Airline Delay Prediction using SPARK and HARP

Apurva Gupta
Masters, Data Science
Indiana University
Bloomington, IN-47408, USA
+1 (812)955-9257
guptaapu@iu.edu

Surbhi Paithankar
Masters, Data Science
Indiana University
Bloomington, IN-47408, USA
+1 (812)955-9256
spaithan@iu.edu

ABSTRACT

Big data is one of the major aspects of predictive analytics and sciences that poses significant challenges to the scientific community. The amount of data that is being generated by real time systems is humongous and the computational power required to analyze them increases proportionally. The field of machine learning requires high performance computing methods for building the models in effective ways. Focusing on classification problem, this paper implements the scalable random forest for big data algorithm with parallel implementation. We aim to build a model for predicting flight delays using real word data set consisting millions of records. We have conducted various numerical experiments leading to highlight the relative performance of different variants of algorithm as well as system specific parameters. We also perform comparison between two different implementations of big data systems i.e. Apache Spark and HARP.

1. INTRODUCTION

Flight delays are one of the major aspects of customer experience that is crucial for all commercial aviation. Every airline strives for reducing the flight delay for improving their customer base. There might be uncountable reasons that can cause delays like weather, time of day, busyness of airports and so on. By instrumenting machine learning, we can deliver the crucial factors that drive flight delays. In this project we have leveraged random forest algorithm to classify whether a flight will be delayed or not.

We know that the daily logs pertaining to flights are humongous. In this project, we used the most recent Airline dataset of RITA that has ~7M records. This dataset also demands data preprocessing and feature engineering that is again computationally expensive. Therefore, we used the power of big data systems to address all the computational and spatial requirements.

Apart from building the machine learning model, we have also focused on measuring the performance between two different big data systems i.e. HARP Vs Apache Spark. We aimed at analyzing the performances in terms of computation time and memory requirements for data preprocessing & Model building.

2. RELATED WORK

Owing to the high reliability on airways and practical significance, there has been a lot of background work done to predict flight delays using modern machine learning techniques like SVM, Logistic Regression, etc. We also came across papers that implement deep learning for analyzing the air traffic data. There are also many studies that attempt to model or predict flight delays using data on network congestions, routing topologies, cascading delays. We have aimed at allowing the origin, destination, carrier and flight time information to predict flight delays.

We would also like to acknowledge the implementation of random forest on airline data on HARP. We have employed them for comparing the performance with the PySpark implementation.

3. TASK AND DATASET

3.1 Dataset

We used Airline data available from RITA[6] for our data analysis. This data set consists of daily flight data from 1987-2008. It contains over 30M records per year that indicates the information of arrival & departures of each flight.
For our Analysis, we are using data from year 2008.

3.2 Feature Selection

We have the following information about Airlines:

FlightNum, CancellationCode, DepTime, CRSDepTime, ArrTime, CRSArrTime, ActualElapsedTime, AirTime, ArrDelay, DepDelay, Distance, TaxiIn, TaxiOut, CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, LateAircraftDelay and CRSElapsedTime
We converted the FlightNum and CancellationCode to String and rest other variables as Integer. Using PySpark Imputer, we imputed the missing values. We also encoded the Time variables into two parts: Hour and Minutes. We found the important variables relevant for the Airline delay and used them for our model building. The important features were: Dep hour of day, Arrival Hour of day, Actual Elapsed Time, Air Time, Distance, Dep Delay, Taxi In, Taxi Out. After pre-processing the dataset in PySpark, we split the data into train and test sets and put them into the Hadoop distributed file system for building models using HARP and Spark.

3.3 Feature Engineering

In order to build the machine learning model, we engineered the response variable 'IsArrDelay' using ArrDelay. If any given flight was delayed for more than 15 Minutes, then it was attributed as delayed i.e. IsArrDelay = 1. Our final dataset consists of csv file with label encoded and numerical features.

4. ARCHITECTURE

4.1 Random Forest

Random Forests are ensemble learning methods used for supervised classification and regression problems. Random Forests mitigates the problem of overfitting faced by decision trees. It constructs number of decision trees at training time and outputs the class which is the mode/mean of combined decision tree result.

Random Forest is a flexible, easy to use algorithm which produces great result even without hyperparameter tuning.

Random Forests adds randomness to the model. While splitting a node, it searches for the best feature among a randomly chosen subset of features. It constructs decision trees using bagging method by choosing a subset of data records to fit each tree. Thus, this added randomness and sampling with replacement mitigates the problem of overfitting.

4.2 Random forest on Distributed Framework

Traditional frameworks using statistical software such as R takes high computational time to build models because of their limited RAM capabilities. Therefore, we need to introduce big data frameworks which helps to build the statistical models on large scale data.[2]

Since RF are ensemble learning methods, we can construct several independent trees in parallel. This ensures faster implementation of the model in which many trees are built in parallel on different cores of the system. Also, RF requires intensive resampling for which parallel processing can significantly reduce run times for massive datasets.

4.3 SPARK Architecture

Apache Spark is a very powerful big data tool for tackling big data challenges. It is one of the most in-demand big data framework across all major industries. Spark has well-defined and layered architecture where all the spark components and layers are loosely coupled and integrated with various libraries and extensions[1]. It is based on two concepts:

4.3.1. Resilient Distributed Datasets (RDD)

RDD's are collection of data items that are split into partitions and can be stored in-memory on nodes of spark cluster. Spark RDDs' support two different types of operations- Transformations and Actions.

4.3.2. Directed Acyclic Graph (DAG)

DAG is a sequence of computations performed on data where each node is a RDD partition and edge is a transformation on top of data.

4.3.3 Spark Components

Spark follows a master-slave architecture where a Spark cluster has a single master and number of slave nodes.

1. Master

Master node is the central point of Spark shell (Scala, Python and R). The driver program runs the main function of application and creates the Spark Context.

2. Executor

It is a distributed agent responsible for execution of tasks. Every spark application has own executor process. They perform all the data processing. Also, they read from and writes data to external sources.

3. Cluster Manager

It is responsible for acquiring resources on spark cluster and allocating them to spark job. Choosing a cluster manager depends on goals of application. Examples of cluster managers: Hadoop YARN Apache Mesos or simple standalone spark cluster manager.

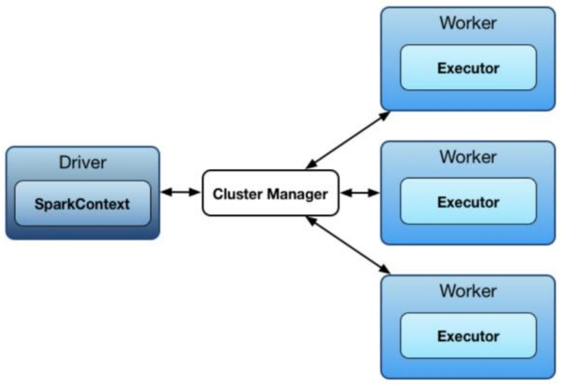


Figure 1. SPARK Architecture Layout [1]

.

4.3.1.1 WORKING

When client submits a spark user application code, the driver converts them into transformations & actions and later into logical DAG. Driver also performs various optimizations and converts the logical DAG into physical execution plan with set of stages. Then, it creates small physical execution units known as tasks. Driver manager requests for resources from cluster manager. Executors register themselves with driver program and then executors start executing the program. When we call the stop method, it terminates all the executors and releases the resources from cluster manager.

4.4 HARP Architecture

Harp[4] is a big data tool which provides data abstraction and communication abstractions. It can be plugged into Hadoop which helps to transform Map jobs to Map collective jobs and users can invoke efficient in-memory collective communication.

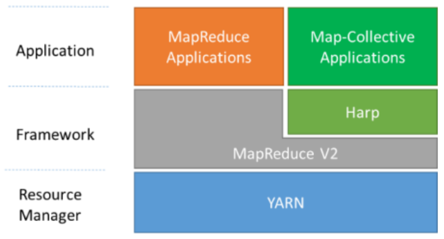


Figure 2. HARP Architecture [4]

Big Data technologies such as Hadoop Map Reduce define data as key-value pairs and computation as Map-Reduce tasks. However, communication patterns are not abstracted and defined in these tools. Harp provides these communication abstractions. It is a collective communication library which can be plugged into Hadoop. Using this plug-in, Map Reduce jobs can be transformed to Map-Collective jobs and in-memory collective communication are invoked directly in map tasks.

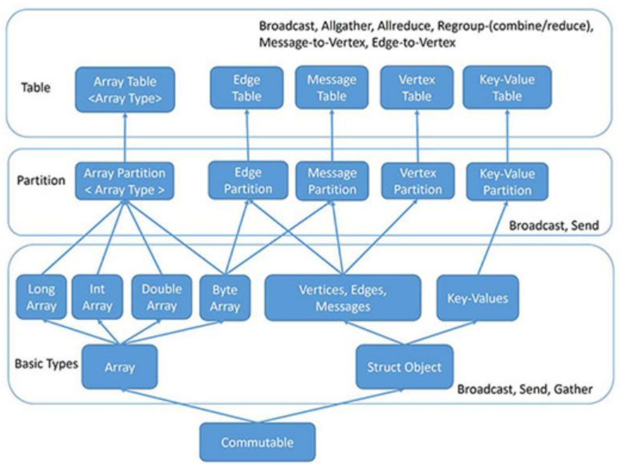


Figure 3. HARP Component Layout[4]

Harp has the following features: hierarchal data abstraction, collective communication model, pool-based memory management, BSP style Computation Parallelism, and fault tolerance support with checkpointing.

The three categories of horizontal data abstraction consist of arrays, key-values, edges and messages in graphs. Arrays and objects are the two basic types of vertical abstractions at lowest level, followed by key-value and graph partitions at middle level. At top level, partitions form tables that are identified using table IDs. These tables are observed as the distribution of dataset.

It is attributed as the shift of data partitions within tables. The three types include MPI collective communication operations, Hadoop MapReduce inherited collective communication and graph communication abstracted communication.

Data sending and receiving using serialization and de-serialization is an expensive operation. However, in HARP it is done on the byte array directly which are cached in a resource pool.

The Hadoop scheduler in harp is modified to schedule tasks in BSP style. Harp also incorporated fault tolerance with checkpointing. They perform several iterations onto small number of jobs and then submitted to cluster. Harp works as a plugin to Hadoop. It supports hadoop-1.2.1 and hadoop-2.2.0. To run map-collective jobs, users only need to put the harp jar package to the directory and perform configurations so that the map reduce jobs and map-collective jobs can run at the same time.

4.5 Future System Architecture

We have leveraged the big data system provisioned by Indiana university to implement our project. This Intel Xeon system has x86_64 architecture with 48 CPUs and 2 sockets. Each socket has 12 cores with 2 threads per core. The system runs on CPU E5-2670 v3 @ 2.30GHz.

5. IMPLEMENTATION

5.1 Random forest on Apache Spark

Apache Spark implements scalable machine learning algorithms through its built-in library MLlib[3] which interoperates with NumPy. This library is robust to any Hadoop data source facilitating easy plug into Hadoop data flows. MLlib contains algorithms that leverage iteration to yield faster results. In this project, we have used MLlib random forest algorithm and utilities for feature extraction and engineering.

5.2 Random forest on HARP

Harp extends the random forest implementation of javaml which is widely used java machine learning library. This extension uses distributed version instead of shared memory version. This procedure requires to specify the number of decision trees, mappers and threads as input parameters. We called the below HARP API for running the HARP RF algorithm on the prepared dataset.

hadoop jar contrib-0.1.0.jar edu.iu.rf.RFMapCollective <no. of decision trees> <no. of mappers> <no. of threads> <path to training data> <path to test data> /out

6. EXPERIMENTS

We conducted different experiments that emphasized the system behaviors under various settings. Firstly, we implemented the algorithm on the local node employing multithreading. This multithreading helps to run several threads parallelly by sharing same memory. Thereafter, we leveraged multiprocessing by running the RF algorithm across different number of slaves. We also tuned parameter specific to random forest algorithm to understand their effects. In order to evaluate the model performance, we calculated F-scores and test error. This is because the flight dataset consists of a huge class imbalance.

6.1 Parameter tuning

6.1.1.1 Decision trees

Random forest is an ensemble method typically based on building a model by constructing different number of decision trees. This is a hyper parameter indicating the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but makes your code slower.

No. Of Decision trees	F-score	Run Time (Sec)	Test Error
10	0.9275	107.8	0.0703
16	0.9276	121.7	0.0703
32	0.9274	153.6	0.0705
64	0.9274	184.5	0.0704
128	0.9278	228.6	0.0701

Table 2. Model Evaluation metrics across different number of decision trees

In the table above, we observed that as the number of decision trees were increased, the model was predicting better. We see drop in test errors and better F-scores with the increase in the number of trees. On the other hand, the runtime of the program worsened with increased trees. Hence, its important to choose the number of trees wisely such that the runtime is feasible, and the built model is good enough to perform the predictions on big data sets.

6.1.2 Number of nodes and threads

Multiprocessing and multithreading are important features of central processing units to execute multiple processes or threads concurrently. We ran our experiments over different nodes and for different number of threads. We evaluated the computational time and memory usage for multithreading and multiprocessing for spark.

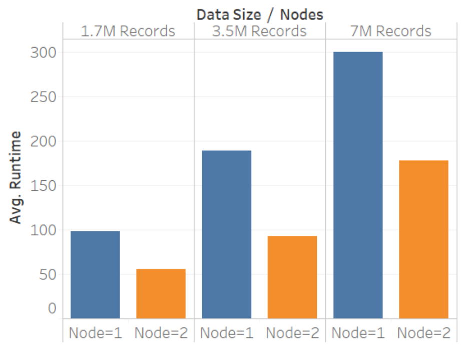


Figure . Avg. Runtimes across different nodes and data sizes

We ran the spark program for three different sizes of data sets for analyzing the system performance. We observed that when the random forest model was run on local machine for just one node, the average runtime was higher. However, it dropped by ~40% when we ran the program leveraging parallel processing across two nodes. Also, we noted that more memory is consumed for when the program was run using multiprocessing.

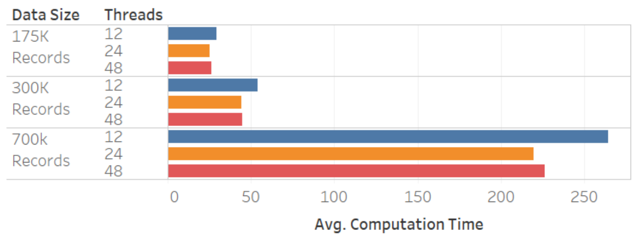


Figure 5. Avg. Runtimes across different data sizes and threads

Apart from the multiprocessing, we also analyzed the average runtimes for same local machine but for different number of threads on HARP. As seen in figure 5, We observe that the relative average runtime decreases with increase in threads. We see a significant improvement in runtimes when the threads are increased from 12 to 24. However, when the threads are increased to 48, there is no notable difference. This may be attributed to the speed up achieved by parallel processing and the additional communication runtime overhead with increase in number of threads.

6.2 SPARK VS HARP COMPARISION

In this section, we focused on measuring the system performance across spark vs HARP. We built and ran the model using the preprocessed datasets on spark and harp to record the results.

We observed to have similar F-scores and test errors across both the system.

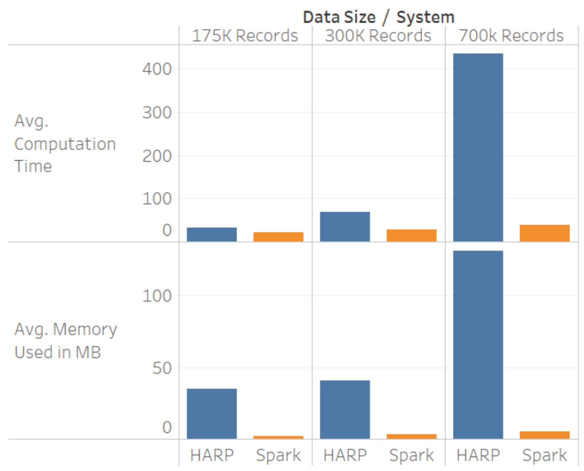
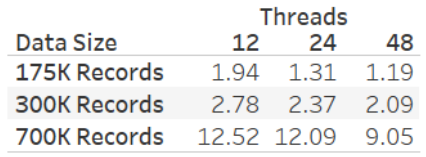


Figure 6. Avg. Runtimes & Memory usage for Spark vs HARP

We recorded the average runtimes and memory consumption of the program run using spark vs HARP architecture. Looking at the graphs, its evident that the spark programs run faster than Hadoop Map Reduce framework.



	Threads		
Data Size	12	24	48
175K Records	1.94	1.31	1.19
300K Records	2.78	2.37	2.09
700K Records	12.52	12.09	9.05

Table 2. Avg. Speedup of Spark over Harp

This can be attributed to the in-memory computation feature of Spark. We calculated the test statistics using Intel's VTUNE tool. The harp program took more memory than the Spark programs. These results insinuate us that the spark programs are better than Hadoop when we need to perform analysis in real time applications. However, when we need to process large amount of data, the economical infrastructure of Harp is a better alternative.

7. CONCLUSION

Random forest is one of the widely used algorithms in the industry today because of its powerful ability to model any given dataset without overfitting. The inherent structure of the algorithm that couples with possible parallel implementation makes it scalable and practical for running over huge datasets. In this project, we implemented the algorithm using different APIs and observed sufficiently high accuracies and F-Scores. We also saw the impact of parameter tuning, multithreading and multiprocessing methods that effected the corresponding runtimes. Synchronization in distributed systems is harder than in centralized systems because relevant information is scattered among multiple machines.This introduces additional runtime overheads to the system. With the HARP plug-in, MapReduce jobs can be transformed to Map-Collective jobs and users can invoke efficient in-memory collective communication directly in map tasks.
Finally, we investigated spark and HARP framework and observed that spark programs ran faster than HARP. The key difference between Hadoop MapReduce framework and Spark is that the former was designed to process data in batches, reading and writing to disks while Spark can process data in micro-batches and stores/processes data mainly in memory. As a result, the speed of processing differs significantly. Therefore, we would recommend

Spark as a better option if speed of processing is critical and adequate RAM is available for in-memory processing. It is an effective paradigm wherever real-time results are expected. Even though Spark outperforms MapReduce in most scenarios due to its in-memory processing features, there are cases when MapReduce is probably a better fit. For example, when there is a large volume of data to be processed and there is not enough RAM to store this data in memory, it may be a good alternative to process the data with HARP.

8. FUTURE WORK

In this project, we learnt how the distributed paradigms help the data science community for analyzing real time data applications. We had the opportunity to understand how the machine learning algorithms run across different data nodes. Given more time and opportunity we would also like to go ahead and perform experiments for a larger dataset across a greater number of nodes. We would also like to experiment with different machine learning algorithms like SVM, logistics regression etc. for building the flight delay prediction models.

9. ACKNOWLEDGMENTS

The success and outcome of this project required a lot of guidance and assistance from many people and we are extremely fortunate to have got all this during out project work. We would like to express our sincere gratitude to Prof. Judy Qui and Prof. Peng for providing us all the support and guidance which helped us complete the project on time. We would also like to appreciate the efforts taken by the teaching assistants by presenting their timely guidance and ideas.

10. REFERENCES

[1] Genuer, R., Poggi, J.-M., Tuleau-Malot, C., Villa-Vialaneix, N., 2015. Random Forests for Big Data. arXiv preprint arXiv:1511.08327

[2] Wright, M.N., Ziegler, A.,2015. ranger: A fast implementation of random forests for high dimernsional data in C++ and R. arXiv preprint arXiv: 1508.04409

[3] Assefi, Ehsun Behravesh, Guangchi Liu, and Ahmad P. Tafti: Big Data Machine Learning using Apache SparkMLlib, IEEE BIG DATA 2017

[4] http://salsaproj.indiana.edu/harp/

[5] http://udspace.udel.edu/bitstream/handle/19716/17628/2015_LiuLu_MS.pdf?sequence=1

[6] http://stat-computing.org/dataexpo/2009/the-data.html