On Scalability of Distributed Machine Learning with Big Data on Apache Spark

Ameen Abdel Hai(✉) and Babak Forouraghi(✉)

Department of Computer Science, Saint Joseph's University, Philadelphia, PA 19131, USA
{aa671849, bforoura}@sju.edu

Abstract. Performance of traditional machine learning systems does not scale up while working in the world of Big Data with training sets that can easily contain petabytes of data. Thus, new technologies and approaches are needed that can efficiently perform complex and time-consuming data analytics without having to rely on expensive super machines.

This paper discusses how a distributed machine learning system can be created to efficiently perform Big Data machine learning using classification algorithms. Specifically, it is shown how the Machine Learning Library (MLlib) of Apache Spark on Databricks can be utilized with several instances residing on Elastic Compute Cloud (EC2) of Amazon Web Services (AWS). In addition to performing predictive analytics on different numbers of executors, both in-memory processing and on-table scans were used to utilize the computing efficiency and flexibility of Spark. The conducted experiments, which were run multiple times on several instances and executors, demonstrate how to parallelize executions as well as to perform in-memory processing in order to drastically improve a learning system's performance. To highlight the advantages of the proposed system, two very large data sets and three different supervised classification algorithms were used in each experiment.

Keywords: Big data · Machine Learning · Apache spark · Timing analysis Accuracy prediction · Data analysis · MLlib · Databricks

1 Introduction

The Internet has widely become the universal source of information for a considerable number of users. Astounding growth rates of online systems such as social media, mobile e-commerce applications, and many others continues to generate large amounts of data on a daily basis, and that has led to the phenomenon of Big Data [15]. Gathering enormous datasets of various types and structures can help enterprises perform advanced machine learning and predictive analytics such as fraud detection, risk analysis, economic and weather forecasts as well as advanced biometrics, which require very large training sets [1, 9–11, 13–15]. Thus, the world of Big Data has become of significant importance in our daily lives.

Unlike traditional structured data sets, Big Data is mostly unstructured (schema-less), and systems analyzing them utilize unconventional data manipulation techniques in order to efficiently perform complex operations. These operations should ideally be performed in-memory, rather than on-disk, with minimal computational latency and without the use of expensive hardware [1, 12]. Therefore, the performance and real-time training speed of machine learning systems on big data sets is of primary concern.

The Apache Hadoop is a distributed computational framework, which allows processing of large datasets across clusters of computers using the MapReduce programming model, and it has been used in advanced machine learning applications [15, 16, 18]. However, Hadoop exhibits high latency, which is directly due to its disk-persistent HDFS write operations as well as the general limitations of MapReduce such as the overhead of map jobs, high latency of storing intermediate computational results on disk and fault-intolerance [10, 16].

Apache Spark is a powerful and efficient in-memory framework for distributed processing, and it is fast becoming the mainstream distributed engine for performing advanced data analytics on big data sets [8, 10, 17]. A key feature of Spark is that its distributed computational platform is based on the Resilient Distributed Dataset (RDD) architecture, which caches intermediate results in order to perform in-memory processing [8, 11]. Resilient data sets are then split into a number of partitions and distributed across a cluster of workers/executors to prevent fault intolerance [1].

Spark's Machine Learning Library (MLlib) drastically reduces time-consuming analysis of mid-sized to very large datasets, and it enhances machine learning in general by its scalability and excellent ability for iterative processing [1, 3]. Further, MLlib supports combination of multiple algorithms in a single pipeline (workflow) to distribute stages across multiple executors as well as integration of various streaming and graphics tools [11]. Spark is flexible in that it supports a variety of programming languages for data scientists and engineers to develop workflows in Python, Scala, R, and Java [1].

To-date, there have been a few attempts to demonstrate the computational power of Spark on learning highly complex models using large data sets of high dimensionality [1, 4, 8, 13, 14]. However, the experiments reported in these works fail to discuss the scalability issue as it specifically relates to the speed-up of a supervised machine learning system across distributed Spark clusters of varied sizes. The aim of this paper, therefore, is to introduce a robust and efficient big data machine learning system, which can perform real-time, distributed learning of large and complex data sets in a computationally efficient manner. The implemented system utilizes Apache Spark in order to take advantage of its low latency and powerful parallel processing framework. To highlight accuracy and efficiency of deep learning on Spark, two large real-world datasets were analyzed using several classification algorithms. To measure the predictive accuracy of the learners on each dataset, several statistically independent experiments were conducted on Spark clusters of 1 to 5 workers/executors. Finally, to assess efficiency of Spark's in-memory persistence mode, experiments were performed using both in-memory and on-disk processing.

The remainder of this paper is organized as follows. Section 2 provides a brief description of Spark's key architectural features and its machine learning library MLlib. Section 3 reports on the outcomes of performing deep learning across several Spark clusters of varying sizes.

2 Apache Spark

Apache Spark was designed to work in conjunction with Apache Hadoop in order to address some of its limitations and downsides such as high latency [1]. Hence, Hadoop lacks the ability to perform Streaming/Real-time analytics since streaming requires persistent in-memory data processing capabilities. Fault-intolerance is also a significant downside of Hadoop, specifically, as regards insufficient memory capacity [4].

Spark overcomes Hadoop's design inefficiencies by utilizing the Resilient Distributed Dataset (RDD) architecture to perform in-memory parallel computation, and it is fault tolerant [1, 10]. An RDD is a collection of elements partitioned across nodes of the cluster that can be operated on in parallel [3]. Optionally, users might persist an RDD in memory, allowing it to be reused efficiently across parallel operations. In cases of failure, RDDs are automatically recovered from the executor failures. RDDs can be referenced to a dataset in external data storage system such as Amazon S3, HDFS, HBase, or any other supported Hadoop input format or by transforming a collection of objects in the program to RDD. Furthermore, RDDs support popular types of transformations such as maps, filters and aggregations [3].

Apache Spark jobs run as independent sets of processes on a cluster, coordinated by the object of SparkContext in the driver program [1]. Specifically, SparkContext connects to either Spark's standalone cluster manager, or YARN (resource manager) on Hadoop to allocate resources across applications. Upon establishing a successful connection between SparkContext and cluster manager, Spark demands workers/executors to compute and store data for the driver. Figure 1 depicts a typical Spark cluster consisting of a master node or driver that runs the entire process with a minimum of one worker or executor [10].

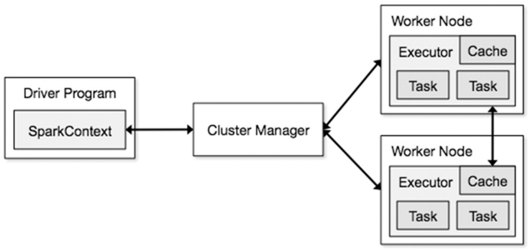


Fig. 1. A spark cluster

Spark's scalable Machine Learning Library (MLlib) consists of the most common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, and dimensionality reduction [3, 10]. MLlib revolves around the concept of pipelines, which allows users to combine multiple algorithms represented as an array of stages in a single pipeline to distribute stages to clusters of machines. Additionally, MLlib utilities support a variety of open source machine learning libraries and data formats such as LIBSVM datasets [1], which aids to perform faster and more efficient analysis since, it is a numeric dataset that excludes headers.

Having discussed the pertinent key features of Apache Spark in this section, Sect. 3 provides specific details as to how to learn from big data sets using Spark clusters of varying sizes and also how to implement classification evaluators for learning from big training data sets and making predictions on unknown test cases.

3 The Experiments

The main goal of the experiments conducted in this work was to assess scalability of big data machine learning on several configurations of Spark clusters. Specifically, the experiments were designed to directly measure relevant performance indices using two very large datasets and three machine learning algorithms. The following sections provide specific details of the utilized classification algorithms, the hardware configuration of the master and slave nodes in Spark clusters, and the obtained results for experiments conducted on each of the two big datasets.

3.1 Machine Learning Algorithms

The process of learning from big data and performing predictive analytics on Apache Spark comprises of two phases.

In Phase 1, datasets are randomly shuffled and split to perform training and testing; for instance, 70% of data may be used for training while the remaining 30% is set aside to evaluate the performance of the generated model. To take advantage of Spark's low latency, both training and testing datasets were made persistent in-memory to perform iterative operations during the learning phase. In addition, to further assess the advantages of in-memory RDD operations, the on-disk persistent model was used in each conducted experiment.

Phase 2 revolves around the concept of machine learning pipeline, which is chosen to combine multiple stages in a single pipeline, represented as an array of stages to scale and distribute to clusters of varying sizes. The first stage uses index values to perform evaluation of classification models, and datasets have to be enumerated and any arbitrary form of text data is not accepted. This is done by using the Vector-Indexer that indexes classification values to be predicted (*i.e.*, features) [5, 10]. The second stage is the classifier, which specifies which machine learning classification algorithm to use. For the experiments conducted in this work, it was decided to utilize three algorithms: Decision Tree, Logistic Regression, and Naïve Bayes.

A Decision Tree (DT) classifier is a machine learning algorithm that generates a predictive model to classify a dataset into target classes based on the training data. Each internal node in an induced DT represents a test on some property (feature) and leaf nodes represent classifications. DT performs well with large datasets, and the generated tree can help users easily visualize the diversion/flow of the data [9]. DT is slow because of representation structure of *replication and fragmentation,* and its ability to deal with missing values; hence, DT is known as Lazy Decision Tree [7].

Logistic Regression is a popular method to predict categorical responses. It is a special case of generalized linear models that predict the probability of outcomes [2]. This method can be used to predict binary (binomial) as well as multiclass (multinomial) outcomes. Multinomial logistic regression was used in the experiments reported here because of the many categorical groups of outcomes present in the training datasets.

Naïve Bayes is one of the most commonly used classifiers, that is based on Bayes Theorem to classify datasets by assuming that the presence of a particular feature in a class is unrelated to the presence of other features in the dataset [9].

Spark's MLlib provides Classification Evaluators with a suit of available metrics to evaluate the performance of its various classifier models [5]. The accuracy metrics used in this work to evaluate the actual versus the predicted outcomes are summarized in Fig. 2.

	Predicted = TRUE	**Predicted = FALSE**
Actual = TRUE	TP (True Positive)	FN (False Negative)
Actual = FALSE	FP (False Positive)	TP (True Negative)

Fig. 2. The confusion matrix

Spark's Multiclass Classification Evaluator was selected due to the fact that the two datasets utilized in this work were multinomial. The Positive Predictive Value (PPV) metric (see Eq. 1) was used to evaluate the accuracy of each classifier model since PPV considers the type of error, and hence it is the ideal metric to consider both the classification and misclassification rates to evaluate the pipelined transformations made by each classifier [5].

$$PPV = \frac{TP}{TP + FP} \tag{1}$$

3.2 Cluster Configurations

In order to assess Spark's efficiency, memory optimized EC2 (Elastic Compute 2) instances of Amazon Web Services (AWS) were created to distribute Spark jobs to cluster's executors, along with a general-purpose driver node on the cloud-based platform of Databricks, which is a unified analytics platform used to run Spark application on supported could platforms. In the experiments reported in the next section, memory-optimized instances were not utilized for the master node since computations do not reside on the driver, and hence, heavy instances are not required or necessary.

The Spark hardware instances used in all conducted experiments were identical in order to accurately and impartially perform timing analysis of both in-memory and on-disk scans of one to five executors including a driver. Specific cluster configurations are depicted in Fig. 3.

Driver Instances Type	General Purpose of m4.large
Worker Instances Type	Memory Optimized of r3.2xlarge
Worker Instances Memory/Core	61GB memory, 8 Cores
Driver Instances Memory/Core	8GB memory, 2 Cores
Operating System	Ubuntu 16.04
EBS Volume	General Purpose Solid State Drive

Fig. 3. Spark cluster configurations

3.3 Experiment 1

The dataset used in the first experiment is H-1B visa petitions, which includes over three million rows of data collected in 2011–2016 by the U.S. Department of Immigration Services [1, 12]. The dataset's unnecessary columns that had no relevance on the performance of the model (*e.g.* unique identifiers, dates, etc.) were removed; therefore, a total of seven features were selected along with the classification outcome *Case Status* as shown in Fig. 4.

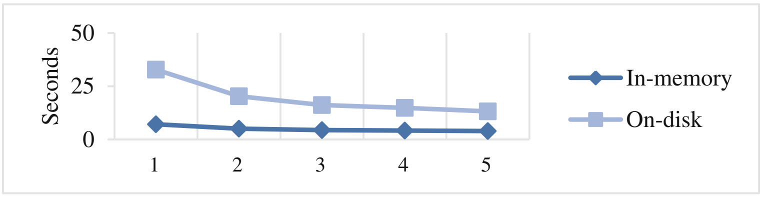
Variable Name	Type
Company	~220K Categorical Classes/Features
SOC	~2K Categorical Classes/Features
Job Title	~60K Categorical Classes/Features
Full Time (0, 1)	Binomial Categorical Class/Feature
Wage (grouped to 5 groups)	5 Categorical Classes/Features
Year	6 Real-valued features
Worksite (grouped by state)	50 Categorical Classes/Features
Case Status	8 Classification labels

Fig. 4. H-1B visa petitions features

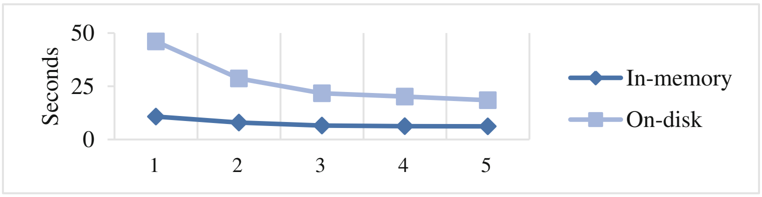
Timing analyses to learn from the training data and evaluate the generated model's classification accuracy required that classification labels and categorical data be enumerated (*i.e.,* indexed) and combined in a single vector. Thus, data cleaning was performed to format the dataset to LIBSVM and process it via Machine Learning Utilities (MLUtils) and MLlib Pipelines.

After converting the dataset into the required format, the three leaning models (see Sect. 3.1) were generated using clusters of one to five executors to assess Spark's performance and scalability. The results of timing analyses are depicted in Fig. 5, where each measurement is the average of five statistically independent runs for both in-memory and on-disk operations.

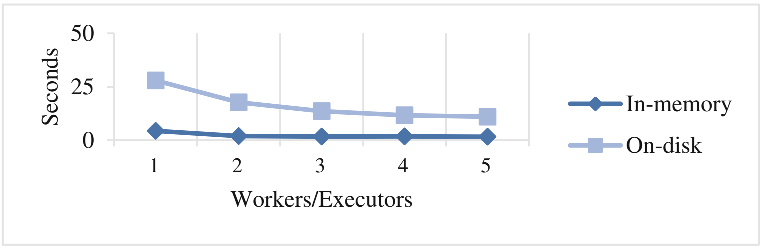
As shown in Fig. 5, there was a drastic difference between in-memory and on-disk operations, which can be directly attributed to Spark's low latency. Further, the most significant speedups occurred by increasing the number of executors from one to four. To determine whether further speedups would be possible, the Auto Scaling [1] feature,



(a) Logistic Regression



(b) Decision Tree



(c) Naïve Bayes

Fig. 5. H-1B visa data

which monitors the status of distributed jobs and automatically adjusts the number of executors on demand to improve performance, was enabled. However, no further speedups could be attained due to the size of the H-1B dataset.

Finally, the obtained classification accuracies for the three algorithms are shown in Fig. 6.

Learning Model	**Classification Accuracy**
Decision Tree	87%
Logistic Regression (multinomial) Maximum iterations = 3	87%
Naïve Bayes (multinomial) Smoothing = 1.0	56%

Fig. 6. Positive Predictive Value (PPV)

3.4 Experiment 2

The dataset used in the second experiment was the Fire Department Calls for Service of San Francisco, including 4.6 million rows of data collected by the city of San Francisco in 2015–2018 [6]. Similar to what was explained in the previous section, the necessary data cleaning and LIBSVM formatting was performed, and Fig. 7 summarizes the nine features or decision variables used in learning along with the classification label *Call Type*.

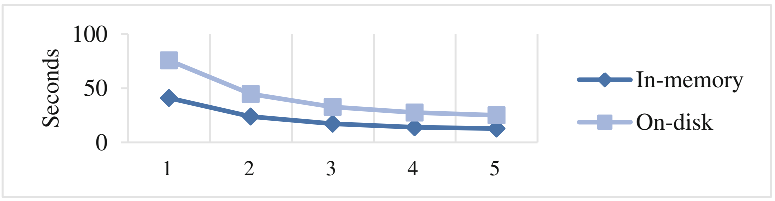
Variable Name	Type
Call Final Disposition	15 Real-valued features
Zip code	53 Real-valued features
Station Area	53 Categorical Classes/Features
Priority	6 Categorical Class/Feature
Final Priority	2 Categorical Classes/Features
ALS Unit	Binomial Real-valued features
Number of Alarms	5 Categorical Classes/Features
Unit Sequence	84 Real-valued features
Neighborhoods	42 Categorical Classes/Features
Call Type	32 Classification labels

Fig. 7. Fire department of San Francisco data model

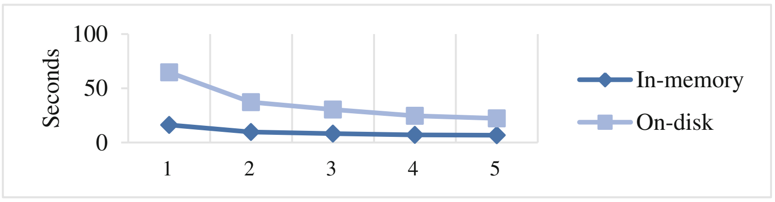
The three leaning models were generated using clusters of one to five executors. The timing analyses results are depicted in Fig. 8, where each measurement is the average of five statistically independent runs for both in-memory and on-disk operations.

Considering the larger dimensionality of the dataset used in this experiment, the learning task for each algorithm required more time. Clearly, distributing very large datasets among clusters with multiple executors is more efficient for big data, which naturally require more time for processing (*e.g.* 1 min or more).

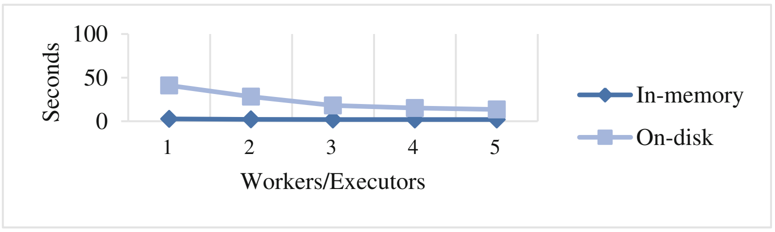
As depicted in Fig. 8, there were generally significant speedups measured for larger clusters of executors for in-memory but especially for on-disk operations. Logistic regression was the slowest algorithms due to the fact that in order to obtain reasonably high classification accuracy the number of iterations had to be increased to 20. Figure 9 summarizes the classification accuracies obtained from the three classifiers.



(a) Logistic Regression



(b) Decision Tree



(c) Naïve Bayes

Fig. 8. San Francisco fire department data

Learning Model	**Classification Accuracy**
Decision Tree	73%
Logistic Regression (multinomial) Maximum iterations = 20	69%
Naïve Bayes (multinomial) Smoothing = 500.0	52%

Fig. 9. Positive Predictive Value (PPV)

4 Conclusions

In this paper, it was demonstrated how a distributed system can be created to efficiently perform Big Data machine learning using various classification algorithms and very large datasets. Specifically, it was shown that Apache Spark's Machine Learning Library

(MLlib) on Databricks can be utilized using several instances (executors) residing on the Elastic Compute Cloud (EC2) of Amazon Web Services (AWS).

In addition to performing predictive analytics on different numbers of executors, both in-memory and on-table scans were utilized to assess the scalability and computing efficiency of Spark. The conducted experiments, which were run multiple times on several instances and executors, demonstrated how to parallelize executions as well as to perform in-memory processing to drastically improve a learning system's performance. To highlight the advantages of the proposed system, two very large data sets and three different supervised classification algorithms were used in each experiment. The obtained timing analyses confirmed that significant speedups can be attained by manually increasing the number of executors across the cluster. Further, it was shown that by using Spark's auto scaling feature, which increases the configuration of instances on demand, maximum speedups can be achieved depending upon the size of the training data.

In summary, the size of datasets chosen in this work did not warrant the use of larger clusters of executor/worker nodes since maximum speedups were achieved by using clusters of size three and four. However, in terms of future directions of this research, it will be beneficial to assess the scalability issue on much larger datasets and clusters.

References

1. Gupta, A., Thakur, H., Shrivastava, R., Kumar, P., Nag, S.: A big data analysis framework using apache spark and deep learning (2017). https://doi.org/10.1109/icdmw.2017.9
2. Classification and Regression: Classification and Regression - Spark 2.2.0 Documentation. https://spark.apache.org/docs/2.2.0/ml-classification-regression.html. Accessed 13 Mar 2018
3. Harnie, D., et al.: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 871–879. IEEE (2015). http://ieeexplore.ieee.org/document/7152571/
4. Miji, D., Varga, E., Member, S.: Machine Learning Driven Responsible Gaming Framework with Apache Spark, pp. 31–34 (2017)
5. Evaluation Metrics - RDD-based API. Evaluation Metrics - RDD-based API - Spark 2.2.0 Documentation. https://spark.apache.org/docs/2.2.0/mllib-evaluation-metrics.html. Accessed 13 Mar 2018
6. Fire Department Calls for Service. Open Data of San Francisco. https://data.sfgov.org/Public-Safety/Fire-Department-Calls-for-Service/nuek-vuh3. Accessed 7 Feb 2018
7. Friedman, J.H.: Lazy decision trees. AAAI **34**, 167–180 (1997)
8. Berral-garcía, J.L.: A quick view on current techniques and machine learning algorithms for big data analytics. In: 18th International Conference on Transparent Optical Networks (ICTON), pp. 1–4 (2016)
9. Vimalkumar, K., Radhika, N.: A big data framework for intrusion detection, pp. 198–204 (2017)
10. Wang, K., Fu, J., Wang, K.: SPARK – a big data processing platform for machine learning. In: 2016 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration, pp. 48–51 (2016)
11. Capuccini, M., Carlsson, L., Norinder, U., Spjuth, O.: Proceedings of 2015 2nd IEEE/ACM International Symposium on Big Data Computing, BDC 2015. Institute of Electrical and Electronics Engineers Inc., pp. 61–67 (2016)

12. Naribole, S.: H-1B Visa Petitions 2011–2016. In: H-1B Visa Petitions 2011–2016 | Kaggle (2017). https://www.kaggle.com/nsharan/h-1b-visa/version/2. Accessed 15 Mar 2018
13. Alfred, R.: [Plenary Speaker] The Rise of Machine Learning for Big Data Analytics, 2016 (2016)
14. Haupt, S.E., Kosovic, B.: Big data and machine learning for applied weather forecasts: Forecasting solar power for utility operations. In: Proceedings of 2015 IEEE Symposium Series on Computational Intelligence, SSCI 2015, pp. 496–501 (2016)
15. Mall, S., Rana, S.: Overview of big data and hadoop. Imperial J. Interdisc. Res. **2**(5), 1399–1406 (2016)
16. Biku, T., Rao, N., Akepogu, A.: Hadoop based feature selection and decision making models on big data. Indian J. Sci. Technol. **9**(10) (2016). https://doi.org/10.17485/ijst/2016/v9i10/88905
17. Liu, T., Fang, Z., Zhao, C., Zhou, Y.: Proceedings of 2016 IEEE/ACIS 15th International Conference on Computer and Information Science, ICIS 2016. Institute of Electrical and Electronics Engineers Inc. (2016)
18. Eluri, V., Ramesh, M., Al-Jabri, A., Jane, M.: A comparative study of various clustering techniques on big data sets using Apache Mahout. In: International Conference on Big Data and Smart City (ICBDSC), pp. 1–4 (2016). https://doi.org/10.1109/icbdsc.2016.7460397