IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A Comparative Analysis on Parallel Implementations of Decision Tree Learning for Large Scale Complex Datasets in Apache Spark

[1]Surekha Samsani

[1]Assistant Professor

[1]Department of Computer Science and Engineering

[1]University College of Engineering Kakinada(A), JNTUK, Kakinada, India

Abstract: Decision Tree classification is one of the most widely used machine learning algorithm in dozens of sectors due to its classification effectiveness and model interpretability. However, the voluminous training data and the inherent problem complexity are the two major factors that strongly influence the time and computational efficiencies of the decision tree classification models. As the complexity of the problem increases, the number of computations required to learn every significant pattern will also increase. These enormous computations upsurge the training time and necessitate extensive computing resources especially when the training dataset is huge. To analyze such voluminous and complex datasets at a rapid speed, the need to shift to parallel processing approaches is essential for lowering overall computational costs. In this scenario, this paper provides a comparative analysis of various parallel implementations of decision tree classification models in Apache Spark on three UCI big datasets namely SUSY, PokerHand, and RLCP. The performance exhibited by various parallel implementations is compared in terms of scalability, time complexity and classification performances.

Index Terms **- Decision Tree classification, Machine Learning, Parallel implementation, Big datasets, Apache Spark.**

I. INTRODUCTION

Decision tree classification models acquire knowledge and enhance their performance through learning complex relationships represented by the training data. Suppose the complexity of the problem increases[4], the underlying decision tree induction logic performs a huge number of computations to unveil complex data patterns. These enormous computations necessitate extensive computing resources and increase the training time as depicted in figure 1.

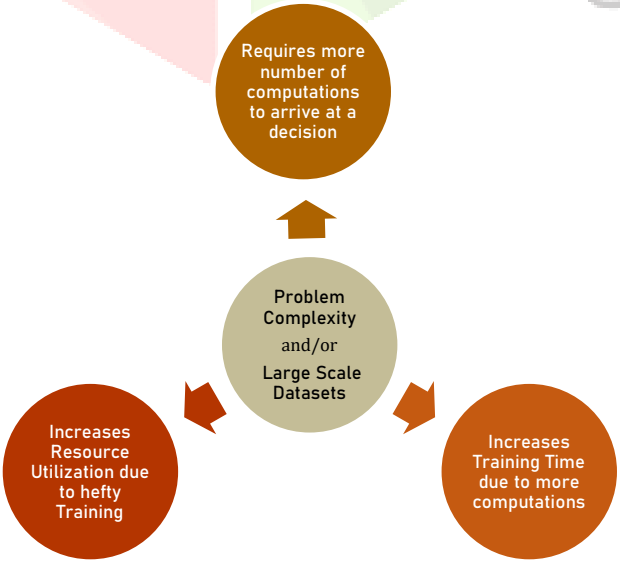

Figure 1 The impact of problem complexity and the size of training data on computational complexities of Decision Tree classification

On the other hand, the increased size of training data also needs a greater number of computing resources to learn all possible patterns. But such huge volumes of data[9] can be tremendously utilized to build better models that can predict future risks with higher precision. Despite gaining purposeful insights, processing such voluminous data[9] demands more computing power. Still, it remains a major challenging issue concerning resource utilization and timely generation of insights. Therefore, both the problem complexity and large-scale datasets drastically increase training time and computational resources. In this scenario, the traditional

way of building a single decision tree on complex and massive datasets is computationally expensive[13]. So, there is an increasing need for parallelizing the decision tree's training process to attain excellent scalability, performance, and minimize the time complexity. In this viewpoint, this paper provides a clear insight on selecting appropriate parallel processing paradigms for inducing scalable and cost-effective classification models. The literature related to parallel decision tree induction algorithms is presented in Section II. Section III elaborates the taxonomy of popular approaches for parallel implementation of decision trees. To compare the performances of various parallel implementations of decision trees, experiments have been carried out on three big UCI datasets. Furthermore, Apache Spark framework[22] is used in this research work for parallel implementation of decision tree induction algorithms. Because the experimental studies of Wang et. al.,[19], Huang et. al., [8], Ait-Mlouk et. al., [1], identified the "in-memory" processing capability of Apache Spark framework is supporting faster executions for implementing iterative algorithms like decision tree when compared over Hadoop MR1. The observed experimental evidences in various computing environments are presented in section IV. Finally, Section V concludes the paper by highlighting the significant observations.

II. Related Work On Parallel Learning of Decision Trees

Scaling up tree learning algorithms to large datasets is an area of active research interest. This section describes some past research developments for parallel implementation of decision tree construction algorithms.

Wang et. al., [19] discussed the limitation of the acyclic data flow MapReduce model for parallel implementation of iterative algorithms and implemented the C4.5 decision tree algorithm in both Apache Spark and MapReduce programming in a cluster environment with one master and 9 slave nodes. Experiments on Lymphography Domain Data Set observed a very significant 73-950%improvement in execution times (Small and Large datasets) of Spark than MapReduce. The authors also implemented the k-means clustering technique on the Apache Spark framework and observed 33 times faster execution over Map Reduce.

Desai et. al.,[5] investigated the performances of different implementations of decision trees which include sequential (DT), Distributed implementations using Map Reduce (DDT), Spark(ST), and Ensemble of trees(BT). Experiments were conducted in a pseudo cluster environment by splitting the data into 4 chunks. Weka's J48 is used to implement the DT and BT. In all experiments, BT has shown more improvement in increasing accuracy. DDT achieved 71% and 82% optimization over the respective tree sizes generated by DT and BT whereas ST achieved only 48% and 67%. For a large-scale dataset, BT's execution time is considerably larger than DDT and ST with only a 1% improvement in accuracy.

Ait-Mlouk et. al., [1] presented a distributed implementation of the C4.5 Decision Tree algorithm to analyze Road accidents. The authors used the Apache Spark Mlib library to predict the primary factors that lead to road accidents in the form of decision rules. Their experiments on big datasets found that the in-memory processing of Apache Spark is faster and beneficial for iterative algorithms when compared with the Map-Reduce programming model and exhibited 91.12% accuracy.

Venkatesh et. al.,[18] developed a Big Data Predictive Analytics Model using the Naïve Bayesian ML technique under the Hadoop-Spark framework. The proposed BPA-NB model assumes independence between features. Experiments were conducted in a single node cluster as well as in a multi-node cluster with 5,10,15,20 nodes. Experimental results showed increased processing time for the proposed system due to data parallelism and achieved better accuracy of about 97.12% to predict the rate of heart disease compared with existing systems.

Assefi et. al., [3] explored the computational perspective of SVM, Random Forest, Decision Tree, and Naïve Bayes machine learning algorithms for big data analytics using Apache Spark Mlib framework. The authors conducted experiments to compare the performance of Spark Mlib and WEKA library in two different cluster environments with 4 and 8 CPUs on Hepmass, Susy, Higgs, Flight, HetroactI, HetroactII datasets of UCI and RITA websites. Experiment results showed that the Area under ROC is pretty similar for both libraries and the performance t-test($p<0.01$) on the running times of Mlib and WEKA are shown that Spark Mlib is 35%faster.

Recent studies have shown that building a single decision tree from the entire collection of data suffers from communication overheads. And a lot of contributions proved that deriving or merging tree models with ensemble techniques, provide excellent scale-ups, speed-ups, and predictive performance across a wide variety of domains[13][20]. The parallel implementation approaches that derive a decision tree from the collection of tree models are presented below.

Amando et. al.,[2] explored some strategies for parallel implementation of decision tree construction algorithms. Various parallel processing techniques such as Data parallelism, Task parallelism, and hybrid parallelism have been discussed. And proposed a novel strategy for parallel implementation of the traditional C4.5 DT construction algorithm using a breadth-first strategy, data, and hybrid parallelism techniques. Experiments were conducted on a Synthetic dataset to evaluate the Speed-up and Scale-up characteristics of the proposed algorithm. From the obtained results, communication overheads are traced as the key factor in data parallelism and concluded that task parallelism performs better for smaller data sets.

In the literature of [13], *Panda et. al.,* described the benefits and challenges of Map-Reduce computing for parallel learning of classification and regression decision trees. Authors presented the PLANET, a scalable distributed framework that applies Map-Reduce to distribute and scale decision tree induction to very large datasets. Experiments on the bounce rate prediction problem unveils that training time increases as the data size increases. Moreover, PLANET involves a significant communication overhead for data transfer between computing nodes. And, increasing the number of machines didn't improve the performance because, after a certain point the overhead of collecting the results from worker nodes may dominate the benefits from each machine processing smaller chunks of data.

Dai et. al.,[6] implemented the classical C4.5 DT algorithm using Map-Reduce programming. Authors defined new data structures namely attribute table, count table, and hash table to minimize the communication cost. New strategies have been proposed for attribute selection, tree growing, and for updating tables as well. The proposed approach was implemented by vertically partitioning the dataset i.e., attributes have been distributed across the computing nodes. Experiments have been conducted on synthetic data in a Hadoop cluster with 8 nodes (4 PCS with dual-core) with 1GB RAM. And their approach exhibited excellent scale-ups and time efficiency over sequential DT induction algorithms as the size of training data increases.

Meng et. al.,[12] proposed a Parallel Voting Decision Tree algorithm for parallel implementation of decision trees called as PV-Tree. PV-Tree is a data-parallel algorithm that primarily aims to tackle the challenge of high communication costs. PV-Tree selects the top K best attributes from the local machines' data and then performs global voting to select the top 2K best attributes.

From these 2K attributes the best attribute will be chosen as the final splitting attribute. Experiments have been conducted on two datasets LTR and CTR by varying the number of computing machines. The results unveil that the PV-Tree achieved a very good trade-off between efficiency and accuracy and outperforms other parallel decision tree algorithms.

Last et al.,[20] proposed the SySM approach for reducing the classification complexity of the decision tree induction approach on big data platforms. Authors proposed the Similarity method ensembles the decision trees induced locally on partitioned data based on three metrics namely a syntactic, a semantic, and a linear combination of both. Comparing their experiments on six big datasets with the standard CART and J48 decision tree induction algorithms proved that the SySM approach provides more accurate decision trees with faster classification.

III. Taxonomy of Parallel Decision Tree Learning

The approaches for parallel implementation of decision tree learning[13] can be broadly categorized as follows,

1. Map-Reduce Approach
2. Ensemble Approach

3.1 Map-Reduce Approach

The conventional way of parallel decision tree implementation is to build a single decision tree from the entire collection of data through Map-Reduce(MR) implementation[13]. With the advances in parallel computing frameworks, the Map-Reduce approach became popular in recent times. In the Map-Reduce approach, a single consistent decision tree is induced from the entire collection of data distributed across a cluster of computing machines(commodity hardware) as shown in figure 2.

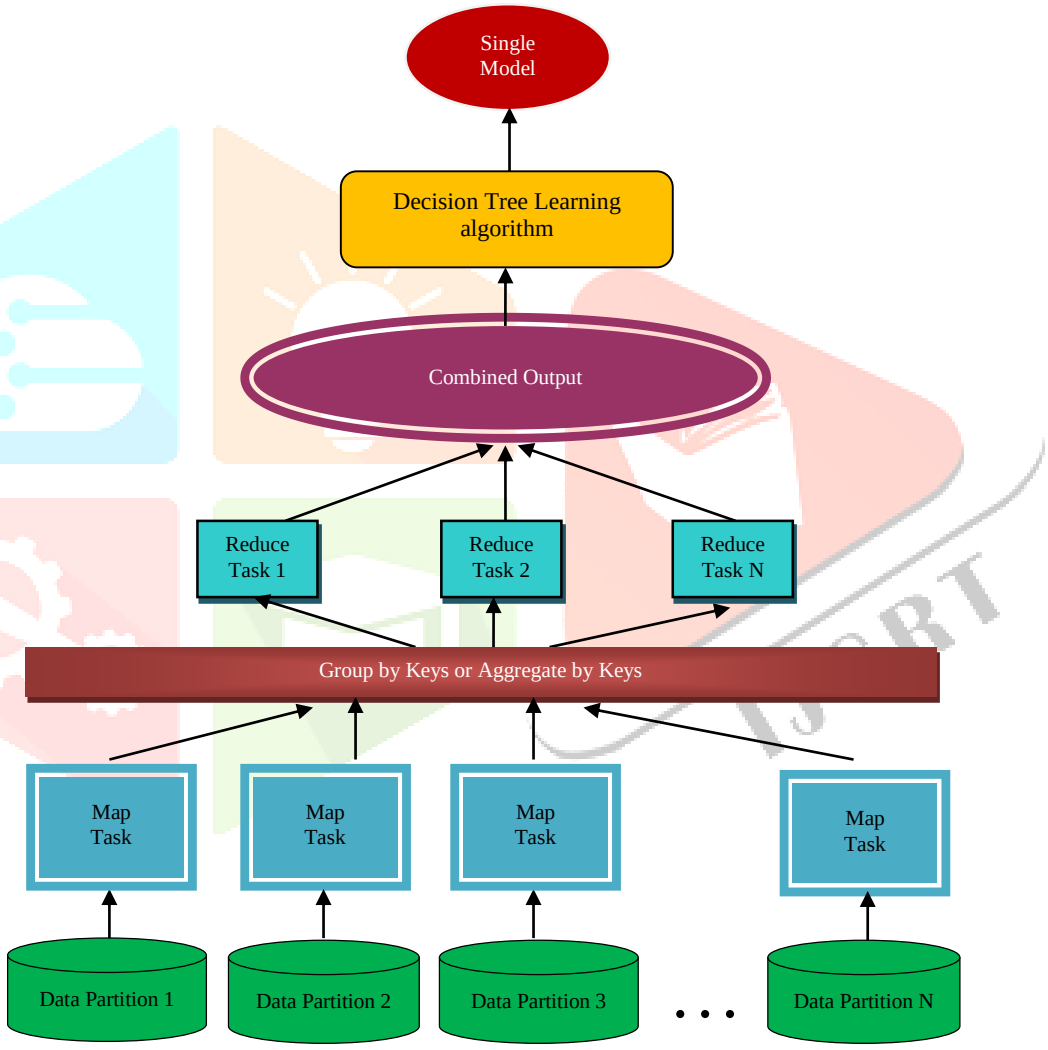

Figure 2. Map-Reduce Approach for inducing Decision Trees[17]

As shown in figure 2, user-defined Map and Reduce based procedures operate on local data partitions to produce required data in the form of Key-Value pairs. The result of Map-Reduce tasks is used by the driver program running at the master node to induce a single decision tree i.e., the decision tree induction logic is executed at the master controller. This paper implements the efficient VPRS-model based decision tree classification[16] algorithm for inducing optimal and accurate decision trees. The Map and Reduce functions have been designed to retrieve the required data from local data partitions by executing the same set of instructions on all machines having nearly the same computing power and local memory. This data transmission between worker nodes and master controller creates a lot of communication overhead across the computation nodes, increasing resource utilization and training time. Nevertheless, the paramount thing to understand the difficulties in parallel implementation of traditional Map-Reduce methodology is noticed as stated below.

- The selection of the best splitting attribute is not a static issue. It can be determined only at run time.
- The best split point selection requires a global view of the dataset from all of its partitions.
- Training of decision tree on a dataset spread across a cluster of machines requires data transmission from slaves to the master controller.

- The communication between machines at every level of the tree induction process unquestionably increases the training time to a great extent.
- Relatively more RAM is required at the master controller to process vast amounts of data from all worker nodes during the tree construction.

Due to the above reasons, Map-Reduce implementations of the decision tree classification on massive and complex datasets is not computationally efficient. There is a great necessity to design new strategies to parallelize decision trees' training process when dealing with large-scale and complex datasets.

3.2 Ensemble Approach

The ensemble approach derives the best tree structure from a collection of decision trees induced in parallel. It constructs decision trees locally from data available at each distributed data partition on local machines and then deriving the final decision tree that best represents the entire dataset from the collection of locally induced trees, and this process is depicted in figure 3.

In the ensemble approach, the decision tree classification algorithm will be executed on local machines and induce separate decision trees on the respective machines' data partition. These local decision trees will be transferred to the master node, where a single decision tree will be derived or selected from locally induced decision trees. If there are 'n' data partitions, then the logic to induce a decision tree is executed individually on each local data partition and generates 'n' separate decision trees. These 'n' locally induced decision trees will be analyzed further at the master controller to derive the final best decision tree. As per the research evidence provided by Weinberg et. al.,[20], the idea of the ensemble approach is a well-known approach and has been useful for implementing iterative algorithms like decision trees. Building a new decision tree from several induced decision trees has been proven as a well-known approach in mining large scale and complex datasets. This approach usually excels in attaining higher classification accuracy and scalability. Because inducing a decision tree from a set of decision trees also acts as an ensemble of classifiers and produces more accurate models with a low computational cost.

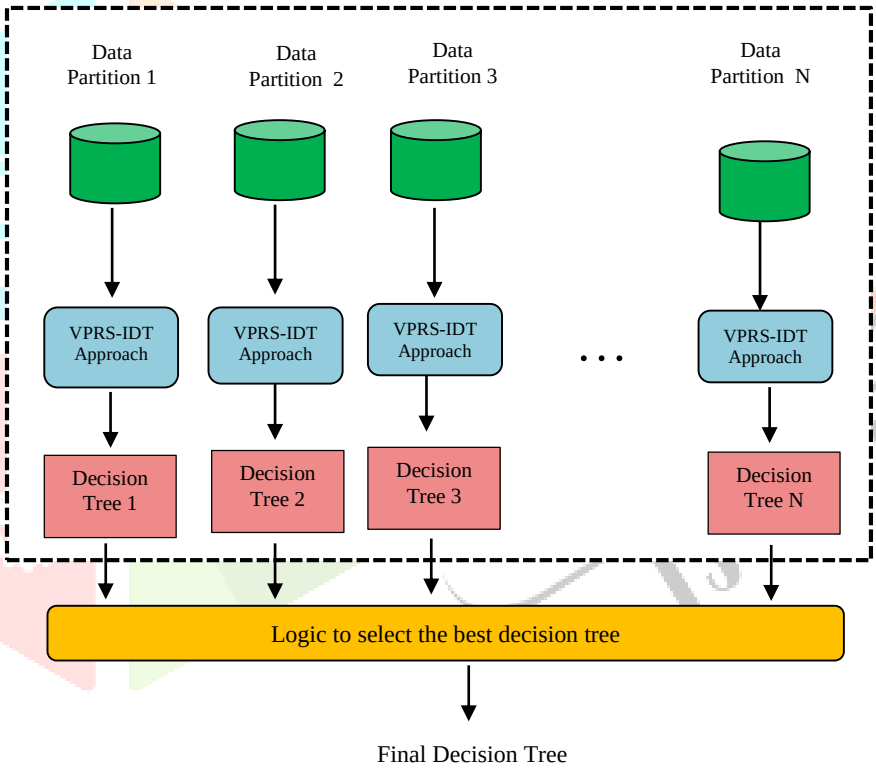

Figure 3 Ensemble approach for inducing decision trees

In this paper, the most effective PPAFL-DT algorithm[15] is used to implement the ensemble approach. The PPAFL-DT ensemble approach parallelizes the decision tree induction process and induces individual decision trees on local data partitions. These locally induced models are then transferred to the master controller to derive the best decision tree representing the entire dataset. Additionally, the transmission of these local decision trees is not computationally expensive as they consume very less network traffic. The step-by-step procedure to induce a single decision tree from an entire collection of data distributed over a cluster of machines is described below.

1) Partition the whole data set into 'N' equal sized chunks and distribute these partitions across machines available in a cluster.
2) One of these machines is designated as the master controller assigned the task of finding the final decision tree.
3) The decision tree classification task is then parallelized by fully utilizing the data level parallelism.
4) Each machine is responsible for inducing a decision tree using the locally available data partition.
5) All the induced models will be transferred to the master controller along with their testing accuracies.
6) Now, the data chunk available at the master controller is used to evaluate induced models' performance.
7) The master controller then uses all models' testing accuracy on the common test data and defines an efficient logic to select the final model.

The PPAFL-DT algorithm considers classification accuracy, β-significance of the root, and tree size as the primary requirements for qualification. If the data is more imbalanced, then the single decision tree induced by the Map-Reduce approach may contain rules that are dominated by majority classes. This problem can be mitigated by inducing the decision trees on smaller partitions of data as partitioning the entire dataset into multiple slices may become balanced to some extent. Therefore, the ensemble approach derives the best consistent decision tree by resolving the data imbalance problem. Also, the derived best model may perform well in exhibiting better classification performances.

IV. EMPIRICAL EVALUATION

This section elaborates on the extensive experimental evaluation conducted in this research work. The details of computational environments and the description of large-scale data sets used in this research study are presented.

4.1 Experimental Setup and Computing Environments

The parallel decision tree learning algorithms have been implemented in python version 2.7.0 programming under Apache Spark(version 2.2.0) framework. Apache Spark follows Master/Slave architecture[22] with a single Master and any number of distributed Executors. Apache Spark supports pluggable cluster management and is responsible for starting executor processes on-demand from the Driver process. In this research work, Distributed Hadoop File System is used to store large volumes of data. Hence, Hadoop YARN is preferred as a cluster resource manager. To fine-tune the performance of the Spark application, the common RDD's have been cached in RAM. The efficiency of the proposed work is evaluated by conducting experiments in two different computing environments. The first Computational environment1 is a fully distributed Apache Hadoop cluster deployment with one master node and 8 worker nodes. The configurations of master and worker nodes are given in Table 1. The second computing environment is a high computing machine with 512 GB RAM and a relatively higher processing speed. The configuration of the high computing bigdata server is given in Table 2.

Table 1. Configuration of Master and Worker Nodes in 8-Node Cluster

Component Name	Master Node Configuration	Worker Node Configuration
Processors	6x Intel® xeon® CPU E5-2640	1x Intel® xeon®CPU E5-2640
Cores	6 per processor and 2.5GHz clock speed	
Cache	15 MB Intel 7500 chipset with node controller	
RAM	64 GB RAM, HP SA 410i RAID controller with 1 GB FBWC	16 GB RAM, HP SA 410i RAID controller with 1 GB FBWC
Hard Drive	3x600GB 10K HDD	2x300GB 10K HDD
Operating System	CentOS 6.5 and Hadoop version 2.6	

4.1.1 Spark Configurations for Performance Tuning

Automatic optimization is not available in Spark it should be explicitly performed. And there are no predefined sets of recommendations available for ideal resource allocation to achieve a Spark application's finest performance. Instead, the key parameters such as executors, cores, and memory distribution for a Spark application must be calculated from the available cluster resources. Inefficient resource allocation in a cluster with limited resources may cause a Spark job to consume entire cluster resources and make other applications starve. The typical components important for any Spark application are mentioned below.

1. Number of Executors
2. Number of Cores per Executor process
3. Distribution of Executor and Driver memory

As per the worker node configuration given in Table 1, each machine has six cores and 16GB RAM. Leaving one core and 1GB of RAM for OS related issues, five cores and 15GB of RAM will be available. Now, the number of cores per executor can be assigned as per the approaches described below.

Table 2. Configuration of Big data server

:

Component Name	Configuration
Processors	8 x Intel® xeon® CPU E7-2830
Cores	8 cores per processor
Clock Speed	2.13 GHz
Cache	24MB Intel 7500 chipset with node controller
RAM	***512 GB DDR RAM***
Hard Drive	6x600GB 10K HDD
Operating System	CentOS 6.5 Server with Hadoop2.6

Tiny Approach: If we assign one core to each task, then five tasks are possible per machine. But leaving one more core to Hadoop daemons, we choose to run four executors on each machine. So, the total number of executors is 8*4=32. Running more executors with limited memory throws away the benefit of parallel programming as all tasks run in a single JVM.

Fat Approach: If we assign all cores to each task then only one task is possible per machine with 15GB RAM. Running an executor with a huge amount of RAM often results in excessive garbage collection delays.

Following the above discussion, the number of cores per executor in all experiments carried out in this research work has been set to two. And based on this information, the remaining parameters can be computed as demonstrated below.

- The number of cores available per node = 4
- Then the total number of cores in cluster = 4* 8= 32
- Total number of executors = $\frac{\text{Total cores}}{\text{No.of cores per Executor}} = \frac{32}{2}$ =16
- The number of executors per node = $\frac{\text{No.of Executors}}{8} = \frac{16}{8} = 2$
- Memory per executor = $\frac{\text{Total Memory}}{\text{No.of Executors per node}} = \frac{15GB}{2} = 7GB$
- Computing heap overhead as 7% of 7GB, which is obtained as 516Mb. Therefore, the actual executor memory is set as 6GB.
- The memory of the Driver process is set the same as that of memory of Executor.

4.2. DataSets

To compare the performances of various parallel approaches for DT classification, three big datasets of UCI ML repository [21] have been taken. And these three datasets are suitable for classification and used as benchmarks in several research studies on big data such as [20]. The details of the RLCP, PokerHand, and SUSY huge datasets and their characteristics are given in Table 3.

Table 3. Description of UCI Big Datasets

S.No	DataSet name	No. of Instances	No. of Attributes	No. of Decision Classes
1	Record Linkage Comparison Patterns (RLCP)	5,749,132	12	2
2	Super Symmetric (SUSY)	5,000,000	18	2
3	Poker Hand	1,025,010	11	10

4.3. Experimental Analysis and Results

This section presents and discusses the results of the experiments that have been conducted in this research work to evaluate the performance of the proposed tree selection algorithms. The experimental evaluation in this research work focused on three vital performance metrics: scalability, time complexity, and classification accuracy.

4.3.1. Scalability and Time Complexity

The first set of experiments have been designed to compare the performance of both MapReduce and ensemble approaches by parallelizing the tree induction process across a cluster of machines. In this scenario, experiments have been conducted in the computation environment1 described in Table 1. The Spark key configuration parameters are set, as mentioned below.

1. Number of Executors per node = 2
2. Number of Cores per Executor = 2
3. Executor Memory = 6GB

The details of each of the experiments carried out in environment1 are described below.

Experiment 1: Executing the spark jobs in computational environment1 described in Table 1, the total running time of Map-Reduce implementation on the RLCP dataset is observed as *70 minutes*. In comparison, the SUSY and Poker Hand datasets ran continuously for *11 hours* and *16 hours*, respectively and finally, jobs have been terminated due to *"Out Of Memory" error* at the driver process. But the ensemble approach can derive a decision tree when executed in the same computation environment. Figure 4 compares the Training times of decision trees induced by the ensemble approach on RLCP, SUSY, and PokerHand datasets.

Though the size of RLCP and SUSY datasets is five times greater than that of the PokerHand dataset, the execution times of RLCP and SUSY is relatively faster than that of PokerHand. It means that the time complexity of classification models is more influenced by the level of complexity in reaching an effective decision and less influenced by the number of samples.

Experiment 2: To overcome the *"Out Of Memory"* error at the driver process, in the second experiment driver memory, has been increased in a step of 2GB. Finally, the MapReduce jobs have been successfully executed for 12GB RAM. SUSY and Poker Hand datasets have been completed in *4.2 hours* and *7 hours*, respectively.

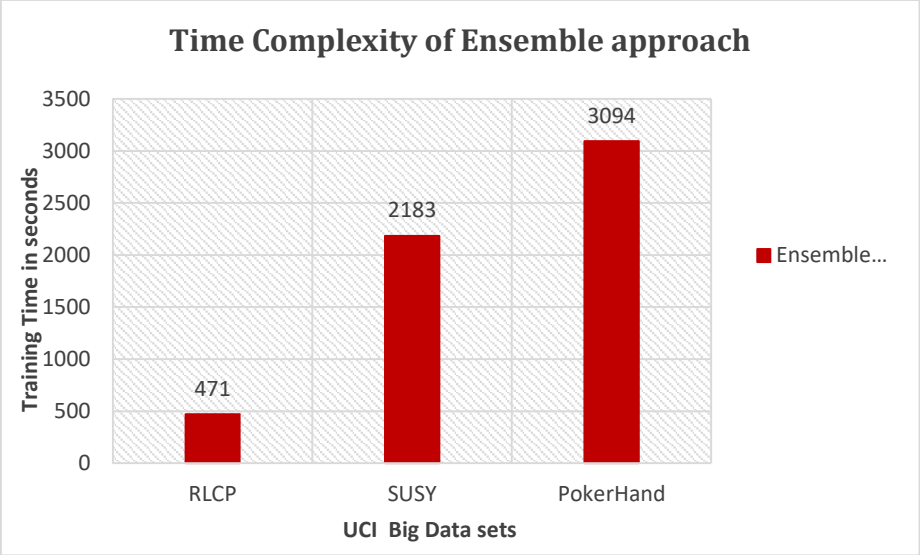

Figure 4. Training times of Ensemble approach on RLCP, SUSY, and PokerHand datasets

Though sufficient RAM is provided, the traditional Map-Reduce implementation of the DT classification took more time. Because all worker nodes work concurrently by executing the same set of instructions to provide the necessary data available in local data blocks. The data retrieved at each node is then transferred to the master controller to select the best splitting attribute at each level of the tree construction process. The complexity of the problem acts as a fuel to this data transmission, which in turn contributes maximum to increase the network access. This increased network access accelerates the training time to a great extent. Therefore, Map-Reduce implementation is not feasible for parallelizing iterative algorithms' training process like decision tree classification.

Experiment 3: The third part of the experiments has been designed to assess the scalability of the VPRS-IDT approach when more RAM and computing power have been added. The total running times for nine slices of RLCP, SUSY, and Poker Hand datasets have been observed as *135, 265, and 393 seconds* respectively.

4.2.2. Classification Performance

This section provides the classification performances of both the approaches measured in testing accuracy and tree complexity. Table 4 compares the performance of decision trees induced by each approach per dataset regarding average classification accuracy and average tree sizes.

Table 4 Classification performance of PPAFL-DT Vs Map-Reduce

S.No	UCI DataSet	β	PPAFL-DT		Map-Reduce	
			Accuracy	Tree Size	Accuracy	Tree Size
1	RLCP	0	***0.9998***	***163***	0.9915	91
2	SUSY	0.1	***0.8063***	***237***	0.7550	392
3	PokerHand	0.15	***0.8396***	***3145***	0.6566	3700

As shown in Table 4, for the massive RLCP dataset, the difference in classification accuracies of PPAFL-DT and Map-Reduce approaches is minor. However, the PPAFL-DT approach can induce more accurate trees for the SUSY and PokerHand datasets. For PokerHand dataset, the PPAFL-DT approach attained 83.96%, which is remarkably over the traditional Map-Reduce approach with 65.66%. Though the SUSY dataset is a binary classification problem, its accuracy rate remained below 90% due to the inherent problem complexity. And when comparing the tree sizes, the ensemble approach is very successful in inducing more optimal decision trees over the map-reduce approach.

In light of all the experimental observations, the Classification accuracy, Training time, and Tree Complexity of the classification models are more influenced by the level of complexity in decision and less influenced by huge sample size. Therefore, it can be concluded that the ensemble approach is very affective at handling data scalability and problem complexities.

IV. Conclusions

This paper provides a comparative analysis of map-reduce and ensemble approaches for parallel learning of decision trees. The performance of map-reduce and ensemble approaches have been evaluated by conducting experiments on very huge UCI datasets of around five Lakh instances in different computing environments. The observed investigations verdict that the map-reduce approach performs poorer for iterative algorithms like decision tree classification. But the Ensemble approach is very effective in minimizing inter-machine communication overheads over popular map-reduce approach and thriven for faster learning of decision trees through exploiting the complete ability of data-level parallelism. Moreover, ensemble approach attained a maximum 20% improvement over the Map-Reduce approach and is very effective in enhancing the scalability of the decision tree classification with optimal utilization of system resources.

REFERENCES

[1] A. Ait-Mlouk, F. Gharnati, T. Agouti, "Application of Big Data Analysis with Decision Tree for Road Accident," Indian Journal of Science and Technology, Vol. 10, 2017, DOI: 10.17485/ijst/2017/v10i29/117325

[2] N. Amado, J. Gama, and F.Silva, "Exploiting Parallelism in Decision Tree Induction", In: Proceedings from the ECML/PKDD workshop on parallel and distributed computing for machine learning, pp. 13–22, 2003.

[3] M. Assefi, E. Behravesh, G. Liu, and A. P. Tafti, "Big Data Machine Learning Using Apache Spark Mllib," In Proceedings of 2017 IEEE International Conference on Big Data (Big Data), pp.3492-3498, 2017.

[4] R. Blake, and P. Mangiameli, "The Effects and Interactions of Data Quality and Problem Complexity on Classification," Journal of Data and Information Quality (JDIQ), Vol.2, No.2, pp.1-28, 2011.

[5] A. Desai and S. Chaudhary, "Distributed Decision Tree", In Proceedings of the 9th Annual ACM India Conference, pp. 43-50, October 2016.

[6] W. Dai, and W. Ji "A MapReduce Implementation of C4. 5 Decision Tree Algorithm", International Journal of Database Theory and Application, Vol.7, No.1, pp.49-60, 2014.

[7] S. Gopalani, and R. Arora, "Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means", International Journal of Computer Applications, Vol.113, pp.8-11, 2015.

[8] K.M. Huang, H. Y. Chen, and K. L. Hsiung, "On Realizing Rough Set Algorithms with Apache Spark," In Proceedings of The Third International Conference on Data Mining", Internet Computing and Big Data, Konya, Turkey, pp.111-112, 2016.

[9] X. Jin, B. W. Wah, X. Cheng, and Y. Wang, "Significance and Challenges of Big Data Research", Big Data Research, Vol.2, pp.59-64, 2015.

[10] J. Li, Q. Chen, and B. Liu, "Classification and Disease Probability Prediction via Machine Learning Programming Based on Multi-GPU Cluster Mapreduce System," The Journal of Supercomputing, Vol. 73, No.5, pp.1782-1809, 2019.

[11] S. Maji, and S. Arora, "Decision Tree Algorithms for Prediction of Heart Disease", In Information and Communication Technology for Competitive Strategies, Springer, Singapore, pp. 447-454, 2019.

[12] Q. Meng, G. Ke, T. Wang, W. Chen, Q. Ye, Z.M. Ma, and T.Y. Liu, "A Communication-Efficient Parallel Algorithm for Decision Tree", In proceedings of 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, pp. 1-9, 2016.

[13] B. Panda, J.S.Herbach, S. Basu, and R.J.Bayard, " Planet: Massively Parallel Learning of Tree Ensembles with Mapreduce", In Proceedings of the 35th International Conference on Very Large Data Bases (VLDB-2009), Vol.2, No.2, pp.1426–37, 2009.

[14] M.K.Sreenivas, K.AlSabti, S.Ranka, "Parallel Out-of-core Decision Tree Classifiers", Advances in Distributed and Parallel Knowledge Discovery, Cambridge, MA, pp.317-336, 2000.

[15] S. Surekha and G. Jaya Suma, "Parallel Processing Algorithm for Fast Learning of VPRS-Model based Decision Tree on Big Data Platforms using Apache Spark Framework", International Journal of Advanced Research in Engineering and Technology (IJARET), vol.11, no.8, pp.122-138, Aug. 2020.

[16] S. Surekha and G. Jaya Suma, "VPRSM based Improved Decision Tree Induction Approach for Handling Uncertain Data," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol.8, no.8, pp.26-31, Jun. 2019.

[17] C. F. Tsai, W. C. Lin, and S. W. Ke, "Big Data Mining with Parallel Computing: A Comparison of Distributed and Mapreduce Methodologies," Journal of Systems and Software, Vol.122, pp.83-92, 2016.

[18] R.Venkatesh, C. Balasubramanian, and M. Kaliappan, "Development of Big Data Predictive Analytics Model for Disease Prediction using Machine LearningTechnique," Journal of Medical Systems, vol.43, No.8, pp.272, 2019.

[19] H. Wang, B. Wu, S. Yang, B. Wang, and Y. Liu, "Research of Decision Tree on Yarn using Mapreduce and Spark", In Proceedings of the 2014 World Congress in Computer Science, Computer Engineering, and Applied Computing, pp.21-24, 2014.

[20] A.I. Weinberg, and M. Last, "Selecting a Representative Decision Tree from an Ensemble of Decision-Tree Models for Fast Big Data Classification", Journal of Big Data, Vol. 6, Article No.23, pp.1-17, 2019.

[21] A. Asuncion, D. J. Newman, "UCI Machine Learning Repository", [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 2007.

[22] H. Karau, A. Konwinski, P. Wendell, M. Zaharia, "Learning Spark: Lightning-Fast Big Data Analysis", O'Reilly Media, Inc, 2015.