Using Hadoop and Spark for Big Data Analytics: Predicting Readmission of Diabetic patients

KUMAR SATYAM[1,*], PIYUSH SHINDE[1,**], AND SRIKANTH RAMANAM[1,***]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: ksatyam@indiana.edu
[**] Corresponding authors: pshinde@iu.edu
[***] Corresponding authors: srikrama@iu.edu

project-004, May 6, 2017

This project proposes and demonstrates the use of Hadoop and Spark on cloud to run predictive analytics using machine learning on large amount of data. Our case study is to predict the readmission likelihood for diabetes patients using their available medical history.

Keywords: Hadoop, Spark, MLlib, Ansible, Cloudmesh Client, Predictive Analysis

Report: https://github.com/cloudmesh/sp17-i524/tree/master/project/S17-IR-P004/report
Code: https://github.com/cloudmesh/diabetic

CONTENTS

1. INTRODUCTION

The idea behind this project is to introduce Hadoop-Spark distributed cluster for building predictive analytics applications. Spark allows us to build applications that are scalable and faster when compared to standalone applications performing similar analytics tasks.

We chose the case study of predicting the likelihood of a diabetes patient getting readmitted within 30 days from the date of discharge using his/her available medical data. Predictive analytics based on medical data to provide healthcare is an active area of research.Predicting readmission likelihood is aimed to providing better care to patients. Readmissions happen due to deterioration of patients' health. Readmission predictions help doctors and patients to take preventive steps to avert medical emergencies that need hospitalization.

We approached this problem as a classification problem to classify the patients into 'Yes' or 'No' classes, indicating whether the patient is likely to be readmitted or not in the next 30 days. We used different machine learning algorithms on the available data, after some pre-processing, to predict the same. The accuracy percentages obtained for all the utilized classification algorithms are included in the report.

Our other important goal is to propose an end-to-end solution that is scalable and faster than a standalone predictive analytics application. While our dataset has about 100,000 records, anticipating real-world scenarios with huge amounts of data we are proposing a Hadoop based solution. We are utilizing Spark for

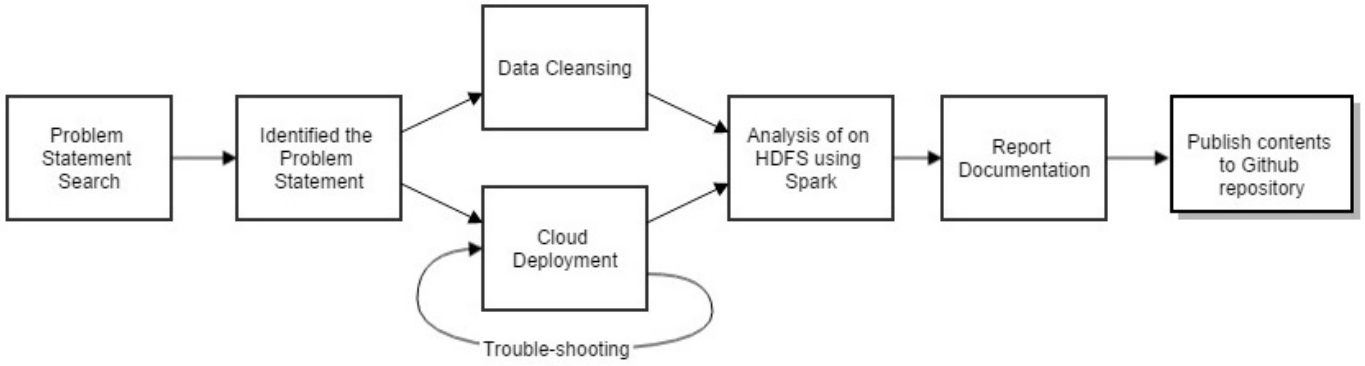

Fig. 1. Workflow of the project

its faster processing [1] and advanced analytics through packages like MLlib [2], which provides several commonly used machine learning algorithms. Finally we are implementing this solution over cloud infrastructure to meet our infrastructure requirements. This helped us demonstrate an end-to-end solution closer to real-world scenarios, where enterprises utilize cloud infrastructure in a pay per-use model. This helped us save time and resources in setting up the infrastructure.

We deployed our solution on two different clouds and obtained metrics to assess the infrastructure performance with our solution. We also deployed our solutions on a distributed Hadoop/Spark environment built on on-premise machines. We also automated the deployment of Spark cluster on cloud virtual machines using Ansible. We compared the performance metrics of all the three infrastructure choices, with respect to our application and included them in this report. Our project workflow can be seen in the Figure 1.

2. ARCHITECTURE

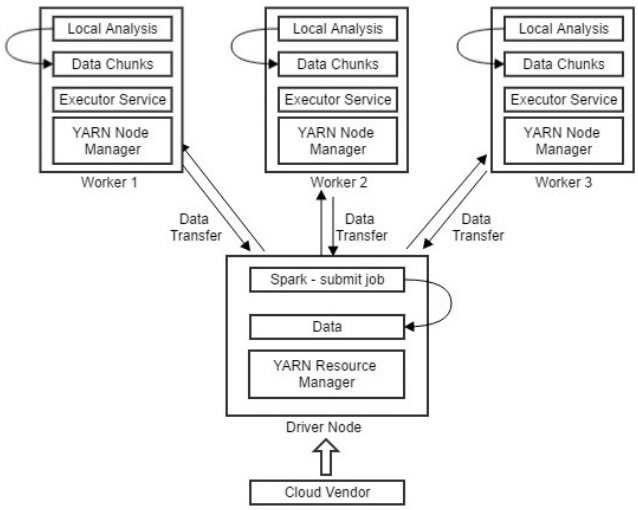

Fig. 2. Architecture Diagram

Figure 2 gives an overview of our solution's architecture. We deployed a spark driver node and three worker nodes. A driver node is a node that runs the driver program. It declares the transformations and actions on RDDs (Resilient Distributed Datasets) of data and submits such requests to the master [3]. In practical terms, the driver is the program that creates the Spark Context [4], connecting to a given Spark Master. It is a node where the yarn resource manager resides. A worker node is a node, which executes the program that involves individual data analysis task. Running the spark-submit script from the master node starts the spark job. It divides the data into data chunks and transfers them to individual worker nodes. Then a processing task is performed on the data chunks on the individual worker nodes. The processed data and analytics results are then written back to the HDFS file system as needed.

We then deployed Hadoop and Spark on these machines to setup a HDFS Spark cluster on our cloud and on-premise machines. We stored our dataset on the Hadoop's HDFS file system. Finally, we ran our predictive analytics application, which utilizes Mllib, on Spark cluster by launching it using spark-submit.

3. TECHNOLOGIES

Table 1. List of technologies used

Technology	*Usage*
Hadoop[5]/**Spark** [6]	Big Data Technologies
Python[7]	Development
MLlib[2]/**scikit-learn**[8]	Machine Learning Library
GitHub[9]	Project Repository
Ansible[10]	Application Deployment & Configuration Management
Chameleon[11], **JetStream**[12], **VirtualBox**[13]	Benchmarking
LaTex [14]	Document Preparation

We used specific technologies for specific tasks in this project as listed in Table 1:

- **Hadoop**: Apache Hadoop is a framework for processing and storing huge amounts of data, commonly known as 'Big

Data', in a distributed applications. It allows users to build scalable and highly available data applications. Hadoop has four modules: HDFS, YARN, MapReduce and Hadoop Common. Hadoop Distributed File System (HDFS) allows users to store large amounts of data. YARN is the framework for job scheduling and resource management. MapReduce supports parallel processing of large data sets stored in the distributed environment through HDFS. Hadoop Common provides utilities that support other Hadoop modules.

- **Spark**: Spark runs on Hadoop and provides faster data processing capabilities for data on HDFS. It primarily uses a new data structure called Resilient Distributed Dataset (RDD) for processing. RDD is a read only multiset of data items distributed over cluster of virtual machines. Spark also has a new feature of fault tolerance and in the event of a primary master node failure, the secondary master takes over. Spark, unlike Hadoop applications, allows the iterative reading and writing in-memory. After processing it writes the data to HDFS.

- **Python**: We chose python as per our programming language. Python is one of the programming languages supported by Spark API through pyspark. We used it because of its simple syntax and data manipulation capabilities.

- **scikit-learn**: It is an open source Python library that provides Machine Learning algorithms and other utilities to preprocess and visualize data [8].

- **MLlib**: Spark MLlib is the Spark's machine learning library provides machine learning algorithms that can be applied on Resilient Distributed Datasets [15]. It also provides other data manipulation utilities. MLlib has API available in Java, Python, Scala and R [2].

- **GitHub**: GitHub is 'a web-based Git or version control repository and Internet hosting service' [16]. We used Github repositories to store all the files related to documentation, Ansible scripts and python code.

- **Ansible**: We are using Ansible for automating deployment of our software on cloud. We used Ansible scripts to automate deployment of cloudmesh client along with its prerequisites like pip, virtualenv etc. We also used Ansible for automating deployment of Hadoop and Spark.

4. CLOUD INFRASTRUCTURE

We have setup the required infrastructure by provisioning virtual machines on two cloud vendors, Chameleon, Jetstream and our on-premise machines.

4.1. Chameleon Cloud

Chameleon is a collaborative cloud service primarily meant for research community. It allows users to explore problems ranging from the creation of Software as a Service to kernel support for virtualization. It is a good example of IAAS loaded with software defined networking and optimized virtualization technologies. We created three virtual machines on this cloud. One for Master node and two for worker nodes of Spark.

4.2. Jetstream Cloud

Jetstream is a cloud service which aims to provide researchers Jetstream's development was led by Indiana University's Pervasive Technology Institute (PTI) in collaboration with other universities [12] across the United States. This cloud service was used to provision the necessary virtual machines. We created three virtual machines on Jetstream for Spark cluster nodes.

4.3. Virtual Box

It is a fully virtualized hypervisor which gives the ability to spawn virtual machines in local commodity hardware. In fully virtualized environment, the guest OS is not aware of the underlying resources on which it is running as the hypervisor creates a complete simulation of the underlying hardware.

A brief comparison of multiple attributes of the clouds used are displayed in Table 2.

Table 2. Comparison of cloud vendors

Clouds	*Chameleon*	*Jetstream*	*VirtualBox*
CPU	Intel Xeon X5550	Dual Intel E-2680v3	Intel Core i5-6200U
RAM	2 GB	2 GB	2 GB
Number of CPU's	1	1	1
CPU Cores	1	1	2
CPU Speed	2.3 GHz	2.3 GHz	2.3 GHz

5. AUTOMATED DEPLOYMENT: ANSIBLE

The Ansible playbooks allow us to install and enable several packages in different VMs simultaneously. This removes the overhead of the logging into individual machine to install different packages and services.

For this project, we used Ansible to automate the deployment of spark and its prequisites. The Ansible script is written such that we can leverage the cloudmesh client technology to deploy the spark cluster. The steps involved in the script can be seen in Figure 3.

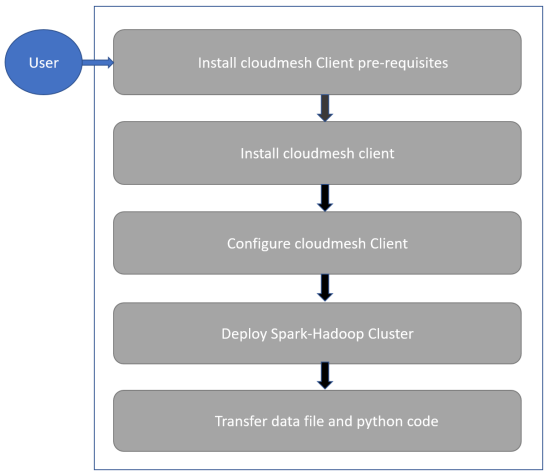

Fig. 3. Automated Cloud Deployment using Ansible

We used the below listed Ansible playbooks for our project:

1. *inventory*: This file is used by the playbook to install packages and services on target servers. This Inventory file consists of target servers and these servers can be categorized into groups like webservers, dbservers etc. Under each group, we can list the server which serves common purpose.

2. *playbook-cloudmesh-first-time-install.yml*: This file was used for deployment of cloudmesh client. It installs all the prerequisite of cloudmesh client first, checks if the ssh keys are already present, and then deploys the cloudmesh client.

3. *cloudmesh-first-time-configure.yml* : It is used to configure the cloudmesh after installation. The main tasks for this playbook include configuring cloud provider name and cloud user name, adding the ssh-key to the cloudmesh database, and upload the ssh key to the cloud.

4. *deployment-playbook.yml:* It is used to deploy the hadoop-spark cluster in chameleon and jetstream clouds. The main tasks include defining a cluster, defining a hadoop-spark stack on the cluster, syncing the clustering with github and deploying the hadoop cluster.

5. *tranfer-files-remotely.yml:* It is used to transfer pyspark codes and input files from local machine to report machines. The main tasks include reset of cloudmesh database, upgrade the python-pip, uninstall and install of cloudmesh client.

6. *upgrade-cloudmesh.yml:* It is used only when there is a need to upgrade.

We developed Ansible playbook for installation and configuration of cloudmesh client. We also developed playbook to deploy hadoop/spark cluster on different clouds. After successful deployment of Hadoop-Spark/hadoop cluster we require the python modules that perform data analysis and the data files in spark master node. For this, we developed an Ansible playbook to automate the manual process of uploading the file, authentication and downloading the file. We invoked the Ansible scripts through shell script, running on local Virtual Machines.

6. DATA CLEANING AND PRE-PROCESSING

We approached the problem from a pure data perspective to address the challenge of lacking medical domain knowledge. For some basic information, we relied on the dataset description on UCI website [17], ICD-9 [18] and earlier studies [19]. The initial data set is publicly available on the UCI Machine Learning Repository [17]. The initial data set has information extracted from the database satisfying the following criteria [19]:

- Each row corresponds to an inpatient encounter (a hospital admission).
- All of the encounters are "diabetic" encounters, that is, one during which any kind of diabetes was entered to the system as a diagnosis.
- Each encounter also corresponds to a patient stay between 1 and 14 days.
- Laboratory tests were performed during the encounter.
- Medications were administered during the encounter.

101,766 encounters were present in the data set that satisfy the above five inclusion criteria. Each encounter consists of 55 features describing the diabetic encounters, including demographics, diagnoses, diabetic medications, number of visits in the year preceding the encounter, and payer information. We defined the readmission field with 2 values: "YES," for cases where the patient was readmitted within 30 days of discharge and "NO," for both readmission after 30 days scenario and no readmission at all.

Diagnosis 1, 2 and 3 had many categorical values in the form of ICD-9 codes and had missing values. These ICD-9 codes were sorted and grouped into 9 categories, namely Circulatory, Respiratory, Digestive, Diabetes, Injury, Musculoskeletal, Genitourinary, Neoplasms and Other based on the ICD-9 codes [19]. The missing values were assigned the group 'Other'.

This data set had several features with empty fields. So we removed the features missing high percentages of data as they affect our analysis. The removed features were weight, medical specialty and payer code. The race attribute had 2% missing values which were filled by the mode value 'Caucasian'.

Observations only with unique patient ids were considered, excluding those with discharge disposition corresponding to the patient's death.

We filtered our data set according to the above-mentioned constraints and retained 62,937 encounters each corresponding to an unique patient and 55 features describing such encounter. We prepared three data-sets to test the multiple machine learning algorithms (Stochastic Gradient Descent, Gaussian Naïve Bayes, K-means and Decision Tree). Finally we also removed the patient id and encounter id as they were not relevant to learning algorithms.

We used one hot encoding to convert the categorical data features to numerical data. This creates new dummy features to represent the categorical data in numerical format.

The first data set was prepared using one hot encoding on the original data (with 62,937 observations), that resulted in 136 features representing the original 55 features.

For our second data set we implemented feature selection using a Variance Threshold algorithm that removes all low-variance features [20]. We set a variance threshold of '0.8'.

The data set B was formed using this algorithm, which helped extract 26 features.

The original data contains the age attribute grouped in 10-year intervals from 0 to 100 years. For the third data set we grouped the 10 intervals to 3 intervals by combining the age groups younger than 30, 30-60 and older than 60 years. One-hot encoding was then applied to form the third data set. This data set contained 129 features.

7. PRELIMINARY DATA ANALYSIS

The unit of our analysis is an encounter; to keep the observations independent, we only analyzed one encounter per patient. We performed early data analysis in python in a local machine. We implemented four classification algorithms using scikit-learn library on each of the 3 data sets created. The data sets were first divided in training and testing set. The training set had about 80% observations (5000 observations approx.) whereas the testing set had the remaining 20% observations (12937 observations).

Each of the algorithms provided by scikit-learn were used following in the manner:

Table 3. Results from scikit-learn

Classification Technique	*Number of Features*	*Accuracy (%)*
SGDClassifier[21]	136	90.68
	26	86.31
	129	86.96
GaussianNB[22]	136	10.43
	26	88.30
	129	9.96
KMeans[23]	136	55.26
	26	55.24
	129	55.26
DecisionTreeClassifier[24]	136	83.35
	26	82.50
	129	83.28

1. Create and fit a model using the observations and readmission of the training set.
2. Predict labels of the testing set.
3. Calculate the accuracy using the predicted labels and true labels of the testing set. The parameters needed in implementing the above-mentioned algorithms were set so as to be valid to our data, give optimum results and make the results to be reproducible.

KMeans clustering gave us approximately 55% accuracy with all the three data sets. Though it is an unsupervised learning algorithm, we used it to examine if the clustering divided the data into readmission classes, Yes and No, to an acceptable level. We concluded that clustering based on the Euclidean distance may not be the right approach for this classifying this data set. The accuracy percentages obtained for other classification algorithms can be found in the Table 3.

8. DATA ANALYSIS ON SPARK CLUSTER

We performed data analysis on Spark cluster using pyspark MLlib on data stored on HDFS.

8.1. Start the Spark service

Start the service of spark using the following command:

```
$SPARK_HOME/sbin/start-all.sh
```

Stop the service of spark using the following command:

```
$SPARK_HOME/sbin/stop-all.sh
```

Using the command 'jps' we get a list of the following services:

```
nodemanger
resourcemanger
master
namenode
applicationmaster
```

8.2. Data Storage

Uploading input data and code to Driver Node After the Spark setup is ready for the deployment, the data is pushed from the localhost to the remote spark master node. For this we used an Ansible script.

1. Ansible script

   (a) In the host files we set the target master(driver node) IP address as follows:

   ```
   [remotehosts]
   129.114.33.106 ansible_ssh_user=cc
   ```

   (b) Now , add the following entry in the yaml file to transfer the file to destination

   ```
   - hosts: remotehosts
     tasks:
       - name: Transfer file from local to
       satyam-001
         synchronize:
           src: /home/<username>/ansible
           script/ansible-spark/traindat.csv
           dest: /home/cc/
           mode: push
         delegate_to: 127.0.0.1
   ```

2. Using Github we uploaded the input data csv file and python execution code to a git repository. We installed git package in the spark driver node. We used 'wget' command with the repository path to download the data set to the virtual machine.

After the data is downloaded to the virtual cloud machine, we uploaded the file to HDFS through the following command.

```
Hdfs dfs  -put  <source file path>  <hdfs-folderpath>
```

This HDFS file serves as an input for our analytics application.

8.3. Launching Data Analysis Application

We used python programming language to develop an application that performs predictive analytics tasks. We leveraged pyspark.mllib library for machine learning algorithms.

We launched our application code using the following command

```
$ ./bin/spark-submit --class path.to.your.Class
--master yarn --deploy-mode cluster [options]
<app jar> [app options]
```

There are 4 main steps to each implementation of.

- Input formatting: MLlib classes expect RDD's of LabeledPoints. For this we parsed the data and converted each entry into a LabeledPoint , with label specifying the true output class.
- Next, the processed data frame is divided into train and test datasets.
- Train the model with the algorithm and training data
- After training, We used the model to predict the classes for test data and calculate the accuracy

Table 4. Results from Spark MLlib

Classification Technique	*Number of Features*	*Accuracy (%)*
Logistic Regression SGD[25]	136	90.88
	26	90.99
	129	90.88
Naive Bayes[26]	136	84.97
	26	85.02
	129	84.987
K-means[27]	136	55
	26	55
	129	55
Decision Tree[28]	136	91.01
	26	90.75
	129	90

9. RESULTS

Kmeans did not successfully separate the data class wise as observed in scikit-learn library, as shown in Table 4. We used supervised classification algorithms namely decision tree, logistic stochastic gradient descent and naïve bayes. We obtained good accuracies of 80-90% with classification algorithms.

Similar analysis in a real world scenario can be utilized to predict the readmission likelihood using medical records of diabetes patients. This enables doctors to pay special attention to those patients, identify the causal factors and give preventive care.

10. BENCHMARKING

We performed benchmarking to assess the performance with respect to four different aspects. (1) Spark deployment time in multiple clouds with variation in number of virtual machines. (2) Run-time of different algorithms on multiple clouds with variation in number of virtual machines. (3) File transfer time of two files of different size in multiple clouds. (4) Accuracy of multiple algorithms in Spark MLlib and scikit-learn.

10.1. Comparison of Spark Deployment Time

We compared the Hadoop-Spark deployment time with different number of machines on each of the clouds, Jetstream and Chameleon. In Table 5, we can see the Hadoop-Spark deployment times for clusters of different sizes on Chameleon and Jetstream clouds.

The variation of Hadoop-Spark deployment times on Chameleon and Jetstream clouds is shown in Figure 4. We observed that Spark deployment times on Jetstream are lesser than those observed on Chameleon, with the exception of cluster with 5 VMs. This aberration was caused due to a network issue, which was confirmed from the deployment logs.The log files recorded several unsuccessful ping requests to the cluster nodes before successful completion of deployment.

Table 5. Hadoop-Spark deployment time

Number of VMs	*Chameleon (seconds)*	*Jetstream (seconds)*
2	631.11	484.62
3	657.86	496.15
5	763.14	1399.54
7	1089	806.36

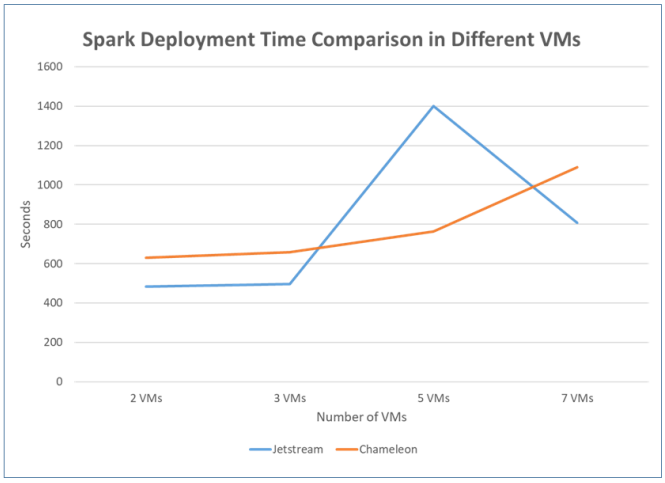

Fig. 4. Comparison of Spark deployment results

10.2. Computation time of algorithms in clouds

We can see in Table 6 the computation time required for running different Spark MLlib algorithms in multiple VMs in Chameleon and JetStream clouds.

Table 6. Computation time of algorithms in clouds in multiple VMs

Algorithms	*Number of VMs*	*Chameleon (seconds)*	*Jetstream (seconds)*
Logistic Regression SGD	1	91.67	115.55
	3	82.57	102.71
	5	31.38	57.23
Naive Bayes	1	66.09	58.48
	3	58.44	45.89
	5	23.2	36.47
Decision Tree	1	72.36	86.69
	3	60.62	71.54
	5	30.25	50.37

Figure 5 shows the performance of multiple machine learning algorithms provided by Spark MLlib on clusters in Chameleon, Jetstream and Virtual Box.

Figures 6 and 7 shows the run-time comparison for machine learning algorithms using Spark MLlib in Jetstream and Chameleon, for predicting readmission likelihood, for 3 and 5

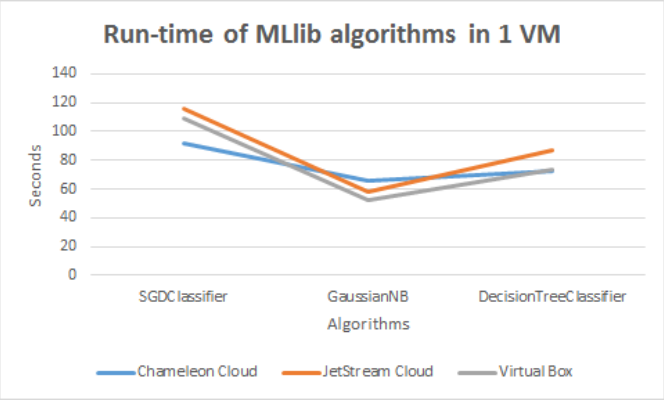

Fig. 5. Run-time in different clouds.

VMs. We can see that the run-time decreased with increase in the number of cluster nodes for all algorithms.

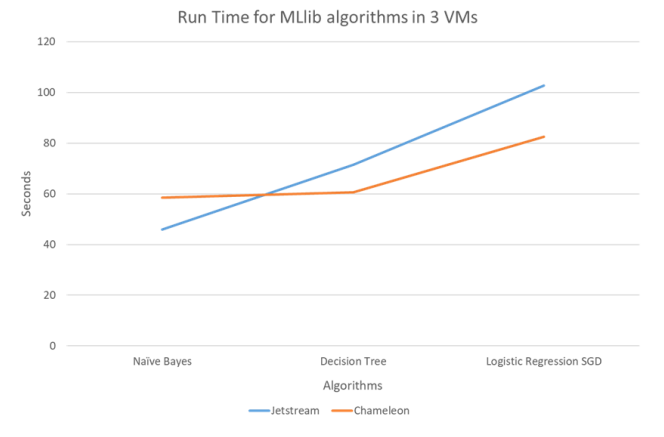

Fig. 6. Run-time in different clouds for 3 VMs

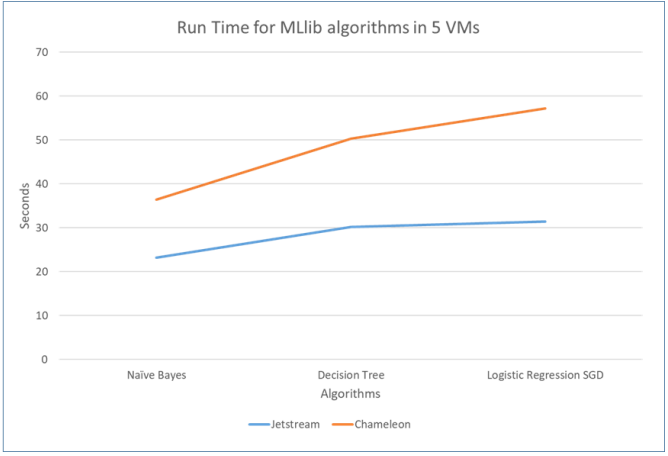

Fig. 7. Run-time in different clouds for 5 VMs

10.3. Accuracy comparison of multiple algorithms in Spark MLlib vs scikit-learn

We have recorded several metrics while running each of the four algorithms in different clouds. The four algorithms are SGD Classifier, Gaussian Naïve Bayes, kmeans and Decision Tree Classifier. The variations in run-time and deployment time are caused due to factors like:

1. *The complexity of the algorithm:* For example, kmeans clustering has O(n log n) time complexity which is worse than Gaussian Naïve Bayes and hence takes more time.

2. *The network latency between the hosts:* The VMs which are provisioned on clouds can be on different hosts spread across different racks of datacenters and even datacenters located in different geographies. This may result in network latency and affect the run-time of a program running in a distributed environment.

We compared the accuracies for different MLlib algorithms using pyspark.mllib package. We can see from Figure 8 the accuracies achieved for multiple algorithms in MLlib. While kmeans gave the lowest accuracy, rest of the algorithms gave similar higher accuracies.

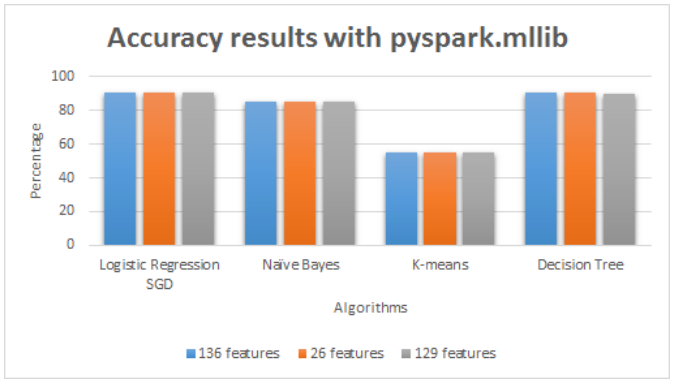

Fig. 8. Accuracy Results with pyspark.mllib

We compared the accuracies achieved from Spark MLlib and scikit-learn algorithms. Figure 9 shows accuracy results of multiple machine learning algorithms in scikit-learn vs MLlib.

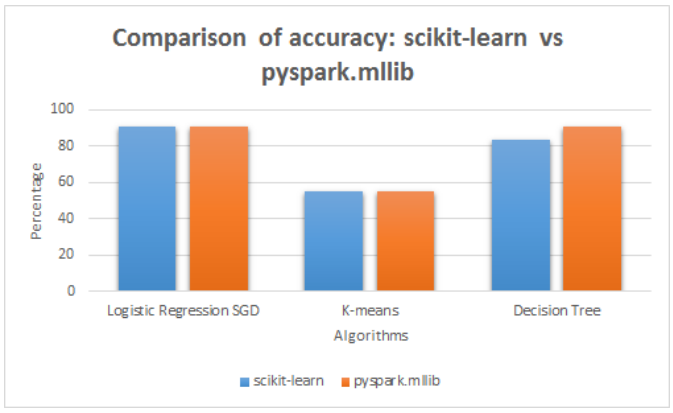

Fig. 9. Accuracy Results: scikit-learn vs pyspark.mllib

11. CONCLUSION

We demonstrated the implementation of a big data based predictive analytics solution using Hadoop, Spark and Spark's MLlib machine learning algorithms. We predicted the readmission

likelihood using MLlib with an accuracy of 90% by analyzing the medical data stored on HDFS. We also compared the results of Spark's MLlib [2] algorithms against scikit-learn's algorithms and found that both yielded similar results.

While Hadoop provided us a distributed file system for storing large amounts of data, Spark provided us big data processing capabilities and several libraries to accomplish several common processing and analytics tasks. We were successfully able to deploy the Spark-Hadoop cluster infrastructure on Chameleon, Jetstream and on-premise virtual box clouds. We developed several shell scripts and Ansible playbooks to automate the deployment. These technologies can be used to build similar solutions for real world scenarios requiring processing and analytics of big data. Our solution can also be extended to build analytics applications that process streams of data in real-time using Spark Streaming [29] and MLlib.

12. ACKNOWLEDGMENTS

This project was a part of the Big Data Software and Projects (INFO-I524) course. We would like to thank Professor Gregor von Laszewski and the associate instructors for their help and support during the course.

REFERENCES

[1] Databricks, "Apache Spark," Web Page, accessed: 2017-04-21. [Online]. Available: https://databricks.com/spark/about

[2] Apache, "Apache Spark MLlib," Web Page, accessed: 2017-04-21. [Online]. Available: http://spark.apache.org/mllib/

[3] Apache, "Class RDD<T>," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/rdd/RDD.html

[4] Apache, "Class RDD<T>," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/2.0.2/api/java/org/apache/spark/SparkContext.html

[5] Apache, "Welcome to Apache™ Hadoop®!" Web Page, accessed: 2017-03-12. [Online]. Available: http://hadoop.apache.org/

[6] Apache, "Apache Spark: Lightning-fast cluster computing," Web Page, accessed: 2017-03-12. [Online]. Available: http://spark.apache.org/

[7] P. S. Foundation, "python," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.python.org/

[8] D. Cournapeau, "scikit-learn," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/

[9] T. Preston-Werner, "GitHub," Web Page, accessed: 2017-04-21. [Online]. Available: https://github.com/

[10] R. Hat, "ANSIBLE," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.ansible.com/

[11] C. Cloud, "Chameleon," Web Page, accessed: 2017-04-21. [Online]. Available: https://www.chameleoncloud.org/

[12] I. University, "Jetstream," Web Page, accessed: 2017-04-21. [Online]. Available: https://jetstream-cloud.org/

[13] Oracle, "VirtualBox," Web Page, accessed: 2017-04-21. [Online]. Available: https://www.virtualbox.org/wiki/VirtualBox

[14] LATEX, "The LATEX Project," Web Page, accessed: 2017-03-12. [Online]. Available: https://www.latex-project.org/

[15] Apache, "Machine Learning Library (MLlib) Guide," Web Page, accessed: 2017-04-21. [Online]. Available: http://spark.apache.org/docs/latest/ml-guide.html

[16] W. Inc., "GitHub," Web Page, accessed: 2017-04-21. [Online]. Available: https://en.wikipedia.org/wiki/GitHub

[17] U. M. L. Repository, "Diabetes 130-US hospitals for years 1999-2008 Data Set," Web Page, accessed: 2017-03-12. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+1999-2008#

[18] U. D. of Health and H. S. (HHS), "ICD9Data.com," Web Page, accessed: 2017-04-21. [Online]. Available: http://www.icd9data.com/

[19] B. Strack, J. P. DeShazo, C. Gennings *et al.*, "Impact of hba1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records," *BioMed Research International*, vol. 2014, no. 781670, April 2014, published as an Article. [Online]. Available: https://www.hindawi.com/journals/bmri/2014/781670/

[20] scikit-learn developers, "sklearn.feature_selection.VarianceThreshold," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.feature$_$selection.VarianceThreshold.html

[21] scikit-learn developers, "sklearn.linear_model.SGDClassifier," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.linear$_$model.SGDClassifier.html

[22] scikit-learn developers, "sklearn.naive_bayes.GaussianNB," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.naive$_$bayes.GaussianNB.html

[23] scikit-learn developers, "sklearn.cluster.KMeans," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[24] scikit-learn developers, "sklearn.tree.DecisionTreeClassifier," Web Page, accessed: 2017-04-21. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[25] Apache, "pyspark.mllib package," Web Page, accessed: 2017-04-21. [Online]. Available: http://spark.apache.org/docs/2.0.0/api/python/pyspark.mllib.html#pyspark.mllib.classification.LogisticRegressionWithSGD

[26] Apache, "Naive Bayes - RDD-based API," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/2.1.0/mllib-naive-bayes.html

[27] Apache, "Clustering - RDD-based API," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/2.1.0/mllib-clustering.html#k-means

[28] Apache, "Decision Trees - RDD-based API," Web Page, accessed: 2017-04-21. [Online]. Available: https://spark.apache.org/docs/2.1.0/mllib-decision-tree.html

[29] Apache, "Spark streaming," Web Page, accessed: 2017-04-23. [Online]. Available: http://spark.apache.org/docs/latest/streaming-programming-guide.html

A. APPENDIX: WORK DISTRIBUTION

- Kumar Satyam [S17-IR-2031]
  - Developed Ansible playbook for installation of cloudmesh client.
  - Developed Ansible playbook for automated deployment of Spark on different clouds.
  - Deployed the initial test of Spark cluster on 1 virtual machine of Jetstream cloud.
  - Deployed a Spark cluster on 5 and 7 virtual machines of Chameleon cloud.
  - Developed Ansible script for file transfer from local VM to Spark master node on different clouds.
  - Executed K-means python script for predicting readmission and recorded the runtime on Spark cluster on 1 VM on Chameleon cloud.
  - Developed python code for predicting readmission using Spark MLlib algorithm Decision Tree.
  - Developed python code for predicting readmission using Spark MLlib algorithm K-means.
  - Tested python code of predicting readmission using Spark MLlib algorithms on a single VM spark cluster deployed on Virtual Box.

  - Executed Naive Bayes python script for predicting readmission and recorded the runtime on Spark cluster of 1, 3 and 5 VMs on chameleon cloud.
  - Executed K-means python script for predicting readmission and recorded the runtime on Spark cluster on 1 VM on Jetstream cloud.
  - Executed Decision Tree classifier python script for predicting readmission and recorded the runtime on Spark cluster on 1, 3 and 5 VMs on Jetstream cloud.
  - Benchmarked the algorithm runtimes of chameleon and Jetstream clouds.
  - Documented the sections: benchmarking, conclusion and references.

- Piyush Shinde [S17-IR-2035]
  - Performed preliminary data analysis.
  - Developed python code for preliminary analysis using sci-kit machine learning algorithm Gaussian Naive Bayes.
  - Developed python code for preliminary analysis using sci-kit machine learning algorithm SGD classifier.
  - Developed python code for preliminary analysis using sci-kit machine learning algorithm Decision Tree.
  - Developed python code for preliminary analysis using sci-kit machine learning algorithm K-means.
  - Developed Ansible playbook for configuration of cloudmesh client.
  - Performed initial test deployment of Spark cluster on 1 virtual machine of Chameleon cloud.
  - Deployed a Spark cluster on 2 and 3 virtual machines of Chameleon cloud.
  - Executed Naive Bayes python script for predicting readmission and recorded the run-time on Spark cluster of 1, 3 and 5 VMs on Jetstream cloud.
  - Documented the sections: introduction, technologies, architecture and preliminary data analysis.
  - Executed SGD classifier python script for predicting readmission and recorded the runtime on Spark cluster of 1, 3 and 5 VMs on Jetstream cloud.

- Srikant Ramanam [S17-IR-2028]
  - Performed data cleaning.
  - Installed cloudmesh client using developed Ansible playbook.
  - Configured cloudmesh client using developed Ansible playbook.
  - Performed initial test deployment of Spark cluster on 1 virtual machine of Virtual box.
  - Deployed a Spark cluster on 2 and 3 virtual machines of Jetstream cloud.
  - Deployed a Spark cluster on 5 and 7 virtual machines of Jetstream cloud.
  - Developed shell script for triggering spark deployment Ansible playbook.
  - Benchmarked spark deployment on chameleon and Jetstream clouds using shell script.
  - Developed python code for predicting readmission using Spark MLlib's algorithm SGD classifier.
  - Executed Decision Tree classifier python script for predicting readmission and recorded the runtime on Spark cluster on 1, 3 and 5 VMs on chameleon cloud.
  - Developed python code for predicting readmission using Spark MLlib's algorithm Naive Bayes.
  - Executed SGD classifier python script for predicting readmission and recorded the runtime on Spark cluster of 1, 3 and 5 VMs on chameleon cloud.
  - Documented the sections: automated deployment using Ansible, Data cleaning and Pre-processing.