Distributed Search on a Large Amount of Log Data

Full text index in a Big Data architecture

Fabrice Mourlin, Charif Mahmoudi
Algorithmic, Complexity and Logic Laboratory
UPEC University
Creteil, France
Email : fabrice.mourlin@u-pec.fr, charif.mahmoudi@lacl.fr

Guy Lahlou Djiken
Applied Computer Science Laboratory
Douala University
Douala, Cameroon
Email : ldjiken@fs-univ-douala.cm

Abstract—**Log analysis is the basis for planning maintenance operations. It is used to predict software and hardware failures. Their costs can be high, ranging from activity blocking to data loss, even penalties for late results. In this context, to achieve an even better quality of service, we have built a distributed application for collecting and analyzing software logs. Key properties had to be taken into account such as the number of logs per hour, the variety of formats of these logs or the indexing of information known in these logs. Our approach responds to these properties by using a Big Data cluster and installing a custom indexing engine for software log analysis. Our results show a reduction in software failures and, therefore, better availability of software services under monitoring. This result leads to rethinking software maintenance and reviewing the sizing of our cluster according to the number of monitored applications correlated with the throughput of each one.**

Keywords-Software log; indexing; Big Data; streaming; planning maintenance.

I. INTRODUCTION

Logs are files hosted on the application servers to be monitored, which regularly record their activities such as access to resources, requests being processed, etc. The logs are used to retrieve information on abnormal behavior, alerts, errors and their scheduling, etc. They are full of information, including date of an event, the invoked Web address, the uniform resource location origin, the response code of the page (code 403, code 201, etc.), its payload, etc. The analysis of log files is the evaluation of a set of information recorded from one or more events that have occurred in an application environment. This practice is used to analyze user behavior and identify patterns of behavior, or identify and anticipate incidents. These same techniques are applied to ensure compliance of server behavior with the regulations in place, such as government applications.

Analyzing logs is a challenge and requires tedious work for the Software supervision teams because of the volume. Other features are crucial such as the diversity of types of logs, as well as the proprietary formats, elastic architectures, aggregation of time-stamped data, detection of behavior patterns, etc. Using log analysis software that leverages machine-learning algorithms dramatically reduces the workload on supervisory teams who can focus on value-added tasks. Such log analysis software allows monitoring, aggregation, indexing and analysis of all application and infrastructure log data. The tools such as the ELK (Elasticsearch, Logstash and Kibana) suite software, become a reference in the monitoring domain [1] because of their adaptability and polyvalence.

Log analysis tools provide better visibility into the health and availability of applications using dashboards. This allows software administrators to monitor critical events from a central location. Synthetic situations thus appear where an administrator is able to decide to anticipate a maintenance task in order to ensure continuity of service. For example, many services are written in Java where memory saturation problems cause the need to restart a Java Virtual Machine (JVM). In Big Data on the edge applications, the use of an energy source is often the constraint that leads not to the restart but to the migration of a service, from a network node on another, having still energy resources.

Over time, the use of specific indexing techniques has enriched log file analysis strategies. The structure of these input data is always formatted even though the formats vary. In addition, the use of a data schema provides additional typing which highlights the meaning of these lines of information. It is then useful to separate data storage from data indexing. The search for a pattern of behavior is more effective and prevention becomes better and more reactive.

Log analysis solutions incorporate additional data sources. Thus, machine learning and other analytical techniques push the boundaries of new use cases in application performance management, security intelligence, event management and behavior analysis.

The rest of this paper is organized as follows. Section II describes the works close to our domain. Section III provides a precise description of our use case. Section IV addresses the software architecture of our distributed platform. Section V goes into finer details on our streaming approach, which includes an indexing step. Section VI focuses about on our results and the impact on the maintenance task. The acknowledgement and conclusions close the article.

II. RELATED WORKS

The use of logging has been common practice in IT for many years. Its use for intervention prediction is more recent, but the interest of this approach has quickly become essential in companies and more particularly in any computer system offering services 24 hours a day. Publications have long been available in order to present the broad spectrum of log analysis techniques [2]. In the context of distributed computing systems, Qiang Fu has published some very useful results on behavior anomaly detection from logs [3]. W. Xu's work focuses on the structure of logs and the impact on the analysis strategy [4].

In the more specific context of analysis with prediction, Chinghway Lim's work is based on the use of individual message frequencies to characterize system behavior and the ability to incorporate domain-specific knowledge through user feedback [5]. Jakub Breier follows the same approach based on Apache Hadoop technique to enable processing of large data sets in a parallel way [6].

T. Li and Y. Jiang propose a platform to facilitate the data analytics for system event logs [7]. This work is an end-to-end solution that utilizes advanced data mining techniques to assist log analysts. They apply learning techniques to extract useful information from unstructured raw logs. The parsing technique contains an index management.

Steven Yen published recently a book on the topic of intelligent log analysis using Machine and deep learning [8]. He explains how deep learning implementation can improve the result quality when the data volume achieves a limit. He provides a comparison with a K mean model and two distinct implementations. This work shows that a global solution does not exist and some add-ons are crucial. For instance, the use of an indexing process of log messages could lead to a cost reduction at runtime.

In more constrained fields such as real time, log analysis systems must be able to detect an anomaly in a limited time. Biplob Debnath presents *LogLens* [9] that automates the process of anomaly detection from logs with minimal target system knowledge. *LogLens* presents an extensible index process based on new metrics (term frequency and boost factors). The use of temporal constraint also intervenes in the recognition of behaviour pattern. So, abnormal events are defined as visible in a time window while other events are not. This allows semi-automatic real-time device monitoring.

Aspects remain to be covered such as the use of cross logging in analysis and log indexing. The reason is the separation of storage and indexing. In the previous works, the storage is generally done by the use of relational databases while the indexing uses rather NoSQL (Not only Structured Query Language) databases where the notion of join cannot be easily implemented.

III. Use Case Description

A. Historic

When doing software monitoring, the first thing we want to get is a reason for each failure, or even the root cause of the problem. The idea is then to automate the creation of an intervention request ticket and to follow up this maintenance operation until the update operation of the service concerned.

Many software programs exist for this need, such as Free Management of Computer Park (GLPI) [10], and new software monitoring needs are appearing in order to improve this incident management by anticipating maintenance operations. The idea is then to reduce the costs of maintenance task, which generally correspond to service interruptions. Even if service replication strategies make it possible to lessen the effects of a failure, it is preferable to anticipate this problem and to research before the event in order to prevent it.

B. Log information

At the heart of log analysis, there is the collection of events, such as the setup of a service, the attempt to connect to the system for example, a configuration request, or variations in CPU or storage, or the trace of an application event (receipt of an order, etc.).

A log entry contains information such as the date and time of the event, on which network node the event occurred, user identification, contextual information (configuration, security) or even the service at the origin of the event.

C. Nominal scenario

The description of our use case is based on our desire to monitor the activity of our information system. This includes several application servers and data management servers, interconnected by a software bus. It enables intelligent message routing between applications and provides a first level of fault tolerance in the event of a service failure.

Our servers provide log files, but also our applications deployed on the servers. Many formatted files are thus written in different directories. To perform a centralized log analysis, a preparatory step consists in moving the files to a dedicated machine. A second step consists in analyzing the data to keep the useful parts on the one hand and to index the key parts on the other hand. This pipeline continues with the use of a statistical model to predict the actions to be planned (Figure 1). Finally, the last step concerns the collection of metrics in order to evaluate the monitoring process.

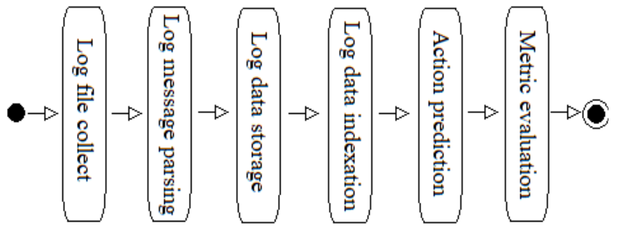

Figure 1. Log file lifecycle.

During our first prototypes, the volume of data processed exceeded 10 MB per hour and it became evident that such a sequential process could not meet our needs. The choice of a Big Data cluster for the processing of such volumes of text is legitimate, especially since this work relates to the monitoring of distributed systems.

IV. Software Architecture

Log analysis tasks often have strict due dates and data quality is a primary concern in software monitoring activities. This underlines the importance of finely managing the sequencing of tasks on the analysis platform.

The Hadoop ecosystem offers a set of software to process huge data sets. It was originally designed to run on clusters of

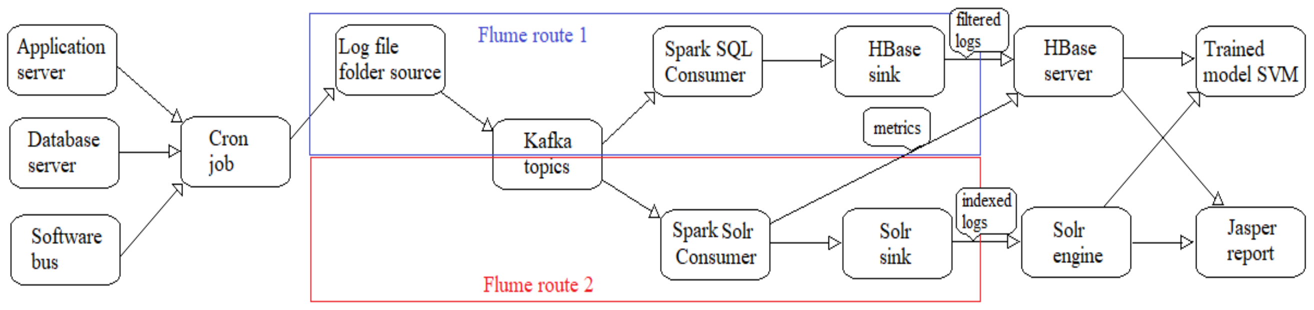

Figure 2. Big Data workflow for log analysis.

physical machines. Distributed analytical frameworks, for example MapReduce, evolve into resource managers that gradually transform Hadoop into a very versatile data operating system. With these frameworks, one can perform a wide range of data manipulation and analysis operations by connecting them to Hadoop Distributed File System (HDFS) as a document storage system [11].

Hadoop is highly scalable; it is easy to add a new service such as a search engine. As it includes the Zookeeper clustering tool, it is able to deploy on a set of nodes a search engine to manage a large volume of text-oriented data.

We have made the choice to use Apache Solr to index, search a large amount of business data, and provide relevant content based on a search query [12].

A. Big Data platform

A part of our work is based on Solr framework 8.1 and the integration with all other components in Hortonworks Data Platform Virtual Machine (HDP VM), such as Apache HBase, Apache Spark, Apache Kafka, in addition to some other open source tools. A part of our work relies on specific configurations of the tools; another part is the development of specific components for customizing the behavior of the Hadoop tools.

Our article outlines our approach and a simplified architecture for analyzing software-generated logs to detect functional-related issues. Our architecture is a batch analytics system analyzing Solr query logs.

The diagram from Figure 2 illustrates the high level of our software architecture.

We use shell scripts to collect log files destined for a remote directory (named "log file folder source"). With a common data ingestion path, the logs go from an Apache Flume source, then to a Kafka channel and are transmitted to a first Spark consumer (named "Spark SQL consumer"). Its essential task is to recognize and process the contents of the file and load them into an SQL table in memory, perform filter operations and put them in common format. Then, the route continues with a backup of these data in HBase tables. The role of this Flume route is to store structured information in a column-oriented database (the blue route in Figure 2).

In parallel, another route has the role of indexing the data from the logs (red route in Figure 2). From the same Kafka source, a second Spark consumer (named "Spark Solr consumer") takes care of data indexing while respecting the Solr schema. The index is updated for the query steps and then we use of a model for the prediction of maintenance tasks.

In this architecture, HBase is a highly reliable data store, supporting disaster recovery and cross-datacenter replication. Solr Cloud is the indexing and search engine. It is completely open and allows us to personalize text analyzes. It allows a close link with HBase database so the schemas used by both tools are designed in a closely related way.

The Jasper Report tool allows us to build a report from data automatically and regularly. Suitable cross tables help to give priorities to software maintenance tasks.

B. Configuration

1) Via operating system

Several elements of this architecture support ad hoc configuration. We have defined specific configuration scripts for routing log files to the "log file folder" directory, source Flume. We use entries in cron tables to ensure regular data collection.

2) Via event streaming-tools.

We have described two Flume routes within our Big Data cluster. Flume configurations correspond to the creation of routing agents so that information reaches the programs that use them.

The Flume and Kafka tools are both event-streaming tools. While their roles are comparable, the developments in these two projects are very different and there are now more Kafka connectors. Thus, the popularity of Apache Kafka is currently higher than that of Flume. We have kept software routes with Flume for event routing, but we define Kafka topics to ensure decorrelation between components. This makes it possible to simplify the management of components, among other things for updates. In addition, the Kafka API allows more controls on the management of messages associated with a topic; for example time management. We have added rules to ensure that a received message is processed within an hour. In that case, we raise an alert and the data is saved in the local file system.

3) Via persistent storage.

We wrote the script for creating tables structured in families of columns to keep the information from the log files. The column families are logical and physical groups of columns. The columns in one family are stored separately from the columns in another family. Because we have data that are not often queried, we assign that data to a separate column family.

Because the column families are stored in separate HFiles, we keep the number of column families as small as possible. We also want to reduce the number of column families to reduce the frequency of mem-store flushes, and the frequency of compactions. Moreover, by using the smallest number of column families possible, we improve the load time and reduce disk consumption.

4) Via indexing engine.

Apache Solr is an open source search engine and Solr index can be considered as an equivalent of a SQL table. A standalone instance maintains several indexes. However, on our Big Data cluster, the Solr installation is also distributed. In that context, we have four shards with a replication rate equals to three. This allows us to distribute operations by reducing blockages due to frequent indexing. We have configured not only the schema, but also the data handlers (*schema.xml* and *solrconfig.xml* files).

Our schema defines the structure of the documents that are indexed into Solr. This means the set of fields that they contain. We also define the datatype of those fields. It configures also how field types are processed during indexing and querying. This allows us to introduce our own parsing strategy via class programming.

C. Component architecture

1) Based on Spark framework.

To implement this architecture, we have developed several components using the Spark framework version 2.4.7. These components are at the heart of Flume routes, so their sequencing is based on the Spark-streaming module. In other words, when log data are available, the scheduler creates micro batches to process these data during a fixed duration window. In order to keep the results of the processing, the components save their results in a HBase database installed on the Hadoop cluster [15].

We have two consumers of the data associated with the Kafka topic. Spark SQL consumer uses the Spark SQL module to store data in an HBase database whose schema is structured in family of columns. The labels of these families of columns are involved in the data schema of the second Spark consumer.

HBase is a database distributed on the nodes of our Hadoop cluster, which allows having a persistence system where the data are highly available because the replicated rate on separate nodes is set to three.

2) Based on Spring Data.

Spark Solr consumer uses the Spring Data and SolrJ library to index the data read from the Kafka topic. It splits the data next to the Solr schema where the description of each type includes a "*docValue*" attribute, which is the name of the HBase column family. For each Solr type, our configuration provides a given analyzer. We have developed some of the analyzers in order to keep richer data than simple raw data from log files. Finally, the semantic additions that we add in our analysis are essential for the evaluation of Solr query. Likewise, we store the calculated metrics in HBase for control.

SolrCloud is deployed on the cluster through the same Zookeeper agents. Thus, the index persistence system is also replicated. We therefore separate the concepts of backup and search via two distinct components. This reduces the blockages related to frequent updates of our HBase database [14].

3) Based on SolrJ library.

At the beginning of our Solr design, we have built our schema based on our data types. Some of them were already defined, but some others are new. In addition, we have implemented new data classes for the new field types. For example, we used *RankFieldType* as a type of some fields in our schema. Then, it becomes a sub class of *FieldType* in our Solr plugin.

We have redesigned Solr filters so that they can be used in our previous setups. Our objective was to standardize the values present in the logs coming from different servers. Indeed, the messages provide information of the form <attribute, value> where the values certainly have units. However, the logs do not always provide the same units for the same attribute calculation. The analysis phase is the place to impose a measurement system in order to be able to compare the results later.

The development pattern proposed by SolrJ is simple because it proposes abstract classes like *TokenFilter* and *TokenFilterFactory* then to build inherited classes. Then we have to build a plugin for Solr and drop it in the technical directory agreed in the installation of the tool [13].

4) Based on Spark-MLlib.

In Artificial Intelligence, Support Vector Machine (SVM) models are a set of supervised learning techniques designed to solve discrimination and regression problems. SVMs have been applied to a large number of fields (bioinformatics, information research, computer vision, finance, etc.) [16]. SVM models are classifiers, which are based on two key ideas, which allow to deal with nonlinear discrimination problems, and to reformulate the ranking problem as a quadratic optimization problem. In our project, SVMs can be used to decide to which class of problem a recognized sample belongs. The weight of these classes if linked to the Solr metrics on these names. This amounts to predicting the value of a variable, which corresponds to an anomaly.

All filtered log entries are potentially useful input data if it is possible that there are correlations between informational messages, warnings, and errors. Sometimes the correlation is strong and therefore critical to maximizing the learning rate. We have built a specific component based on Spark MLlib It supports binary classification with linear SVM. Its linear SVMs algorithm outputs an SVM model [18].

We applied prior processing to the data from our HBase tables before building our decision modeling. These processes are grouped together in a pipeline, which leads to the creation of the SVM model with the configuration of its hyper-parameters such as *weightCol*. Part of the configuration of these parameters comes from metrics calculated by our indexing engine (Figure 2). Once created and tested, the model goes into action to participate in the prediction of incidents. We use a new version of the SVM model builder based on distributed data augmented. This comes from an article written Nguyen, Le and Phung [19].

5) Based on Jasper Report library.

This reporting library allows us to build weekly graphical reports on indexing activity. This information is a help to check the suitability of the SVM model, which supports prediction requests following pattern recognition. The representations are documents in pdf format; we did not automate the impact of this data extraction on the use of our decision-making model.

V. BIG DATA STREAMING

We use Apache Kafka as queue system for our logs. Then we use spark streaming library to read from Kafka topic and process logs on the fly. Spark Streaming is a real-time processing tool that runs on top of the Spark engine. The scheduler exploits all the computation resources of our cluster. Each node runs several executors, which run tasks and keeps data in memory or disk storage across them.

In our program, the Spark context sends all the tasks for the executors to run.

A. Filtered log strategy

1) Asynchronous reading.

Our component called Spark SQL Consumer contains a Kafka receiver class, which runs an executor as a long-running task. Each receiver is responsible for exactly one input discretized stream (called *DStream*). In the context of the first Flume route, this stream connects the Spark streaming to the external Kafka data source for reading input log data.

Because the log data rate is high, our component reads from Kafka in parallel. Kafka stores the data logs in topics, with each topic consisting of a configurable number of partitions. The number of partitions of a topic is an important key for performance considerations as this number is an upper bound on the consumer parallelism. If a topic has N partitions, then our component can only consume this topic with a maximum of N threads in parallel. In our experiment, the Kafka partition number is set to four.

2) Normalized form.

Since log data are collected from a variety of sources, data sets often use different naming conventions for similar informational elements. The Spark SQL Consumer component aims to apply name conventions and a common structure. The ability to correlate the data from different sources is a crucial aspect of log analysis. Using normalization to assign the same terminology to similar aspects can help reduce confusion and error during analysis [17]. This case occurs when log messages contain values with different units or distinct scales. The log files are grouped under topics. We apply transformations depending on the topic the data come from. The filtered logs are cleaned and reorganized and then are ready for an export into an HBase instance.

3) Stuctured data storage.

Next step, the Spark SQL Consumer component inserts the cleaned log data into memory data frames backed to a schema. We have defined a mapping between HBase and Spark tables, called Table Catalog. There are two main difficulties of this catalog.

a) The row key definition implies the creation of a specific key generator in our component.

b) The mapping between table column in Spark and the column family and column qualifier in HBase needs a declarative name convention.

The HBase sink exploits the parallelism on the set of Region servers, which are under control of the HBase master. The HBase sink treats both *Put* operation and *Delete* operation in a similar way, and both actions are performed in the executors. The driver Spark generates tasks per region. The tasks are sent to the preferred executors collocated with the region server, and are performed in parallel in the executors to achieve better data locality and concurrency. By the end of an exportation, a timed window of log data is stored into HBase tables.

B. Index construction and query

1) The index pipeline

The strategy of the Spark Solr Consumer component deals with the ingestion of the log data into Apache Solr for search and query. The pipeline is built with Apache Spark and Apache Spark Solr connector (Figure 3). Spark framework is used for distributed in memory compute, transform and ingest to build the pipeline.

The Apache HBase role is the log storage and the Apache Solr role is the log indexing. Both are configured in cloud mode Multiple Solr servers are easily scaled up by increasing server nodes. The Apache Solr collection, which plays the role of SQL table, is configured with shards. The definition of shard is based on the number of partitions and the replicas rate for fault tolerance ability.

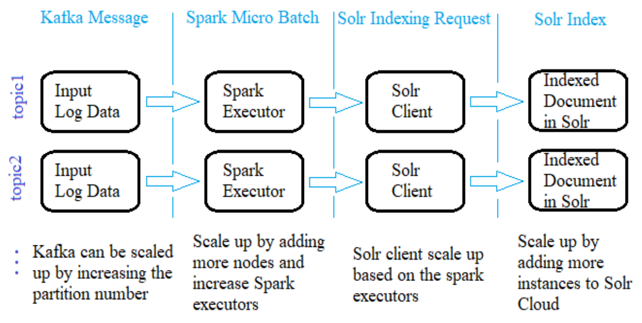
Kafka Message
Spark Micro Batch
Solr Indexing Request
Solr Index
topic1
topic2
Input Log Data
Spark Executor
Solr Client
Indexed Document in Solr
Input Log Data
Spark Executor
Solr Client
Indexed Document in Solr
Kafka can be scaled up by increasing the partition number
Scale up by adding more nodes and increase Spark executors
Solr client scale up based on the spark executors
Scale up by adding more instances to Solr Cloud

Figure 3. Overall high-level architecture of the index pipeline.

The Spark executors run a task, which transforms and enriches each log message (format detection). Then, the Solr client takes the control and sends a REST request to Solr Cloud Engine. Finally, depending on the Solr leader, a shard is updated.

2) The query process.

We also use Solr Cloud as a data source Spark when we create our ML model. We send requests from spark ML classes and read results from Solr (with the use of Solr Resilient Distributed Dataset (SolrRDD class). The pre statement of the requests is different from the analysis of the log document. Their configuration follows another analysis process.

With Spark SQL, we expose the results as SQL tables in the Spark session. These data frames are the base of our ML model construction. The metrics called TF (Term Factor) and

IDF (Inverse Document Frequency) are key features for the ML model. We have also used boost factor for customizing the weight of part of log message.

VI. Results and Actions

We have several kinds of results. A part is about our architecture and the capacity to treat log messages over time. Another part is about the classification of log messages. The concepts behind SVM algorithm are relatively simple. The classifier separates data points using a hyperplane with the largest amount of margin. In our working context, the margin between log patterns is a suitable discriminant.

A. *Data features*

1) *Architecture measurement*

For our tests, we used previously saved log files from 20 days of application server and database server operations. We were interested in the performance of the two Spark consumers, the Spark SQL Consumer and the Spark Solr Consumer.

For Spark SQL Consumer, the volume of data to analyze is 81.7 M rows in HBase. To exploit this data, we used a cluster of eight nodes on which we deployed Spark and HBase. The duration of the tests varies between 24 minutes and 2 hours and 1 minute.

For Spark Solr Consumer, the volume of data indexed is 87.2M rows indexed in about an hour. The number of documents indexed per second is 28k.

We only installed Solr on four nodes with four shards and a replication rate of three. We have seen improved results by increasing the number of Spark partitions (*RangePartitioner*). At runtime for our data set based on a unique log format, the cost of Spark SQL consumer decreases when the partitioning of dataset increases, an illustrated in Figure 4. The X-axis represents the partition number as an integer and the Y-axis represents the time consumed (minute unit). We have to oversize the partitions and the gains are much less interesting.

2) *Model measurement*

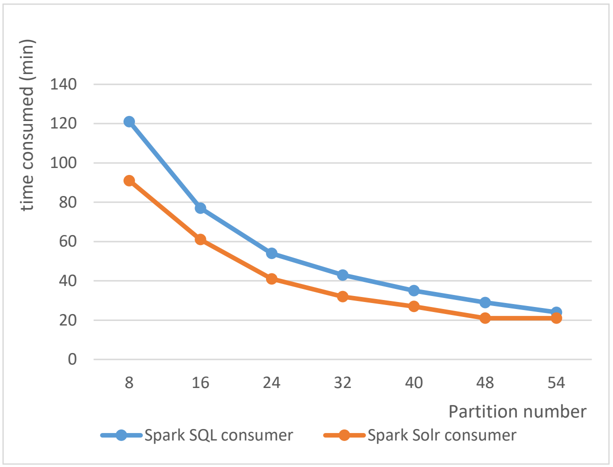

Figure 4. Spark consumer runtime versus number of partitions

SVM offers very high accuracy compared to other classifiers such as logistic regression, and trees. There are several modes of assessment. The first is technical; it is obtained thanks to the framework used for the development (Spark MLlib). The second is more empirical because it relates to the use of this model and the anomaly detection rate on a known dataset.

The analytical expressions of the features precision and recall of retrieved log messages that are relevant to the find are indicated below.

Precision is the fraction of retrieved log messages that are relevant to the find:

$$precision = \frac{|\{relevant\ \log messages\} \cap \{retrieved\ log\ messages\}|}{|\{retrieved\ \log messages\}|}$$

Recall is the fraction of log messages that are relevant to the query that are successfully retrieved:

$$recall = \frac{|\{relevant\ \log messages\} \cap \{retrieved\ log\ messages\}|}{|\{relevant\ \log messages\}|}$$

$$F_\beta = (1+\beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall}$$

In Table 1, we have four classes and for each class we compute three numbers: true positive (tp), false positive (fp) and false negative (fn). For instance, for the third class, we note these numbers tp3, fp3 and fn3. From these values, we compute precision by label, recall by label and F-score by label.

TABLE I. SVM MODEL MEASURES

Class number	Metrics		
	Precision by label	***Recall by label***	***F1 score by label***
0,000000	0.884615	0.920000	0.901961
1,000000	1.000000	1.000000	1.000000
2,000000	0.846154	0.785714	0.814815
3,000000	0.854462	0.7914858	0.842529

Our prediction models are similar to a multiclass classification. We have several possible anomaly classes or labels, and the concept of label-based metrics is useful in our case. Precision is the measure of accuracy on all labels. This is the number of times a class of anomaly has been correctly predicted (true positives) normalized by the number of data points. Label precision takes into account only one class and measures the number of times a specific label has been predicted correctly normalized by the number of times that label appears in the output. The last observations are:

- Weighted precision = 0.917402
- Weighted recall = 0.918033
- Weighted F1 score = 0.917318
- Weighted false positive rate = 0.043919

Our results for four classes are within acceptable ranges of values for the use of the model to be accepted.

The test empirical phase on the SVM model was not extensive enough to be conclusive; however, our results suggest that increasing the number of log patterns deteriorates the performance. In addition, we defined a finite set of log patterns for a targeted anomaly detection approach.

B. *Reporting*

We have created a custom data source to connect to Apache Solr, therefore, we are able to retrieve data and provide them back in following the *JRDataSource* interface of Jasper Report. With this access point, we have extracted metrics about the document cache and Query result cache. Both give an overview of the Solr activities and is meaningful for the analysts.

We have deployed the CData JDBC Driver on Jasper Reports to provide real-time HBase data access from reports. We have found that running the underlying query and getting the data to our report takes the most time. When we generate many pages per report, there is overhead to send that to the browser.

For the reporting phase, we have developed two report templates based on the use of a JDBC adapter. With system requests, we collect data about the last events (Get, Put, Scan, and Delete). From these HBase view, we have designed the report templates with cross tables. For the storage phase, we compute and display the number of Put events per timed window or grouped over a period.

We periodically updated the data across report runs and export the PDF files to the output repository where a web server manages them.

VII. CONCLUSION AND FUTURE WORK

We have presented our approach on log analysis and maintenance task prediction. We showed how an index engine is crucial for a suitable query engine. We have developed specific plugins for cutomizing the field types of our documents, but also for filtering the information from the log message.

Because indexing and storage are the two sides of our study, we have separated the storage into a Hadoop database. We have stressed the key role of our Spark components, one per data source. The partition management is the key concept for improving the performance of the Spark SQL component. The data storage into data frames during the micro batches is particularly suitable for the management of flows originating from Kafka files. We observed that our approach supported a large volume of logs.

From the filtered logs, we presented the construction of our SVM model based on work from the Center for Pattern Recognition and Data Analytics, Deakin University, (Australia). We were thus able to classify the recognized log patterns into classes of anomalies. This means that we can identify the associated maintenance operations. Finally, to measure the impact of our distributed analysis system, we wanted to automatically build reports based on templates and highlight indexing and storage activity.

Our study also shows the limits that we want to push back, such as the management of log patterns. The use of an AI model is not the guarantee of an optimal result. We want to make more use of indexing metrics to give more weight to some information in the analyzed logs. We are, therefore, thinking of improving the classification model of log data.

A first perspective will be to improve the indexing process based on a custom schema. We think that the use of DisMax query parser could be more suitable in log requests where messages are simple structured sentences. The similarity detection makes DisMax the appropriate query parser for short structured messages.

The log format has a deep impact on the Solr schema definition and about the anomaly detection. We are going to evolve our approach. In the future, we want to extract dynamically the log format instead of the use of a static definition.

We think also about malicious messages, which can perturb the indexing process and introduce bad requests in our prediction step. The challenge needs to manage a set of malicious patterns and the quarantine of some message sequences.

REFERENCES

[1] J. Andersson and U. Schwickerath, “Anomaly Detection in the Elasticsearch Service,” CERN Openlab Summer Student Report, pp. 1-18, 2019.

[2] T. S. Collett, “A review of well-log analysis techniques used to assess gas-hydrate-bearing reservoirs,” Natural Gas Hydrates : Occurrence, Distribution, and Detection, Geophysical Monograph, The AmericanGeophysical Union, vol. 124, pp. 189-210, 2001.

[3] Q. Fu, J. G. Lou, Y. Wang and J. Li, “Execution anomaly detection in distributed systems through unstructured log analysis,” 2009 ninth IEEE international conference on data mining IEEE, pp. 149-158, 2009.

[4] A. Oliner, A. Ganapathi and W. Xu, “Advances and challenges in log analysis,” Communications of the ACM, vol. 55, no. 2, pp. 55-61, 2012.

[5] C. Lim, N. Singh and S. Yajnik, “A log mining approach to failure analysis of enterprise telephony systems,” 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN), IEEE, pp. 398-403, 2008.

[6] J. Breier and J. Branišová, “Anomaly detection from log files using data mining techniques,” In Information Science and Applications, Springer, Berlin, Heidelberg, pp. 449-457, 2015.

[7] Li Tao et al., “FLAP : An end-to-end event log analysis platform for system management,” Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1547-1556, 2017.

[8] S. Yen and M. Moh, “Intelligent log analysis using machine and deep learning,” Machine Learning and Cognitive Science Applications in Cyber Security, IGI Global, pp. 154-189, 2019.

[9] B. Debnath et al., “Loglens : A real-time log analysis system,” 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), IEEE, pp. 1052-1062, 2018.

[10] M. Picquenot and P. Thébault, “GLPI (Free Management of Computer Park) : Installation and configuration of a park management solution and support center,” ENI Editions, 2016.

[11] K. Shvachko, H. Kuang, S. Radia and R. Chansler, “The hadoop distributed file system,” 2010 IEEE 26th symposium

on mass storage systems and technologies (MSST), IEEE, pp. 1-10, 2010.

[12] D. Smiley, E. Pugh, K. Parisa and M. Mitchell, “Apache Solr enterprise search server,” Packt Publishing Ltd, 2005.

[13] J. Kumar, “Apache Solr search patterns,” Packt Publishing Ltd, 2015.

[14] K. Koitzsch, “Advanced Search Techniques with Hadoop, Lucene, and Solr,” Pro Hadoop Data Analytics, Apress, Berkeley, CA, pp. 91-136, 2017.

[15] R. C. Maheshwar and D. Haritha, “Survey on high performance analytics of bigdata with apache spark,” 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), IEEE, pp. 721-725, 2016.

[16] M. F. Ghalwash, D. Ramljak and Z. Obradović, “Early classification of multivariate time series using a hybrid HMM/SVM model,” 2012 IEEE International Conference on Bioinformatics and Biomedicine, IEEE, pp. 1-6, 2012.

[17] F. E. N. G. Changyong et al., “Log-transformation and its implications for data analysis,” Shanghai archives of psychiatry, vol. 26, no. 2, 2014.

[18] M. Assefi, E. Behravesh, G. Liu and A. P. Tafti, “Big data machine learning using apache spark MLlib,” 2017 IEEE International Conference on Big Data (Big Data), IEEE, pp. 3492-3498. 2017.

[19] T. D. Nguyen, V. Nguyen, T. Le and D. Phung, “Distributed data augmented support vector machine on spark,” 2016 23rd International Conference on Pattern Recognition (ICPR), IEEE, pp. 498-503, 2016.