2016 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration

SPARK—A Big Data Processing Platform for Machine Learning

Jian Fu
School of Automation
Wuhan University of Technology
Wuhan, Hubei, China 430070
Email: fujian@whut.edu.cn

Junwei Sun
School of Automation
Wuhan University of Technology
Wuhan, Hubei, China 430070
Email: sunjunwei@whut.edu.cn

Kaiyuan Wang
School of Automation
Wuhan University of Technology
Wuhan, Hubei, China 430070
Email: wangkaiyuan@whut.edu.cn

***Abstract*—Apache Spark is a distributed memory-based computing framework which is natural suitable for machine learning. Compared to Hadoop, Spark has a better ability of computing. In this paper, we analyze Spark's primary framework, core technologies, and run a machine learning instance on it. Finally, we will analyze the results and introduce our hardware equipment.**

***Keywords*-Apache Spark; Machine Learning; MLlib; Memory computing; Iteration**

I. Introduction

In recent years, the volume of data being collected, stored, and analyzed has exploded, in particular in relation to the activity on the Web and mobile devices, as well as data from the physical world collected via sensor networks. When faced with this quantity of data human-powered systems quickly become infeasible. This has led to a rise in the so-called big data and machine learning systems.

There are many open source technologies used to handle massive data volumes by distributing data storage and computation across a cluster of computers. The most widespread of these technologies is Apache Hadoop (via Hadoop MapReduce, a framework to perform computation tasks in parallel across many nodes in a computer cluster). However, MapReduce has some important shortcomings, including high overheads to launch each job and reliance on storing intermediate data and results of the computation to disk, both of which make Hadoop relatively ill-suited for use cases of an iterative or low-latency nature. Apache Spark is a new framework for distributed computing that is designed to be optimized for low-latency tasks and to store intermediate data and results in memory, so it is natural suitable for iterative application and machine learning.

This section gives a brief introduction of big data and machine learning systems. Section 2 describes Spark‘s core technologies and members. Section 3 describes the core technologies of Machine Learning on Spark with a regression instance, we test Spark‘s performance through it and analyze the results. Section 4 introduces the hardware of our cluster. Section 5 summarizes the work of this paper and get the conclusion.

II. Core Technologies Of Spark And Its Components

Spark is a general distributed computing framework which is based on Hadoop MapReduce algorithms. It absorbs the advantages of Hadoop MapReduce, but unlike MapReduce, the intermediate and output results of the Spark jobs can be stored in memory, which is called Memory Computing. Memory Computing improves the efficiency of data computing. So, Spark is better suited for iterative applications, such as Data Mining and Machine Learning.

Spark revolves around the concept of RDD (Resilient Distributed Dataset): RDD is a fault-tolerant collection of elements that can be operated in parallel and allows users to explicitly store the data in disk and memory. We can use RDD to achieve some new features that is not supported by most of current cluster programming model and earlier programming model. Such as Iterative Algorithms, SQL query, Batch, Flow calculations. RDD is a read-only data sets, and it is able to remember the graph of operations. At the same time, RDD provides a rich set of operations to manipulate the data.

Spark provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL,MLlib for machine learning, GraphX for graph processing, and Spark Streaming. The ecosystem is described as Fig 1.

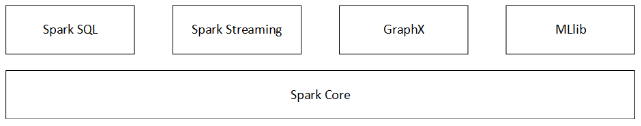

Figure 1: The ecosystem of Spark

- Spark SQL supports the SQL implementation in Spark. It has a great development in terms of data compatibility, performance optimization, components extension.
- Spark Streaming is a stream computing framework based on Spark, it provides a rich API, and integrates streaming, batch and interactive query applications.
- GraphX is a parallel computation API used for Spark charts and graphs processing. GraphX is developed

978-1-5090-3575-5/16 $31.00 © 2016 IEEE
DOI 10.1109/ICIICII.2016.27

Authorized licensed use limited to: Nitte Meenakshi Institute of Technology. Downloaded on November 18,2022 at 12:27:37 UTC from IEEE Xplore. Restrictions apply.

based on Bagel, and has a great improvement on the performance and the memory overhead reducing.
- MLlib (Machine Learning library) is a scalable Machine Learning library of Spark, it includes relevant tests and data generators. The performance of Machine Learning algorithms has a 100 times increase than MapReduce. MLlib supports the main Machine Learning algorithms, such as classification, regression, clustering, collaborative filtering, dimensionality reduction, and supports Sparse Matrix.

In general, the core technology and the foundation architecture of Spark is RDD. And the Spark SQL, MLlib, GraphX, Spark Streaming are the core members of the Spark ecosystem.

III. Core Technologies Of Machine Learning On Spark

A. Data ingestion,Cleansing and Transformation

The first step in our machine learning pipeline will be taking in the data that we require for training our models. This data can be ingested in various ways, for example, gathering user activity data from browser and mobile application event logs or accessing external web APIs to collect data on geolocation or weather. In order to use this raw data in our models, in almost all cases, we need to perform preprocessing, which might include Filtering data, Dealing with missing, incomplete, or corrupted data, Dealing with potential anomalies and errors, Joining together disparate data sources, Aggregating data. Once we have performed initial preprocessing on our data, we often need to transform the data into a representation that is suitable for machine learning models. For many model types, this representation will take the form of a vector or matrix structure that contains numerical data. Common challenges during data transformation and feature extraction include:

- Taking categorical data (such as country for geolocation or category for a movie) and encoding it in a numerical representation.
- Extracting useful features from text data.
- Dealing with image or audio data.
- Converting numerical data into categorical data to reduce the number of values a variable can take on.
- Transforming numerical features; for example, applying a log transformation to a numerical variable can help deal with variables that take on a very large range of values.
- Normalizing and standardizing numerical features ensures that all the different input variables for a model have a consistent scale. Many machine learning models require standardized input to work properly.

These data-cleansing, exploration, aggregation, and transformation steps can be carried out using both Sparks core API functions as well as the SparkSQL engine and other external Scala, Java, or Python libraries. We can also take advantage of Sparks Hadoop compatibility to read data from and write data to the various different storage systems.

Spark's MLlib library offers two broad classes of regression models: linear models and decision tree regression models. We use linear models in this paper and use the Bike Sharing dataset to run our program. This dataset contains 17379 hourly records of the number of bicycle rentals in the capital bike sharing system. It also contains variables related to date and time, weather, and seasonal and holiday information. We extract the right features from the bike sharing dataset and creating feature vectors for the linear model. Then we use the LinearRegressionWithSGD method from MLlib to train the regression model on the Bike Sharing dataset.

B. Evaluating the Performance of Regression Models

When dealing with regression models, it is very unlikely that our model will precisely predict the target variable, because the target variable can take on any real value. However, we would naturally like to understand how far away our predicted values are from the true values, so will we utilize a metric that takes into account the overall deviation.

Some of the standard evaluation metrics used to measure the performance of regression models include the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R-squared coefficient, and many others. MSE is the average of the squared error that is used as the loss function for least squares regression, the formula is :

$$MSE = \sum_{i=1}^{n} \frac{(w^T x(i) - y(i))^2}{n} \tag{1}$$

It is the sum, over all the data points, of the square of the difference between the predicted and actual target variables, divided by the number of data points. RMSE is the square root of MSE. We choose RMSE to evaluate the performance of our regression models.

In order to evaluate our predictions based on the mean of an error metric, we will first make predictions for each input feature vector in an RDD of LabeledPoint instances by computing the error for each record using a function that takes the prediction and true target value as inputs. This will return a [Double] RDD that contains the error values. We can then find the average using the mean method of RDDs that contain Double values.

C. Creating training and testing sets to evaluate parameters

We should create a test and training set for cross-validation purposes. The way is to take a random sample of 20 percent of our data as our test set (use Sample method) and the other 80 percent as our train set.

Authorized licensed use limited to: Nitte Meenakshi Institute of Technology. Downloaded on November 18,2022 at 12:27:37 UTC from IEEE Xplore. Restrictions apply.

D. Model Training

We use SGD (Stochastic Gradient Descent) to train our regression model. The SGD module is imported from the M-Llib library. We test the model‘s performance with different iterations. In general, a model trained with SGD will achieve better performance as the number of iterations increases, although the increase in performance will slow down as the number of iterations goes above some minimum number. To test the speed of machine learning on Spark, we deliberately set a large number of the maximum iteration as 100000. In our cluster, it total takes 42s to train this model. The iterations parameter list is: Params = [1, 5, 10, 20, 50, 100, 1000, 10000, 100000]

The corresponding RMSE list is: RMSE = [2.9147, 2.0623, 1.7904, 1.5859, 1.4245, 1.3793, 1.3993, 1.3993, 1.3993]

we use the matplotlib library to plot a graph of the RMSE metric against the number of iterations and use a log scale for the x axis to make the output easier to visualize:

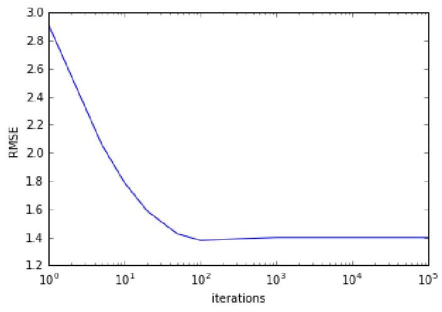

Figure 2: Train model with SGD

The fig 2 shows that the error metric indeed decreases as the number of iterations increases. It also does so at a decreasing rate as expected.

IV. Hardware Introduction And Result Analyze

Apache Spark is a fast and general-purpose cluster computing system. Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the driver program).

Specifically, to run on a cluster, the SparkContext can connect to several types of cluster managers (either Sparks standalone cluster manager, Mesos or YARN), which allocate resources across applications. Once connected, Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to SparkContext) to the executors. Finally, SparkContext sends tasks to the executors to run. The relationship chart is showed in fig 3.

In our deployment cluster, We have 3 nodes. A master node that runs the Sparkmaster process as well as the driver

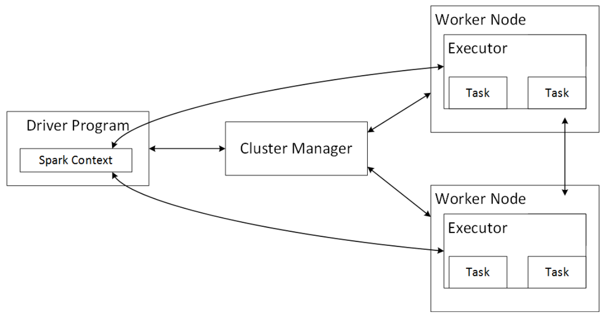

Figure 3: The connection of cluster

program, 2 worker nodes, each running an executor process. The communication between each node is supported by a Gigabit Router which is used to set up a local area network. Our cluster‘s configuration is listed as following Table I:

Table I: Hardware Equipment

Nodes	CPU	MEMORY
Master node	Intel i3 , Dual-core	8G
Worker node 1	Intel i7, Six-core	32G
Worker node 2	AMD Athon tm, Dual-core	8G

Because Spark is a memory-based computing platform, the most important factor of the Spark cluster is memory, so we are required equipping each node at least 8 GB memory. Spark currently supports three cluster managers:

1) Standalone: a simple cluster manager included with Spark that makes it easy to set up a cluster.
2) Apache Mesos: a general cluster manager that can also run Hadoop MapReduce and service applications.
3) Hadoop YARN: the resource manager in Hadoop 2.

In our deployment cluster, we use the third mode. We deploy the latest Hadoop-2.7.2 version in our cluster in advance, then we deploy Spark-1.6.1 on it. First we run the linear regression model with one worker node (Worker node 1) and it total takes 58 seconds, then we run the linear regression model with two worker nodes (node 1 and node 2) and it total takes 42 seconds. See fig 4.

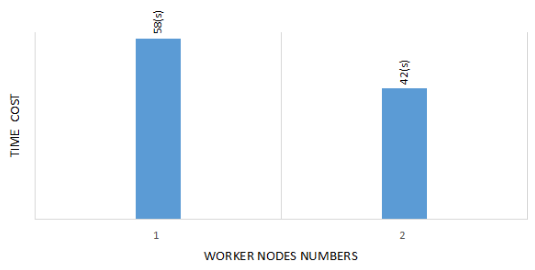

Figure 4: The influence of Workernodes numbers to Cluster

Authorized licensed use limited to: Nitte Meenakshi Institute of Technology. Downloaded on November 18,2022 at 12:27:37 UTC from IEEE Xplore. Restrictions apply.

When running Machine Learning algorithm on Spark, we can monitor Spark applications from web UIs. Every SparkContext launches a web UI, by default on port 4040, that displays useful information about the application. We list some important data in Table II when running the linear regression model with two worker nodes :

Table II: Running Information of the Cluster

Executor	RDD Blocks	Complete Tasks	Input
Node1	1	2358	4.0GB
Node2	1	1108	1841.2MB
Driver	0	0	0.0B

Based on the above results, we can know that when we add nodes to the cluster, we can take full advantage of the cluster‘s parallel computing ability and improve the performance of regression computing. From Table II, we can also know that resources in the cluster is dynamic allocation. When a worker node has better calculate ability, the Driver(master node) will advisably allocate more tasks to it, so that every node can fully utilize their calculate ability and the tasks will be finished in shortest time.

Our cluster's physical composition is showed in fig 5, it is composed of three host computers, two computer monitors and one Gigabit Router. All the three host computer‘s system are Ubuntu.

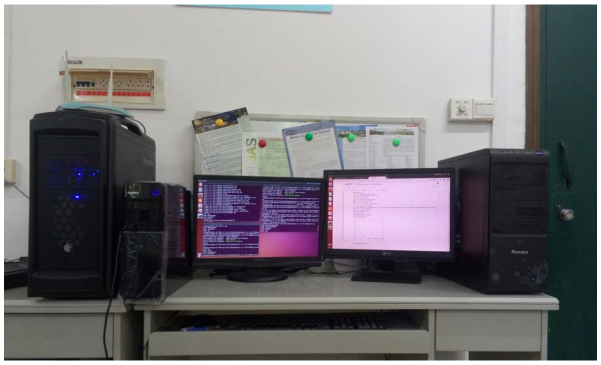

Figure 5: Cluster Physical Composition

V. Conclusion

This paper provides a general introduction of the core technologies of the Spark and an in-depth look at Spark for machine learning with big data. Based on the results from running linear regression model, we know that Spark is good at parallel computing and iteration application, it is suitable for machine learning.

Acknowledgment

This work is supported by the Natural Science Foundation of Hubei Province China (Grant No.ZRY2014001277) and the National Nature Science Foundation of China (Grant No.51475347) and the National Natural Science Foundation Committee (NSFC) of China (Grant No.51575412.)

References

[1] Nick Pentreath, *Machine Learning with Spark*, Beijing, 2015, pp. 1-140

[2] Zhijie Han, and Yujie Zhang, *A Big Data Processing Platform Based On Memory Computing*, in Parallel Architectures, Algorithms and Programming (PAAP), 2015 Seventh International Symposium on, Nanjing, 2015, pp. 172-176

[3] Aaron N. Richter, Taghi M. Khoshgoftaar, Sara Landset, and Tawfiq Hasanin, *A Multi-Dimensional Comparison of Toolkits for Machine Learning with Big Data*, in Information Reuse and Integration (IRI), 2015 IEEE International Conference on, San Francisco CA, 2015, pp. 1-8

[4] Sauptik Dhar, Congrui Yi, Naveen Ramakrishnan, and Mohak Shah, *ADMM based Scalable Machine Learning on Spark*, in Big Data (Big Data), 2015 IEEE International Conference on, Santa Clara CA, 2015, pp. 1174-1182

[5] Asmelash Teka Hadgu, Aastha Nigam, and Ernesto Diaz-Aviles *Large-scale learning with AdaGrad on Spark*, in Big Data (Big Data), 2015 IEEE International Conference on, Santa Clara CA, 2015, pp. 2828-2830

[6] Hang Tao, Bin Wu, and Xiuqin Lin, *Budgeted mini-batch parallel gradient descent for support vector machines on Spark*, in 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), Hsinchu, 2014, pp. 945-950

[7] Andre Luckow, Ken Kennedy, Fabian Manhardt, Emil Djerekarov, Bennie Vorster, and Amy Apon, *Automotive big data: Applications, workloads and infrastructures*, in Big Data (Big Data), 2015 IEEE International Conference on, Santa Clara CA, 2015, pp. 1201-1210

[8] Mark Gates, Hartwig Anzt, Jakub Kurzak, and Jack Dongarra, *Accelerating collaborative filtering using concepts from high performance computing*, in Big Data (Big Data), 2015 IEEE International Conference on, Santa Clara CA, 2015, pp. 667-676

[9] Yicheng Huang, Xingtu Lan, Xing Chen, and Wenzhong Guo, *Towards Model Based Approach to Hadoop Deployment and Configuration*, in 2015 12th Web Information System and Application Conference (WISA), Jinan, 2015, pp. 79-84

[10] E.Dede, B.Sendir, P.Kuzlu, J.Weachock, M.Govindaraju, and L.Ramakrishnan, *Processing Cassandra Datasets with Hadoop-Streaming Based Approaches*, IEEE Transactions on Services Computing , 2015, pp. 46-58

[11] Alexander J.Stimpson, and Mary L.Cummings, *Assessing Intervention Timing in Computer-Based Education Using Machine Learning Algorithms*, in IEEE Access, 2014, pp. 78-87

[12] Xianqing Yu, Peng Ning, and Mladen A.Vouk, *Enhancing security of Hadoop in a public cloud*, in Information and Communication Systems (ICICS), 2015 6th International Conference on, 2015, Amman, pp. 38-43

Authorized licensed use limited to: Nitte Meenakshi Institute of Technology. Downloaded on November 18,2022 at 12:27:37 UTC from IEEE Xplore. Restrictions apply.