American Journal of Engineering Research (AJER)
e-ISSN: 2320-0847 p-ISSN : 2320-0936
Volume-5, Issue-7, pp-01-06
www.ajer.org
Research Paper Open Access

Cloud Based Log file analysis with Interactive Spark as a Service

Nurudeen Sherif[1], MuktharAbdirahman[2], Md. Fazla Elahe[3]
[1,2,3](School of Computer Science and Engineering, Nanjing University of Science and Technology, P.R. China)

ABSTRACT : *TheSoftware applications are usually programmed to generate some auxiliary text files referred to as log files. Such files are used throughout various stages of the software development, primarily for debugging and identification of errors. Use of log files makes debugging easier during testing. It permits following the logic of the program, at high level, while not having to run it in debug mode. Nowadays, log files are usually used at commercial software installations for the aim of permanent software observation and fine-tuning. Log files became a typical part of software application and are essential in operating systems, networks and distributed systems. Log files are usually the only way to determine and find errors in a software application, because probe effect has no effect on log file analysis. Log files are usually massive and may have an intricate structure. Though the method of generating log files is sort of easy and simple, log file analysis may well be an incredible task that needs immense computing resources and complex procedures.*

Keywords: *Spark, MapReduce, Big Data, Data analysis, Log files*

I. INTRODUCTION

In the past, it is cost-prohibitive to capture all logs, let alone implement systems that act on them intelligently in real time. Recently, however, technology has matured quite a bit and today, we have all the right tools we need in the Apache Spark[1] ecosystem to capture the events in real time, process them, and make intelligent decisions based on that information. In this article, we will explore a sample implementation of a system that can capture Server logs in real time[2], index them for searching, and make them available to other analytic apps as part of a “pervasive analytics” approach[3]. This implementation is based on open source components such as Apache Spark[1] and Hue[3]. Hue can be easily installed using Cloudera Manager and parcels (via the CDH parcel, and Spark via its own parcel).

Log files are regularly large and therefore distributed techniques are usually the best way to process them. Apache Hadoop[5] is a common framework used to process log files in a distributed platform. Hadoop is an open source system and it uses the MapReduce[6]model that comprises of two stages: the mapping stage and the Reduction stage. The mapping stage problem item into smaller chunks and reduction stage collects and combines solutions to the smaller problems to present a final solution[7]. Logs are used to track a systems current state and user behavior[1]therefore log analytics systems does require huge and stable processing abilities in order to enhance performance and produce accurate results. This cannot be obtained from standalone analytics tools or perhaps a single computing framework. Thus, a log analyzer built on a cluster-computing framework is recommended.

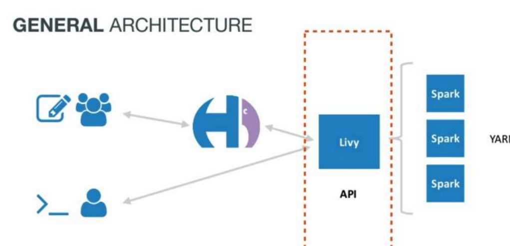

Fig.1 General Architecture

II. RELATED WORKS

The Apache Spark is a general-purpose cluster-computing engine, which is very fast and reliable. This system provides application-programing interfaces in various programing languages such as Java, Python, Scala[4]. This tool is specialized at making data analysis faster, it is pretty fast at both running programs as well

as writing data. Spark supports in-memory computing[5], that enables it to query data much faster compared to disk-based engines such as Hadoop, and also it offers a general execution model that can optimize arbitrary operator graph. Initially the system was developed at UC Berkeley[1] as research project and very quickly acquired incubator status in Apache in June 2013.

Generally speaking, Spark is an advance and highly capable upgrade to Hadoop aimed at enhancing Hadoop ability of cutting edge analysis[6]. Spark engine functions quite advance and different than Hadoop. Spark engine is developed for in-memory processing as well a disk based processing. This inmemory processing capability makes it much faster than any traditional data processing engine. For example project sensors report, logistic regression runtimes in Spark 100 times faster than HadoopMapReduce. This system also provides large number of impressive high level tools such as machine learning tool - MLib, structured data processing, Spark SQL, graph processing tool called Graph X, stream processing engine called Spark Streaming, and Shark for fast interactive question device[5].

Spark can now be offered as a service to anyone in a simple way: Spark shells in Python or Scala can be ran by Livy in the cluster while the end user is manipulating them at his own convenience through a REST api. Livy is a new open source Spark REST Server for submitting and interacting with your Spark jobs from anywhere. Livy is conceptually based on the incredibly popular IPython/Jupyter[7], but implemented to better integrate into the Hadoop/Spark ecosystem with multi users. Regular non-interactive applications can also be submitted.

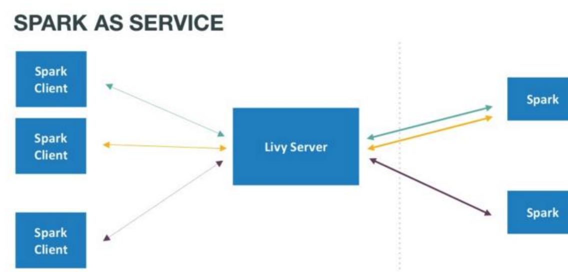

Fig.2 Spark as a Service

The output of the jobs can be introspected and returned in a tabular format, which makes it visualizable in charts. Livy can point to a unique Spark cluster and create several contexts by users. With YARN impersonation, jobs will be executed with the actual permissions of the users submitting them. Livy also enables the development of Spark Notebook applications. Those are ideal for quickly doing interactive Spark visualizations and collaboration from a Web browser!

III. PROPOSED METHOD

Spark programming tools[1] makes it possible to perform batch analysis in minimum time interval. Together with a livy server[3], Analysis can be done in a cloud computing environment and in real-time. Below is an illustration of the design.The Log Analysis framework proposed in this paper consists of three different elements; the feature extraction, the feature transformation and pattern mining[8]. Spark supports in-memory computing and performs far better on recurring algorithms[9].

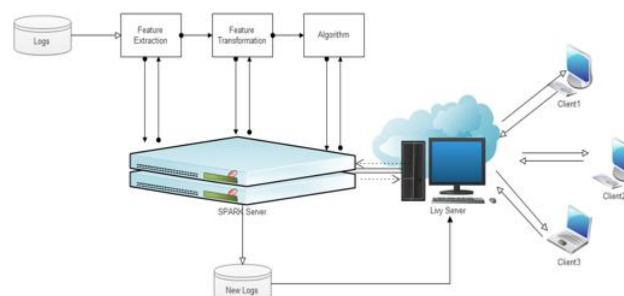

Fig.3 Log Analysis Architecture

IV. FEATURE EXTRACTION

The Term frequency-inverse document frequency $(TF - IDF)$[10] is a feature vectorization method widely used in text mining to reflect the importance of a term to a document in the corpus. Denote a term by tt, a document by dd, and the corpus by DD. Term frequency $TF(t,d)TF(t,d)$ is the number of times that term tt appears in document dd, while document frequency $DF(t,D)DF(t,D)$ is the number of documents that contains

term tt. If we only use term frequency to measure the importance, it is very easy to over-emphasize terms that appear very often but carry little information about the document, e.g., “a”, “the”, and “of”. If a term appears very often across the corpus, it means it doesn’t carry special information about a particular document. Inverse document frequency is a numerical measure of how much information a term provides:

$$IDF(t, D) = log \frac{|D|+1}{DF(t,D)+1},$$

where |D||D| is the total number of documents in the corpus. Since logarithm is used, if a term appears in all documents, its IDF value becomes 0. Note that a smoothing term is applied to avoid dividing by zero for terms outside the corpus. The TF-IDF measure is simply the product of TF and IDF:

$$TFIDF(t, d, D) = TF(t, d)ID \cdot F(t, D).$$

There are several variants on the definition of term frequency and document frequency. In spark.mllib, we separate TF and IDF to make them flexible[10]–[12]. Below is an implementation of feature extraction in Scala:

```
import org.apache.spark.ml.feature.{HashingTF, IDF, Tokenizer}

val sentenceData = sqlContext.createDataFrame(Seq(
  (0, "Hi I heard about Spark"),
  (0, "I wish Java could use case classes"),
  (1, "Logistic regression models are neat")
)).toDF("label", "sentence")

val tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words")
val wordsData = tokenizer.transform(sentenceData)
val hashingTF = new HashingTF()
  .setInputCol("words").setOutputCol("rawFeatures").setNumFeatures(20)
val featurizedData = hashingTF.transform(wordsData)
val idf = new IDF().setInputCol("rawFeatures").setOutputCol("features")
val idfModel = idf.fit(featurizedData)
val rescaledData = idfModel.transform(featurizedData)
rescaledData.select("features", "label").take(3).foreach(println)
```

V. FEATURE TRANSFORMATION

Feature transformation[13] is implemented using a tokenizer. Tokenization is the process of taking text (such as a sentence) and breaking it into individual terms (usually words). The scala code below shows how to split sentences into sequences of words[14]. RegexTokenizer[15] allows more advanced tokenization based on regular expression (regex) matching. By default, the parameter “pattern” (regex, default: \s+) is used as delimiters to split the input text[15]. Alternatively, users can set parameter “gaps” to false indicating the regex “pattern” denotes “tokens” rather than splitting gaps, and find all matching occurrences as the tokenization result.

```
import org.apache.spark.ml.feature.{Tokenizer, RegexTokenizer}

val sentenceDataFrame = sqlContext.createDataFrame(Seq(
  (0, "Hi I heard about Spark"),
  (1, "I wish Java could use case classes"),
  (2, "Logistic,regression,models,are,neat")
)).toDF("label", "sentence")
val tokenizer = new Tokenizer().setInputCol("sentence").setOutputCol("words")
val regexTokenizer = new RegexTokenizer()
  .setInputCol("sentence")
  .setOutputCol("words")
  .setPattern("\\W")  // alternatively .setPattern("\\w+").setGaps(false)

val tokenized = tokenizer.transform(sentenceDataFrame)
tokenized.select("words", "label").take(3).foreach(println)
val regexTokenized = regexTokenizer.transform(sentenceDataFrame)
regexTokenized.select("words", "label").take(3).foreach(println)
```

Logs usually contain time-stamp data for each of their entrances. The execution time of an abnormal log can aid in recognizing the irregularity.

VI. PATTERN MINING

The FP-growth algorithm is described in the paper[16] **as** Mining frequent patterns without candidate generation, where "FP" stands for frequent pattern. Given a dataset of transactions, the first step of FP-growth is to calculate item frequencies and identify frequent items[17]. Different from Apriori-like algorithms designed for the same purpose, the second step of FP-growth uses a suffix tree (FP-tree) structure to encode transactions without generating candidate sets explicitly, which is usually expensive to generate. After the second step, the frequent item sets can be extracted from the FP-tree. In spark, a parallel version of FP-growth called PFP is used, as described in [18], PFP: Parallel FP-growth for query recommendation. PFP distributes the work of growing FP-trees based on the suffices of transactions, and hence more scalable than a single-machine implementation. We refer readers to [16], [17], [19]–[25] for more details. Spark's FP-growth implementation takes the following parameters:

- minSupport: the minimum support for an itemset to be identified as frequent. For example, if an item appears 2 out of 4 transactions, it has a support of 2/4=0.5.
- numPartitions: the number of partitions used to distribute the work.

The following code illustrates how to mine frequent itemsets and association rules in scala:

```
import org.apache.spark.mllib.fpm.FPGrowth
import org.apache.spark.rdd.RDD

val data = sc.textFile("data/mllib/sample_fpgrowth.txt")

val transactions: RDD[Array[String]] = data.map(s => s.trim.split(' '))

val fpg = new FPGrowth()
  .setMinSupport(0.2)
  .setNumPartitions(10)
val model = fpg.run(transactions)

model.freqItemsets.collect().foreach { itemset =>
  println(itemset.items.mkString("[", ",", "]") + ", " + itemset.freq)
}

val minConfidence = 0.8
model.generateAssociationRules(minConfidence).collect().foreach { rule =>
  println(
    rule.antecedent.mkString("[", ",", "]")
      + " => " + rule.consequent .mkString("[", ",", "]")
      + ", " + rule.confidence)
}
```

VII. EVALUATING THE MODEL

The preparation log is attached with data about its state; if the experiment that created that log record passed or failed. **F-score** might be utilized to assess results. Here, a brief presentation of the F-score[26]. Firstly, the

accompanying terms are clarified.

- True Positive (TP) represents the quantity of irregular logs that were accurately mined by the algorithm as unusual logs.
- False Negative (FN) represents the quantity of irregular logs mistakenly anticipated, as would be expected logs.
- False Positive (FP) represents the quantity of the mistakenly sorted logs as unusual logs.
- True Negative (TN) represents the quantity of regular logs that were effectively ordered.

Precision and recall are two broadly utilized measurements for assessing classifiers[27]. Precision, measures the extent of the unusual logs that were accurately ordered. Recall, from the other perspective, measures the extent of the unusual logs with those ordered as unusual logs by the learning algorithm[28]. While having high review, the learning technique infrequently miss-predicts unusual logs as ordinary logs, while having high similarity, it once in a while miss-predicts typical logs as anomalous ones.

VIII. CONCLUSION

To finish up, this Spark-based execution of Log Analysis trying to sort log documents as regular or irregular required an excessive amount of time. This cloud-based implementation of Spark log analysis is ideal, since the entire procedure, including the component extraction, transformation and training, now takes minutes rather than a few hours. Sadly, this accompanies lower precision as far as what logs HUE finds as irregular, since the cloud-based execution has a tendency to order more logs as irregular.

REFERENCES

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark : Cluster Computing with Working Sets," *HotCloud'10 Proc. 2nd USENIX Conf. Hot Top. cloud Comput.*, p. 10, 2010.

[2] A. Loganathan, A. Sinha, V. Muthuramakrishnan, and S. Natarajan, "A Systematic Approach to Big Data Exploration of the Hadoop Framework," vol. 4, no. 9, pp. 869–878, 2014.

[3] M. Beckmann, F. F. Nelson, B. S. Pires, and M. a. Costa, "A User Interface for Big Data with RapidMiner," no. February 2016, p. 16, 2014.

[4] J. Scott, "Getting started with Apache Spark," *J. Chem. Inf. Model.*, vol. 53, p. 160, 1989.

[5] I. Stoica, "Conquering big data with spark and BDAS," in *The 2014 ACM international conference on Measurement and modeling of computer systems - SIGMETRICS '14*, 2014, vol. 42, no. 1, pp. 193–193.

[6] G. Mone, "Beyond Hadoop," *Commun. ACM*, vol. 56, no. 1, pp. 1–3, 2013.

[7] U. Woiski, Emanuel Rocha(Department of Mechanical Eng., Faculty of Engineering, "Data Analysis and Machine Learning in Python," *Proc. PROBABILISTIC Progn. Heal. Manag. ENERGY Syst. Work.*, p. p. 38., 2015.

[8] R. Iváncsy and I. Vajk, "Frequent pattern mining in web log data," *Acta Polytech. Hungarica*, vol. 3, no. 1, pp. 77–90, 2006.

[9] S. R., B. Ganesh H.B., S. Kumar S., P. Poornachandran, and S. K.P., "Apache Spark a Big Data Analytics Platform for Smart Grid," *Procedia Technol.*, vol. 21, pp. 171–178, 2015.

[10] T. P. Hong, C. W. Lin, K. T. Yang, and S. L. Wang, "A heuristic data-sanitization approach based on TF-IDF," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6703 LNAI, no. PART 1, pp. 156–164.

[11] Z. Yun-tao, G. Ling, and W. Yong-cheng, "An improved tf-idf approach for text classification," *J. Zhejiang Univ. A*, vol. 6, no. 60082003, pp. 49–55, 2005.

[12] T. Roelleke and J. Wang, "TF-IDF uncovered: a study of theories and probabilities," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, 2008, pp. 435–442.

[13] A. Kusiak, "Feature transformation methods in data mining," *IEEE Trans. Electron. Packag. Manuf.*, vol. 24, no. 3, pp. 214–221, 2001.

[14] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "MLlib: Machine Learning in Apache Spark," *CoRR*, 2015.

[15] C. S. Lengsfeld and R. a; Shoureshi, "Specifying a Parser Using a Properties File," vol. 1, no. 19, 2008.

[16] R. Mishra and Choubey, "Discovery of Frequent Patterns from Web Log Data by using FP-Growth algorithm for Web Usage Mining," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 2, no. 9, pp. 311–318, 2012.

[17] X. Wei, Y. Ma, F. Zhang, M. Liu, and W. Shen, "Incremental FP-Growth mining strategy for dynamic threshold value and database based on MapReduce," in *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2014*, 2014, pp. 271–276.

[18] Z. Rong, D. Xia, and Z. Zhang, "Complex statistical analysis of big data: Implementation and application of apriori and FP-growth algorithm based on MapReduce," in *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, 2013, pp. 968–972.

[19] C. Borgelt, "An implementation of the FP-growth algorithm," *Proc. 1st Int. Work. open source data Min. Freq. pattern Min. implementations - OSDM '05*, pp. 1–5, 2005.

[20] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Zhexue, and S. Feng, "Balanced parallel FP-growth with mapreduce," in *Proceedings - 2010 IEEE Youth Conference on Information, Computing and Telecommunications, YC-ICT 2010*, 2010, pp. 243–246.

[21] J. Korczak and P. Skrzypczak, "FP-growth in discovery of customer patterns," in *Lecture Notes in Business Information Processing*, 2012, vol. 116 LNBIP, pp. 120–133.

[22] R. K. et. al Soni, "An FP-Growth Approach to Mining Association Rules," *Int. J. Comput. Sci. Mob. Comput.*, vol. 2, no. 2, pp. 1–5, 2013.

[23] B. Kumar and K. Rukmani, "Implementation of Web Usage Mining Using APRIORI and FP Growth Algorithms," *Int. J. of Advanced networking and Applications*, vol. 404, no. 06. pp. 400–404, 2010.

[24] Q. Tu, J. F. Lu, J. Bin Tang, and J. Y. Yang, "The FP-tree algorithm used for data stream," in *2010 Chinese Conference on Pattern Recognition, CCPR 2010 - Proceedings*, 2010, pp. 829–833.

[25] J. Tan, Y. Bu, and B. Yang, "An Efficient Frequent Pattern Mining Algorithm," in *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, 2009, vol. 2, pp. 47–50.

[26] Y. Z. Guandong Xu, Yu Zong, Peter Dolog, "Co-clustering Analysis of Weblogs Using Bipartite Spectral Projection Approach," *14th Int. Conf.* , vol. 6278, no. Proceedings, Part III, pp. 398–407, 2010.

[27] M. Zhu, "Recall, precision and average precision," *Dep. Stat. Actuar. Sci. …*, pp. 1–11, 2004.

[28] M. Buckland and F. Gey, "The relationship between Recall and Precision," *J. Am. Soc. Inf. Sci.*, vol. 45, no. 1, pp. 12–19, 1994.