476	29.376672
14	4.077183	17.77995	32.170327
16	4.081471	17.79192	35.396624

a random forest regressor (refer to Table 2), it is noticed that as each of the tree grows larger, after a max depth of 10, large task transfer warning is shown by spark indicating that deep models with large number of tree nodes are being transferred across the tasks which results in more data transfer. Referring to Table 2, it can be noted that by increasing the depth of the model, the training accuracy is reducing and the time taken is close to each other. However, after a max depth of 10, there is a jump in the time and the time is gradually increasing. This indicates that more amount of time is being utilized in transferring data, hence such a scenario has to be avoided during the execution or the spark parameters have to be tuned further to accommodate large task binaries.

3) OVERFITTING

Most of the machine learning models perform accurately post tuning of hyperparameters. However, excess tuning of parameters tends to fit the training data so accurately that the model is overfitted. Once overfit, the models do not perform as expected on the new forecasting dataset. To avoid such a case many measures are taken to avoid overfitting in the training data. Consideration of holdout data set which has never been used in the training is one of the measures to prevent overfitting. In the case of tree ML models, the depth of the tree or the number of nodes while training can be regulated. An experiment is performed by increasing the depth of the tree and the num of nodes in the trained model is monitored (refer to Fig. 9). The x-axis shows the number of nodes and the y-axis the performance measure in terms of RMSE. It can be noted that the red line in Fig. 9, which indicates the training RMSE, is decreasing with an increase in the number of nodes by fitting the dataset onto the trees as deeper as possible. Whereas, the blue line which denotes the holdout RMSE does not show such a trend. After a point, the models' performance starts deteriorating. This point is called the sweet spot where the models tend to start overfitting. Hence the depth of the model has to be restricted below this point which indicates a value of 820 nodes for a tree depth of 9 in Fig. 9. By taking such a measure on the average RMSE of all the transformers data, a max depth of 8 is chosen to perform training on the spark ML models.

4) VALIDATION OF ACCURACY

This section discusses the results after considering the validation of accuracy, computation time, and overfitting in the previous sections. For each of the spark ml models, the performance of the training dataset and holdout dataset are compared. Table 3 reports the *ARMSE* computed for all the 1000 datasets. Holdout *ARMSE* indicates the quality of load forecasting. Thus, a lower value of *ARMSE* indicates a better load forecasting model. The table indicates that the values of holdout *ARMSE* are the lowest for the spark random forest regression model. Referring back to Fig. 7 (b), the execution time for the random forest is not the lowest but is comparable to the spark DT model. The training RMSE is lowest for Gradient-Boosted Trees, but the lowest holdout RMSE is noted for the spark random forest algorithm. Even though the random forest is an ensemble model, the execution time is not as large compared to other models. This is because the way spark performs its execution is that it utilizes its parallel computing capability to execute each of the decision trees individually and gives back the result. The actual power of

TABLE 3. Final ARMSE, for training and holdout dataset after choosing tuned parameters.

Algorithm	Training *ARMSE* (kWh)	Holdout *ARMSE* (kWh)
Spark Decision Tree Regression Model	9.171939547	10.80716307
Spark Random Forest Regression Model	8.01070855	10.60056138
Spark Gradient-Boosted Trees	4.206574519	11.88559708

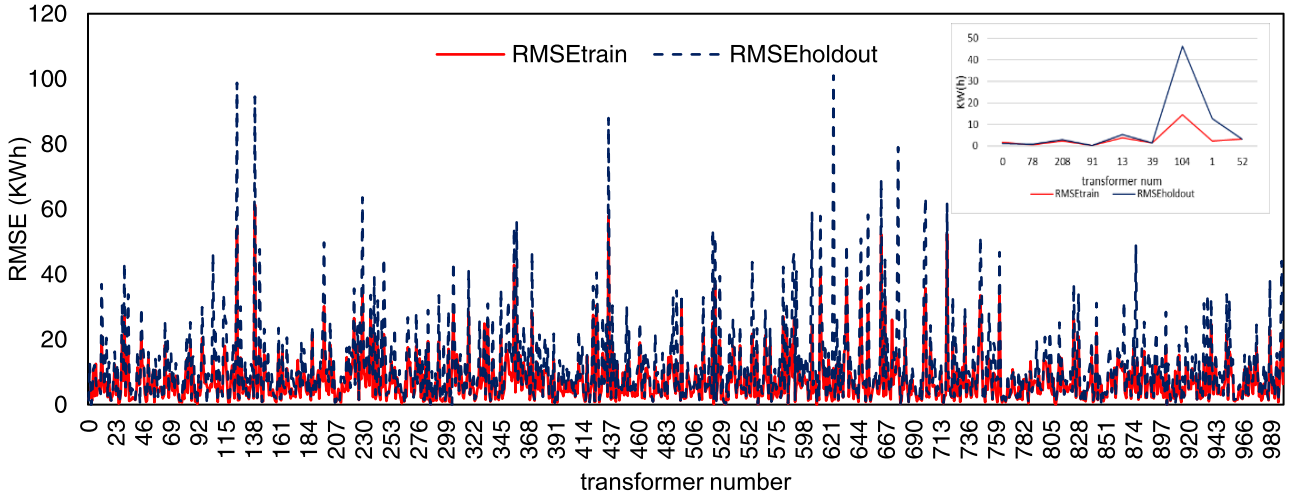

FIGURE 10. ARMSE comparison of training and holdout dataset for all the DT's.

spark in terms of execution can be observed here. Thus, it can be concluded that the spark RF performs better than the other spark ml models under comparison.

Fig. 10 shows the plot of *RMSE* of all the distribution transformers under consideration. The red line indicates the RMSE(kWh) obtained for all the distribution transformers. The blue line indicated as the holdout RMSE (RMSE of the data never used for training) is the forecasting error in kWh. To measure the quality of the trained models the holdout RMSE is expected to be as close as possible to the training RMSE. From Fig. 10, it can be observed that the blue line follows the red line for almost all the transformers. For randomly chosen DTs, indexed as 0, 78, 208, 91, 13, 39, 104, 1, 52 present the training *RMSE* and holdout *RMSE* zoomed in the top right of the figure. The plots indicate that the forecasting accuracy follows the training accuracy closely attributing to the fact that the built ML model is quite robust in terms of performance while increasing the speedup when a large number of jobs is performed.

VI. CONCLUSION

In this paper, a smart scheduling algorithm to perform load forecasting on multiple DTs was proposed. The proposed approach was implemented on Apache spark to not only deal with the challenges associated with computation time while handling the big data but also to submit jobs using an optimized methodology in a parallel manner. The processed big data was partitioned into various chunks and cached to improve the performance in terms of storage and in-memory processing. One distinctive characteristic of the proposed methodology is to be able to submit a maximum number of jobs and to process all the jobs in parallel. Several experiments were performed to optimize the scheduling strategy in terms of ML model error and execution time. A large number of DTs training procedures were performed with reduced run-times which allow handling big data that is too large to be stored. The training time for a group of 93 clusters data with a data size of ∼24 million records was performed in ∼50 sec and forecasting of 1000 transformers with ∼2.4 million records took ∼57 sec. The total time including, grouping, training, and forecasting was performed in ∼450 sec. The other important achievement of this paper is 2 times faster execution time with the use of thread pool and fair scheduler. This is a good optimization strategy for load forecasting using multi-sensor big datasets. Empirical evaluations significantly outperformed the previously proposed iterative algorithms. Moreover, the proposed ML models achieved higher accuracies. The merits shown in the experiment indicated that there is great potential for the proposed method to be used in big data load forecasting of multi AMI environments.

As this work chooses the optimized cluster value of 93, the next plan is to conduct experiments to investigate the optimal cluster value utilizing the proposed approach while using the spark platform. Also, scaling the dataset to more than 1000 DTs requires more than a minimum of 100 jobs to be submitted. Scaling the size of the spark cluster to an optimal value is a subject for future work.

ACKNOWLEDGMENT

The HPC (and/or scientific visualization) resources and services used in this work were provided by the Research Computing group in Texas A&M University at Qatar. Research Computing is funded by the Qatar Foundation for Education, Science and Community Development (http://www.qf.org.qa).

REFERENCES

[1] P. Wang, B. Liu, and T. Hong, "Electric load forecasting with recency effect: A big data approach," *Int. J. Forecasting*, vol. 32, no. 3, pp. 585–597, Jul. 2016, doi: 10.1016/j.ijforecast.2015.09.006.

[2] A. L'Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz, "Machine learning with big data: Challenges and approaches," *IEEE Access*, vol. 5, no. 1, pp. 7776–7797, 2017, doi: 10.1109/ACCESS.2017.2696365.

[3] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core vector machines: Fast SVM training on very large data sets," *J. Mach. Learn. Res.*, vol. 6, pp. 363–392, Apr. 2005.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *2nd USENIX Work. Hot Top. Cloud Comput. (HotCloud)*, vol. 10, 2010, p. 95.

[5] Y. Tang, Z. Xu, and Y. Zhuang, "Bayesian network structure learning from big data: A reservoir sampling based ensemble method," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, vol. 9645. Dallas, TX, USA, 2016, pp. 209–222, doi: 10.1007/978-3-319-32055-7_18.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008, doi: 10.1145/1327452.1327492.

[7] P. Mika, "Flink: Semantic Web technology for the extraction and analysis of social networks," *J. Web Semantics*, vol. 3, nos. 2–3, pp. 211–223, Oct. 2005, doi: 10.1016/j.websem.2005.05.006.

[8] A. Baldominos, E. Albacete, Y. Saez, and P. Isasi, "A scalable machine learning online service for big data real-time analysis," in *Proc. IEEE Symp. Comput. Intell. Big Data (CIBD)*, Orlando, FL, USA, Dec. 2014, pp. 1–8, doi: 10.1109/CIBD.2014.7011537.

[9] Y. Zhang, S. Chen, Q. Wang, and G. Yu, "i^2 MapReduce: Incremental mapreduce for mining evolving big data," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 7, pp. 1906–1919, Jul. 2015, doi: 10.1109/TKDE.2015.2397438.

[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Symp. Networked Syst. Design Implement.*, San Jose, CA, USA, 2012, pp. 15–28.

[11] N. Bharill, A. Tiwari, and A. Malviya, "Fuzzy based scalable clustering algorithms for handling big data using apache spark," *IEEE Trans. Big Data*, vol. 2, no. 4, pp. 339–352, Dec. 2016, doi: 10.1109/tbdata.2016.2622288.

[12] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, Santa Clara, CA, USA, Oct. 2013, pp. 1–16, doi: 10.1145/2523616.2523633.

[13] B. Hindman, A Konwinski, M Zaharia, A Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. NSDI*, 2011, vol. 11, no. 2011, pp. 295–308.

[14] T. White, *Hadoop: The Definitive Guide*, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2012.

[15] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. S. Wyckoff, and R. Murthy, "Hive: A warehousing solution over a map-reduce framework," *Proc. VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[16] L. George, *HBase: The Definitive Guide: Random Access to Your Planet-Sized Data*, 1st ed. Sebastopol, CA, USA: O'Reilly Media, 2011.

[17] Z. Hu, D. Li, and D. Guo, "Balance resource allocation for spark jobs based on prediction of the optimal resource," *Tsinghua Sci. Technol.*, vol. 25, no. 4, pp. 487–497, Aug. 2020, doi: 10.26599/TST.2019.9010054.

[18] R. E. Edwards, J. New, and L. E. Parker, "Predicting future hourly residential electrical consumption: A machine learning case study," *Energy Buildings*, vol. 49, pp. 591–603, Jun. 2012, doi: 10.1016/j.enbuild.2012.03.010.

[19] S. S. Reddy and J. A. Momoh, "Short term electrical load forecasting using back propagation neural networks," in *Proc. North Amer. Power Symp. (NAPS)*, Sep. 2014, pp. 1–6, doi: 10.1109/NAPS.2014.6965453.

[20] S. S. Reddy, C.-M. Jung, and K. J. Seog, "Day-ahead electricity price forecasting using back propagation neural networks and weighted least square technique," *Frontiers Energy*, vol. 10, no. 1, pp. 105–113, Mar. 2016, doi: 10.1007/s11708-016-0393-y.

[21] H. Shi, M. Xu, and R. Li, "Deep learning for household load forecasting—A novel pooling deep RNN," *IEEE Trans. Smart Grid*, vol. 9, no. 5, pp. 5271–5280, Sep. 2018, doi: 10.1109/TSG.2017.2686012.

[22] H. Aprillia, H.-T. Yang, and C.-M. Huang, "Statistical load forecasting using optimal quantile regression random forest and risk assessment index," *IEEE Trans. Smart Grid*, vol. 12, no. 2, pp. 1467–1480, Mar. 2021, doi: 10.1109/tsg.2020.3034194.

[23] S. S. Reddy, "Bat algorithm-based back propagation approach for short-term load forecasting considering weather factors," *Electr. Eng.*, vol. 100, no. 3, pp. 1297–1303, Sep. 2018, doi: 10.1007/s00202-017-0587-2.

[24] W. Jiang, H. Tang, L. Wu, H. Huang, and H. Qi, "Parallel processing of probabilistic models-based power supply unit mid-term load forecasting with apache spark," *IEEE Access*, vol. 7, pp. 7588–7598, 2019, doi: 10.1109/ACCESS.2018.2890339.

[25] *Classification and Regression—Spark 3.0.1 Documentation*. Accessed: Nov. 26, 2020. [Online]. Available: https://spark.apache.org/docs/latest/ml-classification-regression.html#decision-trees

[26] X. Meng, J. Bradley, B. Yavuz, and E. Sparks, "MLlib: Machine learning in Apache Spark," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1235–1241, 2016.

[27] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark SQL: Relational data processing in spark," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, May 2015, pp. 1383–1394, doi: 10.1145/2723372.2742797.

[28] *TAMUQ Research Computing Policies–Research Computing @ TAMUQ*. Accessed: Jan. 11, 2021. [Online]. Available: https://rc-docs.qatar.tamu.edu/index.php/Main_Page

[29] *STAR Project–Iberdrola*. Accessed: Jan. 11, 2021. [Online]. Available: https://www.iberdrola.com/about-us/lines-business/flagship-projects/star-project

[30] The Apache Software Foundation. *Spark Configuration*. Accessed: Feb. 11, 2021. [Online]. Available: http://spark.apache.org/docs/1.2.1/ec2-scripts.html

[31] D. Syed, H. Abu-Rub, A. Ghrayeb, S. S. Refaat, M. Houchati, O. Bouhali, and S. Banales, "Deep learning-based short-term load forecasting approach in smart grid with clustering and consumption pattern recognition," *IEEE Access*, early access, Apr. 8, 2021, doi: 10.1109/ACCESS.2021.3071654.

[32] D. Syed, S. S. Refaat, and H. Abu-Rub, "Performance evaluation of distributed machine learning for load forecasting in smart grids," in *Proc. Cybern. Informat. (K&I)*, Jan. 2020, pp. 1–6, doi: 10.1109/KI48306.2020.9039797.

[33] Y. Xu, H. Liu, and Z. Long, "A distributed computing framework for wind speed big data forecasting on apache spark," *Sustain. Energy Technol. Assessments*, vol. 37, Feb. 2020, Art. no. 100582, doi: 10.1016/j.seta.2019.100582.

[34] A. Zainab, D. Syed, A. Ghrayeb, H. Abu-Rub, S. S. Refaat, M. Houchati, O. Bouhali, and S. Banales Lopez, "A multiprocessing-based sensitivity analysis of machine learning algorithms for load forecasting of electric power distribution system," *IEEE Access*, vol. 9, pp. 31684–31694, 2021, doi: 10.1109/ACCESS.2021.3059730.

AMEEMA ZAINAB (Member, IEEE) received the bachelor's degree in electronics and communication engineering from Osmania University, Hyderabad, India, in 2013, and the M.S. degree in data science and engineering from Hamad Bin Khalifa University (HBKU), Qatar. She is currently pursuing the Ph.D. degree in electrical engineering with Texas A&M University (TAMU), College Station, TX, USA. She has three years of industry experience, working as a Data Analytics Professional, supporting audit at Deloitte Touche LLP, Hyderabad. She is also a base SAS Certified Programmer. Her research interests include data science, big data machine learning, power forecasting, and big data management in the smart grids.

ALI GHRAYEB (Fellow, IEEE) received the Ph.D. degree in electrical engineering from The University of Arizona, Tucson, AZ, USA, in 2000. He was a Professor with the Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada. He is currently a Professor with the Department of Electrical and Computer Engineering, Texas A&M University at Qatar. His research interests include wireless and mobile communications, physical layer security, massive MIMO, and visible light communications. He served as an instructor or a co-instructor in technical tutorials at several major IEEE conferences. He served as the Executive Chair for the 2016 IEEE WCNC Conference. He has served on the editorial board of several IEEE and non-IEEE journals.

HAITHAM ABU-RUB (Fellow, IEEE) received two Ph.D. degrees. He has been with many universities in many countries, including Poland, Palestine, USA, Germany, and Qatar. Since 2006, he has been with Texas A&M University at Qatar (TAMU-Q). He is currently a Full Professor of electrical engineering (EE). He is also the Managing Director of the Smart Grid Center-Extension in Qatar (SGC-Q). He has supervised many research projects on the smart grid and renewable energy systems. He has published more than 400 journal and conference papers, five books, and five book chapters. His principal research interests include smart grid, power electronic converters, renewable energy, and electric drives. He was a recipient of the American Fulbright Scholarship, the German Alexander von Humboldt Fellowship, and many national and international awards and recognitions.

SHADY S. REFAAT (Senior Member, IEEE) received the B.A.Sc., M.A.Sc., and Ph.D. degrees in EE from Cairo University, Giza, Egypt, in 2002, 2007, and 2013, respectively. For more than 12 years, he has worked in the industry as an Engineering Team Leader, a Senior EE, and an Electrical Design Engineer. He is currently an Associate Research Scientist with the Department of ECEN, TAMU-Q. He has published more than 100 journal and conference papers. His main research interests include power systems, electrical machines, smart grid, big data, development of fault-tolerant systems, reliability of power grids and electric machinery, fault detection, condition monitoring, and energy management systems. He is also a member of IET and the SGC-Q.

OTHMANE BOUHALI (Member, IEEE) is a currently a Research Professor of physics with TAMU-Q. He is also the Founder and the Director of the TAMU-Q Advanced Scientific Computing Center. He has been involved in the Large Hadron Collider research program for more than 25 years and has supervised various research projects. His research interests include large scale modeling, high-performance computing, and detector technologies for radiation and medical physics.