Applying Scalable Machine Learning Techniques on Big Data using a Computational Cluster

Dev Dua[1], Sujala D. Shetty[2], Lekha R. Nair[3]
Department of Computer Science
BITS Pilani – Dubai Campus
Dubai, U.A.E.
devdua@live.com[1], sujala@dubai.bits-pilani.ac.in[2], lekharnair@gmail.com[3]

Abstract— **Machine Learning is a relatively new avenue in exploring Big Data, and this involves having a working understanding of the commonly used machine learning techniques, and the algorithms that each technique employs. There will be a focus on making the algorithms scalable to utilize large amounts of data, and this will be done using open source machine learning tools and libraries. Since big data resides on the internet, or on a cloud network, the machine learning algorithms studied in this paper will be utilized in applications deployed on a cloud service like Windows Azure or Amazon Web Services, which will carry out compute tasks on big data residing in the cloud.**

Keywords - Big Data, Machine Learning, Cluster Computing

I. Introduction

The computers of the current year have been improving exponentially in terms of performance as per Moore's Law, and development of fast and efficient computing platforms has significantly helped us to understand computationally and structure-wise complex systems, such as biochemical processes, and sophisticated industrial production facilities and financial markets [7]. The human tendency of thinking and analyzing, and further predicting, arises from the fact that given historical data, we can estimate and model the processes in the system at a level of abstraction that, although not able to provide a complete understanding of the inner workings, is detailed enough to provide useful information about dependencies and interconnections at a higher level. This, in turn, can allow us to classify new patterns or predict the future behavior of the system.

We have been harnessing the processing power of computers to build intelligent systems, systems that, given training data or historical data as mentioned above, can learn from, and as a result give us results when the test data is fed into the system. During the previous few decades, there has been incremental growth in our data generation and storage capabilities [2]. In general, there is a competitive edge in being able to properly use the abundance of data that is being collected in industry and society today. Efficient analysis of collected data can provide significant increases in productivity through better business and production process understanding the highly useful applications for e. g. decision support, surveillance and diagnosis.

The focus of this paper is on exploring and implementing intelligent applications that harness the power of cluster computing (on local machines as well as the cloud) and apply machine learning on big data. However, the concepts that will be explored are by no means specific to these fields, and can be extended/modified for other fields as well.

II. Objectives

The objective of this paper is to meet the following objectives:

- Explore machine learning techniques, and evaluate the challenges faced when operating on Big Data.
- Explore current machine learning libraries, analyze the feasibility of exploiting them on a cloud platform
- Understand the basics of cluster computing, and how an Apache Spark cluster can be setup on Microsoft Azure.
- Cluster geospatial data, and analyze the performance of the implementation on a cluster.

III. Understanding Machine Learning

To put it simply, one can say that machine learning focuses on designing and implementing algorithms and applications that automatically 'learn' the more they are executed. We will however not be concerned with the deeper philosophical questions here, such as what learning and knowledge actually are and whether they can be interpreted as computation or not. Instead, we will tie machine learning to performance rather than knowledge and the improvement of this performance rather than learning. These are a more objective kind of definitions, and we can test learning by observing a behavior and comparing it to past behaviors. The field of machine learning draws on concepts from a wide variety of fields, such as philosophy, biology, traditional AI, cognitive science, statistics, information theory, control theory and signal processing. This varied background has resulted in a vast array of methods, although their differences quite often are skin-deep and a result of differences in notation and domain. Here we will briefly present a few of the most important approaches and discuss their advantages, drawbacks and differences.

A. *Association Rule Learning*

ARL is an ML method for discovering relations among attributes in large transactional databases, and is quite popular and well researched. The measures used to discover similarities are varied, and it mainly involves generation of item sets recursively to finally build the rules, based on support count and confidence. This way of learning is often applied in market basket analysis (affinity analysis) where trends that relate products to transaction data are discovered to boost the sales of the organization.

B. *Artificial Neural Networks*

An ANN learning algorithm is inspired by the structure of the biological computer i.e. the brain, and is structurally designed in a manner similar to biological neural networks. The interconnected group of artificial neurons structure and divide the computation in such a manner that information can be processed in a parallel manner. Applications of NNs include use in tools that model non-linear statistical data. NNs make it easy to model complex relationships and process a large amount of inputs and compute outputs in a massively parallel manner. Other applications include pattern discovery and recognition, and discovering structure in statistical data distributions.

C. *Support Vector Machines (SVMs)*

SVMs, is a binary learner used for regression and classification, are supervised ML methods. It is applied mostly to categorical data, where the training set of data has records belonging to 1 of 2 categories. The model generated by the SVM training algorithm is then used on the test data to predict which category does each record fall into. Thus it can be seen as a non-probabilistic linear classifier. The data is represented as points in space, mapped so that the 2 categories are divided by a gap that is ideally as far apart as possible. The test records are then fit into the same space so that they fall into a point in space that corresponds to the category they fall into.

D. *Clustering*

Clustering can be viewed as separating records of data into subsets, called clusters, so that data points lying within the same cluster are highly similar, and this similarity is determined by employing pre-designated criteria. Data points belonging to different clusters are ideally placed as far as possible, i.e. they are highly dissimilar. There are various types of clustering techniques – partitional, hierarchical, and density based clustering being the most common. They are built on the basis of some similarity metric and the result of clustering is scrutinized by looking at the relative placement of members within the same cluster (internal compactness), and also how well separated different clusters are from each other. This ML method is an example of unsupervised learning. Applications of clustering are varied, from spatial data analysis to document clustering.

E. *Collaborative Filtering*

CF is a recommendation technique being increasingly for generating suggestions/recommendations. Collaborative filtering can be viewed as the process of filtering information to discover patterns involving 'collaboration' among data sources, viewpoints, multiple agents, etc. Collaborative filtering can be applied to very large data sets, and is a commonly applied to social media and entertainment services, like Netflix.

These approaches above are applied to many types of data sets, which vary in size, structure, attributes and complexity. Also, most of these approaches don't work well with all kinds of data, i.e. there is no 'super-algorithm' that can encompass all types of data sets. Therefore this is one problem that connects machine learning with big data. This scenario is better described as scalability [6], where the application/algorithm has to be redesigned to deal with huge sets of data, which are structurally big and complex to be read and operated upon by conventional computers. The structure of the data being used also matters, and impacts the way that it has to be pre-processed before the machine learning application can actually start working on the data.

IV. Big Data and the Challenges to Data Analytics

Big data is a buzz word used to describe the explosive generation and availability of data, mainly on the web [1]. Big Data, going by the name, is so large that traditional software techniques and databases fail to process this exponentially growing structured and unstructured data. It is not only the monolithic structure of big data that makes it a challenge, other factors include its rate of generation (that might be too fast to capture such huge amounts of data successfully without losing the other incoming data) or one may not have the processing prowess to quickly analyze the data. It can be characteristically described by [10] -

- *Volume:* This describes the scale of data being handled. An estimate shows that 40 zettabytes (equivalent to 43 trillion gigabytes) of data will be created by 2020, a 300x increase compared to data generated by 2005. It is also estimated that 2.3 trillion gigabytes of data are generated every day, and is exponentially growing.
- *Variety:* This refers to the different forms of data. It also indicates the various sources that generate structured and unstructured data. Taking healthcare as an example, in 2011 itself, data in healthcare was estimated to be 161 billion gigabytes. On YouTube, more than 4 billion hours are viewed every month.
- *Velocity:* It deals with the rate at which sources like human interaction with things like social media sites, mobile devices, etc., networks, machines and business processes, generate the data. This characteristic is most important when dealing with huge flows of streaming data. Velocity of Big Data can be handled by sampling data from data streams. For example, 1TB of information about stock trades is captured by the

New York Stock Exchange during each trading session. If this is analyzed in an efficient way, businesses can really benefit.

- *Veracity:* Veracity describes the abnormality, biases, noise and inaccuracy in data. The immense flow and size of the data itself is so overwhelming that noise and errors are bound to exist. Thus, to have clean data, filters and other monitoring measures need to be implemented to prevent 'dirty data' from accumulating.

Loosely structured data is often inaccessible and incomplete. Difficulties in being able to create, manipulate, and manage big data are the most common problems organizations face when dealing with large databases. Since standard procedures and tools are not built from the ground up to analyze massive amounts of data, big data particularly poses a problem in business analytics. As can be inferred, the above elicited characteristics of big data make it particularly hard for machine learning tasks to be carried out on it. Sampling such huge data is the first difficulty that is faced. The lack of structure (or poorly defined structure) is another hurdle while preprocessing the data. The performance of the algorithm also suffers because of the sheer volume of the data to be trained. Thus, an efficient platform with high computational prowess and the ability to handle huge sizes of data is required.

V. Current Machine Learning capable Cluster Computing Platforms and their Limitations

Since the 4 V's of big data, as described in the previous section are a hurdle to processing of data at a small scale, a high performance computing solution, or an alternative to high performance computing on a small or distributed scale has to be explored. There are platforms that have been in existence for a long time now, but not all of them currently support applying machine learning on big data, in an explicit and intuitive way, or tradeoff between performance and ease of use.

The key idea behind Hadoop is that instead of having a single juggernaut server that handles the computational and storage task of a very large dataset, Hadoop divides the whole task into a set of many subtasks, using the divide and conquer paradigm. After all the single tasks have been done, Hadoop is responsible for managing and recombining all the single subsets once their computation is over and the output is generated. In this case, it is possible to divide heavy computational tasks into many single node machines even if they are not so powerful, and obtain the results.

The simple programming model of Hadoop provided by the built in software library is basically a framework that enables distributed processing of large datasets across single clusters containing a few worker nodes (as shown in Figure 1), to clusters of computers containing several nodes each. Hadoop can take advantage of the storage and local computation offered by every node in the cluster, and can scale up from single servers to thousands of machines effortlessly.

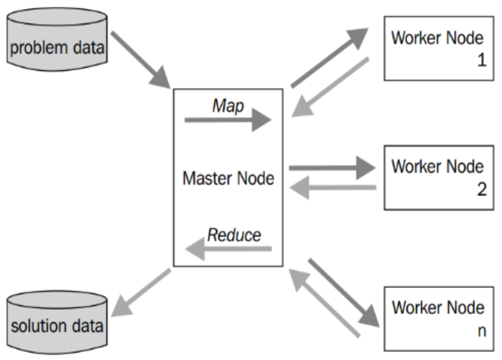

Figure 1. A high level abstraction of Hadoop's MapReduce paradigm.

Users who wished to exploit this great performance model offered by Hadoop to run machine learning tasks, used Apache Mahout, as it was tuned to Hadoop in a very efficient way. Apache Mahout [8][9], another Apache Software Foundation project, is a suite of open source implementations of scalable machine learning algorithms. The library contains algorithms primarily in the areas of classification, clustering and collaborative filtering. To enable parallelized operations, the implementations of the algorithms in Mahout use the Apache Hadoop platform. Like most of the projects in Apache Incubator, Mahout is a work in progress as various machine learning algorithms haven't yet been made available to users, even though the number of implemented algorithms has grown quickly.

Mahout fixes one of the major issues with Machine Learning techniques, which is scalability. Mahout can scale algorithms to large data sets. Since the algorithms implemented in Mahout have been written with Hadoop and MapReduce at their core, the core libraries of machine learning contain code that highly optimized to extract maximum performance out of the available nodes in the cluster. Currently Mahout supports mainly three use cases: collaborative filtering, clustering, and classification.

Even though Mahout on Hadoop are advantageous in many ways, there are some limitations [4][5]. Apache Mahout on Hadoop, although a great platform for data scientists, is not intuitive and easy to learn. The real-time and offline Hadoop backend are not integrated into one system. There exist some performance bottlenecks in the computation of item-item similarities, and finer control needs to be implemented over the sampling rate in most applications. Hadoop tends to convert the Job into a Batch Processing task. Also, since it is iterative in nature, just I/O and serialization of the data during Mapping (in MapReduce) can take up 90% of the processing time. The machine learning task itself runs for only about 10% - 15% of the actual running time. Also, there is no real-time data analysis or data stream analysis for dynamic machine learning applications. This called for development of and even more powerful and fast computing platform, that could take the best of Hadoop's MapReduce, but implement it in a much more optimized and efficient way.

VI. The Apache Spark platform

Apache Spark[11] was an incubator project, and gained a lot of attention from the data science community, regardless of its incubation status. Apache Spark is now a fully supported Apache product, and is out of its incubation status. Apache Spark is an open source computing engine evolved from Hadoop, and built from the ground up to deliver speed, ease of use, and sophisticated analytics as a powerful platform for the computing community

The component of prime interest is MLLib, the Machine Learning library for Apache Spark. It features highly optimized implementations of machine learning algorithms in Scala, and written from the base up to handle big data effectivelySpark give users access to a well-designed library of parallel and scalable machine learning algorithms. MLLib contains high-quality scalable machine learning algorithms as well as unbelievable speed that out performs MapReduce and many other machine learning libraries available publically. Since it is a component of Spark, it is usable through not only Scala, but Python and Java as well. MLlib is a Spark subproject providing machine learning primitives, relevant to mainly classification, regression, clustering, collaborative filtering and gradient descent. Algorithms under each category are:

- classification: logistic regression, linear support vector machine(SVM), naive Bayes
- regression: generalized linear regression (GLM)
- collaborative filtering: alternating least squares (ALS)
- clustering: k-means
- decomposition: singular value decomposition (SVD), principal component analysis (PCA)

A. *Experimental Setup*

The setup of Spark is fairly simple [12], and it is recommended that the pre-built binaries be download from the Spark website. The results obtained for this paper were collected by running the program on Spark version 0.9.1, when it was still in the incubation state. No substantial changes were made in the MLLib library, so the results obtained using Spark 0.9.1 will be identical to those possible with version Spark 1.0. A Spark cluster was deployed using Cloud Services on Microsoft Azure, and Linux VMs were used as the cluster nodes. Each machine had 4 core processors, with 14GB of memory each. Since the VMs had to be connected to each other in the cluster, a Virtual Network was setup, with RSA secured SSH.

VII. Clustering Geo-Spatial Data using the k-means Clustering implementation of MLLib

Most clustering methods used today either use k-means in conjunction with other clustering techniques, or they modify the algorithm in terms of sampling or partitioning. Given the number of clusters to be formed ‘k’, and ‘n’ data points in the data set, the goal is to choose k centers so as to maximize the similarity between each point and its closest center. The similarity measure most commonly used is the total squared distance between the point and the mean. This algorithm, also called the Lloyd’s algorithm first initializes k arbitrary “centers” from the data points, typically chosen at random, but using a uniform distribution. Each point is then assigned to the cluster whose center it is nearest to. After this, the centers are re-evaluated, keeping in mind the centers of mass of the points that surround the current center. Until the centers stabilize, the last 2 steps are repeated.

Thus, it can be considered to be one of the simplest unsupervised learning algorithms that can be used to find a definite clustering of a given set of points, even with varied data types. The objective function that this algorithm aims to minimize, is the squared error function. The objective function is given as below:

$$J = \sum_{j=1}^{k}\sum_{i=1}^{n}\left\|x_i^{(j)} - c_j\right\|^2$$

Here J is a chosen distance measure between a data point and the cluster center, and thus J is an indicator of the distance of the n data points from their respective cluster centers.

Since there are only a limited number of clustering ways that are possible, this algorithm will always give a definite result, and will always terminate. Also, users who go for the k-means algorithm are interested not in the accuracy of the result it produces, but the simplicity and speed with which it gives the clustering result. It does sometimes generate arbitrarily bad clustering, but the fact that it doesn’t rely on how the starting dummy cluster centers were placed with respect to each other makes it a good option when performing clustering tasks. In particular, it can hold with high probability even if the centers are chosen uniformly at random from the data points. The area in which k-means can be improved considerably is the way the initial centers are chosen. If this process is optimized, the algorithm can be considered to be more computationally sound, and overall a good option to go for. In the next section, we look at 2 of the best improvements made to the algorithm to date, both of which are used in the clustering library of Spark.

A. *The k-means++ and k-means|| algorithms*

As discussed earlier, k-means is relatively not a good clustering algorithm [13] if the quality of clustering or the computational efficiency is considered. Analysis shows that the running time complexity of k-means is exponential in the worst case scenario. K-means aims at locally optimizing the clusters by minimizing distance to the center of the clusters, and thus the results can possibly deviate from the actual globally optimal solution to a considerable extent. Although repeated random initializations can be used to tweak the results a little bit, they prove to be not so effective in improving the results in any way. In spite of all these shortcomings, there are a meagre number of algorithms that can match the simplicity of and speed of the k-means algorithm. Therefore, recent research has focused on optimizing and tweaking how the centers are initialized in the first step. If the initialization method is improved, the performance of the algorithm can be vastly sped up, both in terms of convergence and quality. One of the procedures to improve the initialization is k-means++.

The k-means++ algorithm makes a small change in the original initialization, by choosing just the first mean (center) at random, uniformly from the data. It also takes into consideration the contribution of a center to the overall error, and each center chosen by the k-means++ algorithm is selected with a probability that is proportional to this contribution. Thus, intuitively, k-means++ exploits the relatively high spread out of a good clustering. The new cluster centers chosen by k-means++ are thus the ones that are preferably further away from the previously selected centers. After analysis, it has been shown that k-means++ initialization improves the original algorithm by serving a constant approximation (O(log k) in some cases, when the data is difficult to cluster) of the optimum solution, if the data is known to be well cluster-able. The evaluation of the practical execution of the k-means++ algorithm and its variants is critical if performance of an actual running implementation is to be optimized. Tests demonstrated that correctly initializing the original k-means algorithm did lead to crucial improvements and lead to a good clustering solution. The k-means++ initialization obtained order of magnitude improvements, using various data sets, when the random initialization was put into effect.

However, its inherent sequential structure is one downside of the k-means++ initialization. Although when looking for a k-clustering of n points in the data set, its total running time is the same as that of a single K-Means iteration, it is not easily parallelizable. The probability with which a point is chosen to be the ith center depends critically on the realization of the previous i-1 centers (it is the previous choices that determine which points are away in the current solution).

A simple bare bones implementation of k-means++ initialization makes k iterations through the data in order to select the initial k centers. This fact is augmented and made clear when big data is brought into picture. As datasets become bigger, as in the case of big data, so does the number of partitions into which the data can be divided. For example, a typical cluster number k = 100 or 1000 is chosen to cluster, say clustering millions of points. But in this case, k-means++ being sequential in nature, proves to be very inefficient and slow. This slowdown is even more noticeable and unfavorable when the rest of the algorithm, i.e. the actual k-means algorithm can be parallelized to run in a parallel environment like MapReduce. For many applications, an initialization algorithm is desirable that guarantees efficient parallelizability, while providing the same or similar optimality to k-means++.

To make k-means++ even better, and to formulate a parallel implementation, Bahmani et al. developed k-means||. the k-means|| algorithm, instead of sampling a single point in each iteration, samples O(k) points and repeat the process for approximately O(log n) rounds. These O(k log(n)) points are then re-clustered into k initial centers for the original k-means. This initialization algorithm, which we call k-means||, is quite simple and lends itself to easy parallel implementations.

B. *Description and pre-processing of the dataset*

3D Road Network (North Jutland, Denmark) Data Set is essentially geo-coordinates of a road network in North Jutland (spanning over 185x135 sq. km), which has been augmented by adding the altitude (elevation information) of those geo-coordinates to the data set[3]. The Laser Scan Point Cloud Elevation technology was used to achieve this. This 3D road network was eventually used for benchmarking various fuel and CO2 estimation algorithms. For the data mining and machine learning community, this dataset can be used as 'ground-truth' validation in spatial mining techniques and satellite image processing.

Attribute Information:

1. OSM_ID: OpenStreetMap ID for each road segment or edge in the graph.
2. LONGITUDE: Web Mercaptor (Google format) longitude
3. LATITUDE: Web Mercaptor (Google format) latitude
4. ALTITUDE: Height in meters.

Since the first attribute is not significant in clustering the points, only the other 3 relevant attributes had to be extracted for the actual clustering step. The data set file was loaded into GNU Octave, and extraction was achieved by initializing a matrix of dimensions 434874X4 and then slicing off the first attribute using the built in slicing implementation of Octave. The resulting matrix was a 434874X3 matrix, which was then written to disk as a TXT file. This file was then used in the next step, which is dividing the data into training and test data sets.

The next step to preparing the data for training the K-Means model was to sample the data into a training data set, and a test data set. Different proportions of test and train data were tested - 40% of training data and 60% of test data, 50% of training data and 50% of test data, 60% of training data and 40% of test data, 70% of training data and 30% of test data. The best results were found in the last sample, as a good and robust model was built. At the end of pre-processing two files were created, train_70.txt (304412 records) and test_30.txt (134062 records).

C. *Explanation of the program*

In the program, we use the KMeans object of the MLLib library to cluster the data into clusters. The number of desired clusters is passed to the algorithm, which after performing numerous rounds of clustering, computes the Within Set Sum of Squared Error (WSSSE). WSSSE is the sum of the squared distance between each point in the cluster and the center of the cluster, and is used as a measure of variation within a cluster. You can reduce this error measure by increasing k. In fact the optimal k is usually one where there is an “elbow” in the WSSSE graph.

The parameters accepted by the train() method of the KMeans object are –

i. Data: The training data in the form of and RDD (Resilient Distributed Dataset) is fed into the train method, which will be iterated through to build the KMeans model.
ii. No. of clusters: specifies the number of clusters that the data is to be partitioned into.
iii. Max iterations: maximum number of iterations of the initialization algorithm (random, k-means++ or k-means||) is to be run.
iv. No. of runs: number of times the k-means algorithm has to be run, and this is a crucial parameter as k-means does not guarantee a globally optimal solution.

Increasing the number of runs can give some surety that the best clustering result would be obtained.

v. Initialization mode: initializationMode specifies either random initialization or initialization via k-means||.

VIII. Test Cases and Analysis of results

The test cases were formulated in a way that could help analyze how the implementation of the clustering algorithms included with the MLLib library of Apache Spark performed with a relatively dense, yet simple data set. The data set used, due to its spatial nature is inherently suitable for clustering. Yet, the points that have been recorded as part of the 3D Road Network, are at really close proximity of each other, and thus the data is very dense. The data, being dense, is a challenge for k-means as k-means goes for a partitional approach rather than a density based clustering approach. This would lead to understandable errors in clustering, and that would be an interesting point to observe. Also, since there are 434874 lines containing 3 floating point numbers each, performance of the algorithms with respect to the parameters specified for the clustering would be a crucial point to observe.

The test cases were designed to range from less computationally intensive to highly computationally intensive tasks. The tests cases have been described below –

i. Cluster count k = 10, maxIterations = 10, Runs = 10

A relatively low number of clusters specified guarantees that the algorithm will take a short amount of time to run. Also, because the runs are limited to 10, the algorithm will produce a high error of clustering. Since this is the first test case, it serves to be a placeholder for designing the next few test cases.

ii. Cluster count k = 20, maxIterations = 50, Runs = 100

Increasing the cluster count guarantees a lower WSSE, but since the number of maxIterations have been increased, along with the number of runs, it will be interesting to note the effect this change in parameters has on the performance as well as running time of the clustering.

iii. Cluster count k = 30, maxIterations = 15, Runs = 50

iv. Cluster count k = 40, maxIterations = 15, Runs = 50

v. Cluster count k = 50, maxIterations = 15, Runs = 50

vi. Cluster count k = 100, maxIterations = 15, Runs = 50

The above 4 runs simply increase the number of clusters, and this is done to observe trends in performance when only the cluster count is increased.

The results obtained exhibited interesting patterns, and helped infer that performance of the clustering is directly linked to the cluster count parameter. The legend is a triple, (k,m,r) which stands for cluster count k, maxIterations m and runs r. The results were measured in seconds, and since the magnitude of the results obtained when changing the no. of slave nodes ranged from 100s of seconds to 1000s, the results had to be normalized to have a clearer and more intuitive insight into the patterns in performance. The normalization was carried out using the z-score method, which transforms data into a range of [-2,2]. It uses the standard deviation and mean of the data to calculate the values. Also, this method proves useful to easily identify outliers, as any value that has a z-score > 2 or z-score < -2 doesn't fall in the normal range of the data being normalized. After z-score normalization, the runtime in seconds was plotted against the number of slave nodes (Worker nodes) being used by the algorithm. The resulting graph is shown in the following figure.

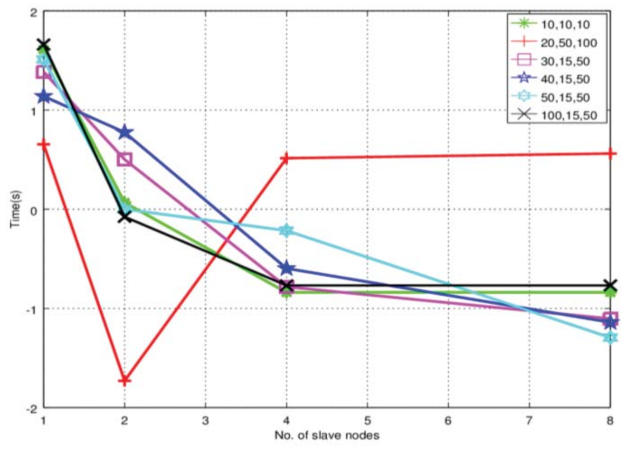

Figure 2. Clustering time vs. Number of Slave Nodes

As can be seen, in the first case, the time to cluster data decreases as number of slave nodes are increased. The performance doesn't change much when the number of slave nodes is increased from 4 to 8, as most of the slaves are scheduled randomly, and the rest remain idle while the jobs are running on the other nodes.

In the 2nd case, the max iterations and runs are increased, and the unnecessary stress on the computation is apparent. This case completely stands out from the rest of the cases as time complexity shoots up due to the relatively more extreme parameters. The 100 runs take longer on 4 and 8 slave nodes, which is unexpected according to the trend. This could be explained by arguing that scheduling and distribution process would be easier on 2 slaves as compared to that on 4 and 8 slaves, and more so when there is just one file being operated upon. This case helps infer that the number of runs increases the complexity and causes unnecessary fluctuations in running time, when accuracy of the model is not favorable over the speed (as in the case of big data). So, in further cases the runs are reduced to 50, and max iterations reduced to 15, as it was observed that the k-means++ converged in not more than 15 iterations every time.

In the consecutive cases, only the cluster count was increased by 10 with each case, and the number of slave nodes were varied as before. The trend remained the same across the last 4 cases – the running time decreased, with run times almost the same in the case of 4 and 8 slave nodes. This is due to idle states of the nodes when they're not needed, mostly in the case of the 8 slave nodes.

The result of clustering is however more understandable in terms of the Average WSSE (Within Set Squared Errors) which dropped considerably across all 6 cases. This is attributed solely to the number of clusters being created, and has no relation with the other parameters of the KMeans model. As the number of clusters are increased, the WSSE decreases. Here, the values plotted are the average of the WSSE calculated in each case where the number of slave nodes was calculated.

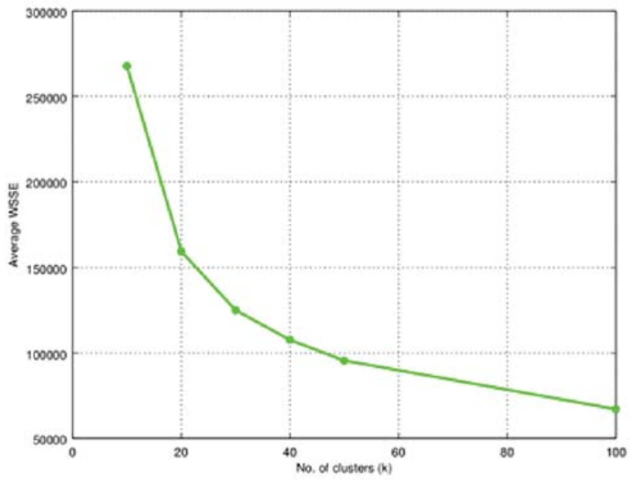

Figure 3. Average WSSE vs. Number of clusters

IX. Conclusion

The focus of this paper was to explore platforms that can be used to implement intelligent applications that harness the power of cluster computing (on local machines as well as the cloud) and apply machine learning on big data. Current cluster computing platforms like Google Cloud Engine and Apache Hadoop, and their corresponding machine learning libraries – Prediction API, and Apache Mahout were studied, and compared against Apache Spark and MLLib.

A cluster was created on Windows Azure, and each node in the cluster had a quad core processor with 14 GB of RAM, running Ubuntu Server 12.04 LTS. Apache Spark was downloaded and built on each machine. The program was written in Python, and interfaced with Apache Spark using Pyspark. A simple clustering task was run on a relatively large and complex data set, and the run times were recorded. Varying the configuration of the cluster with every run showed some interesting trends in the results. As compared to traditional iterative implementations of k-means clustering, running it on Apache Spark on a cloud cluster definitely gave it an advantage on run time.

With the rise of diverse, flexible and economical cloud service, users from both research and business backgrounds can harness the power of Spark on a cloud cluster, and apply data mining and machine learning concepts to their everyday tasks. It is even more suited for big data, as Spark features excellent parallelization of data, and optimized code libraries so that jobs can be processed quickly. Big data and machine learning are essentially a very good combination of areas to work upon, and research carried out in these areas are definitely going to influence the development of intelligent and computationally powerful platforms for the ever growing domain of Big Data.

References

[1] NG DATA, *"Machine learning and Big Data analytics: the perfect marriage",* Internet: http://www.ngdata.com/machine-learning-and-big-data-analytics-the-perfect-marriage/

[2] Daniel Gillblad, Doctoral Thesis, Swedish Institute of Computer Science, SE–164 29 Kista, Sweden, 2008, "*On practical machine learning and data analysis*", Internet: http://soda.swedish-ict.se/3535/1/thesis-kth.pdf

[3] 3D Road Network (North Jutland, Denmark) Data Set, UCI Machine Learning Repository, Internet: http://archive.ics.uci.edu/ml/datasets/3D+Road+Network+%28North+Jutland%2C+Denmark%29

[4] Sean Owen, Contributor at Quora**, "***What are the pros/cons of using Apache Mahout when creating a real time recommender system?***",** Internet: http://www.quora.com/Apache-Mahout/What-are-the-pros-cons-of-using-Apache-Mahout-when-creating-a-real-time-recommender-system

[5] Nick Wilson, BigML, "Machine Learning Throwdown", Internet: http://blog.bigml.com/2012/08/02/machine-learning-throwdown-part-1-introduction/

[6] Georgios Paliouras, Department of Computer Science, University of Manchester, Thesis on "*Scalability of Machine Learning Algorithms",* Internet: http://users.iit.demokritos.gr/~paliourg/papers/MSc.pdf

[7] Tom M.Mitchell, School of Compter Science, Carnegie Mellon Univesity, Pittsburgh, July 2006, "*The Discipline of Machine Learning",* Internet: http://www.cs.cmu.edu/~tom/pubs/MachineLearning.pdf

[8] Gaston Hillar, "*Machine Learning with Apache Mahout: The Lay of the Land*", Internet: http://www.drdobbs.com/open-source/machine-learning-with-apache-mahout-the/240163272

[9] Apache Mahout, Apache Foundation, Internet: https://mahout.apache.org/

[10] IBM, Articles on Big Data, Internet: http://www.ibm.com/developerworks/bigdata/

[11] Apache Spark, Apache Foundation, Internet: http://spark.apache.org/

[12] Mbonaci, "Spark standalone cluster tutorial", Internet : http://mbonaci.github.io/mbo-spark/

[13] Songma, S.; Chimphlee, W.; Maichalernnukul, K.; Sanguansat, P., "Classification via k-means clustering and distance-based outlier detection," *ICT and Knowledge Engineering (ICT & Knowledge Engineering), 2012 10th International Conference on* , vol., no., pp.125,128, 21-23 Nov. 2012