Real Time Analytic of SQL Queries Based on Log Analytic

Zirije Hasani, Boro Jakimovski, Margita Kon-Popovska, Goran Velinov

Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University in Skopje
Macedonia
zh12796@seeu.edu.mk, {boro.jakimovski,margita.kon-popovska,
goran.velinov}@finki.ukim.mk

Abstract. Analyzing huge amount of data are a big challenge. On one hand we are faced with the problem of storing a large amount of data, and on another to process it in a reasonable or even real time. Real time analytics can be defined as the capacity to use all available enterprise data and sources in the moment they arrive or happen in the system. In this paper, we present an infrastructure that we have implemented in order to analyze data from big log files in real time. The main components of the infrastructure are Redis, Logstash, Elasticsearch and Kibana. Redis is used for temporary buffering of the log data, Logstash utilizes different filters to manipulate and analyze the data, Elasticsearch is used for indexing and storing the data and Kibana is a user interface used to visualize the results. We explore implementation of several filters in order to post-process the log information and produce various statistics that suit our needs in analyzing log files containing SQL queries from a big national system in education. The post-processing of the SQL queries is mainly focused on preparing the log information in adequate format and information extraction. The purpose of the analysis is to monitor performance and detect unusual behavior in order to alert or prevent possible unwanted activities, or to develop (in future) triggers that can indicate or even prevent possible problems in real time.

Keywords: Big data, log data, real time processing, Redis, Logstash, Elasticsearch, Kibana.

1 Introduction

With the increased number of internet users, the need for analyzing data and specifically log data is increasing too. The two general requirements of big data projects are common: analysis of the (near) real time information extracted from a continuous inflow of data and persisting analysis of a massive volume of data. Log management is complex and time consuming process, even harder when we have to deal with big log files that came in real time. Log file is a file that records all the events that happened during one software or/and operating system is running. Also, it may register all the exchange of personal messages between different users under some communication software. The content of log files could be diverse, e.g. it could be structured, semi-structured and weakly structured. Our special interest is log files that contain SQL queries.

Building an infrastructure for analyze of big log files in real time is a computational, storage and scalability challenge. To make proper choice of infrastructure we have done extensive investigation reported in [7], [8], [9] and [10]. In this paper, we present adjusted infrastructure proposed by Ian Delahorne [2]. Among open source and free software tools, we find it appropriate because it is possible to modify it when needed, by adding various other components (like Hadoop), or scale up or down by adding (duplicate, triplicate,…) some of existing components.

Study and experiments are motivated by need to use such an infrastructure for analyze of the log files from a system called e-Dnevnik (ednevnik.edu.mk[1]). e-Dnevnik is an electronic system for managing the data records of students in Macedonian schools. System enables daily communication between teachers, parents and students and various statistical analyzes used by Ministry of education and research of RM and other public institutions. System receives a big number of requests during a day and analyze of these requests is required before they are saved to database in order to reduce the amount of logs that is necessary to be saved. The idea is to save into database just the information that is of interest for future processing and other to be ignored. Even more, analyze of log files in real time can signalize and detect errors, track CPU usage, monitor parameters and similar. If some of the parameters rise above expected values, or error occurs, built-in (in future) triggers will indicate or even prevent possible problems in real time.

Paper is structured as follows: In the second chapter we explain which are the components and functions of this infrastructure; in the third chapter, we demonstrate pre-processing of the SQL queries contained in log files and usage of several Logstash filters important for real time analytics; in the fourth chapter we implement the infrastructure for real time analysis of e-Dnevnik database log file; the last chapter is the conclusion for our work done thus far, including also ideas for future work.

2 Infrastructure for analyzing log data in real time

With aim to deal with big log files in real time produced by PostgreSQL server in order to analyze query performance, we start with the solution proposed by Ian Delahorne [2]. Since Elasticsearch [5], together with Logstash has evolved during the past several years, we include in our architecture several new and remove several unnecessary components. Our proposed architectural is shown in Figure 1. This architectural design is based on pipeline event processing, divided in following phases: input (collects and manages events and logs), buffering, decode/pre-process (extract structured data into variables, parse), filter (modify, extract information) and output (ship the data for storage, index, search and visualize).

An important characteristic of this architecture is the capability to scale up/out of every component, depending on the input stream size and rate, by running one or

[1] http://ednevnik.edu.mk/

more of its components as a separate threads/servers. For example, we can scale out the input phase with three shipper servers as shown in Figure 1, or we can scale up the Logstash filtering server on a bigger machine with more CPU cores/RAM.

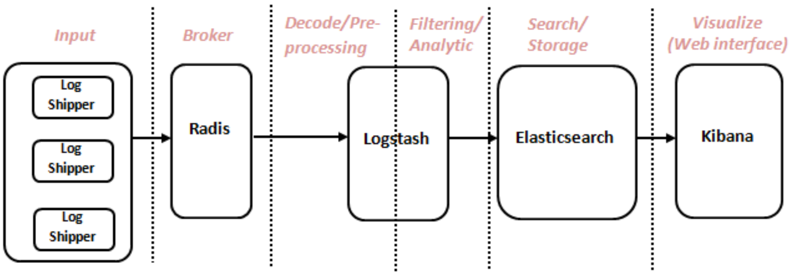

Fig. 1. Components of infrastructure for big data log analytic in real time

Flexibility is achieved also by possibility of adding various additional components as Hadoop, Cassandra, statistical or graphical tools like Statsd, Graphite and others. Generally in most cases when we run the Logstash server there will be two broad classes of Logstash host [3]:

- The first one is the host which runs the Logstash agent as an event "shipper" that sends application, service and host logs to a central Logstash server.
- The second one is central Logstash host which runs a combination of components of this architecture for pre-processing and filtering of events.

Broker (usually Redis [12]) acts as efficient temporary buffer for logs. This especially is important to enable interruptions in the processing of the log events in an occasion of upgrade process of the Logstash instances, or in the case of an unexpected raise of event size and number.

2.1 Logstash

The main component of the infrastructure is Logstash [4]. It is written in JRuby and runs in a Java Virtual Machine (JVM) [4]. It is easy to deploy, as a single JAR file that can be started directly using a JAVA SE VM (no Apache Tomcat Containers are needed). Its architecture is simple comparing with other similar software architectures since it consists of a three phase pipeline (input, filter, output) and it provides an easy way of extension of functionalities in each phase using plugins.

Input phase collects the logs and sends the collected events to the filter phase. Logs can generally arrive from various sources: Files, TCP/UDP files, Syslog, Microsoft Windows EventLogs, STDIN, Key-value stores and a variety of others. In our case log file includes Postgres SQL CSV log files and Key-value stores (Redis [12]).

Logstash comprise a large collection of filters which enable us to extract structured data into variables, parse, modify and enrich the data, before they are pushed to the Elasticsearch.

2.2 Elasticsearch / Kibana

Elasticsearch enables efficient indexing and storing of the event logs, and enables a full text search on them. It is an open-source distributed search engine library, built on top of Apache Lucene [11]. ElasticSearch [5] allows us to implement store, index and search functionality and as such help us in easier and more efficient computation of various data analytics. ElasticSearch is a NoSQL data store where data are stored as documents. Although it is mainly used by Java applications, the important thing is that applications need not to be written in Java in order to work with ElasticSearch, since it can send and receive data over HTTP in JSON to index, search, and manage our Elasticsearch cluster.

The last component is Kibana [6] which is a HTML/JS frontend web interface to Elasticsearch for viewing the log data. The beauty of Kibana is that we can easily search in the data with different queries, produce charts, histograms and other visual products [2].

3 Processing of SQL queries

Processing of database transaction logs presents a big challenge due to their massive volume. The main target of the SQL queries analytics is to gather information and detects anomalies in query performance on an operational level. This means that we want an early detection of performance degradation of SQL queries in real time and alert adequately in order to remove the possible causes.

3.1 Log data pre-processing

In order to get more realistic results we must do a SQL queries pre-processing by performing a normalization procedure on them. The normalization of the SQL queries tries to remove all data and parameters from the queries in order to gather better grouping/clustering of SQL query types. These includes elimination of comments, start of transactions, string content, null parameters, non essential numbers and hexadecimal numbers, the last line of code, removing of extra space, new line and tab characters and lower-casing. Similar normalization process can be referred in pgBadger [1] that is used for batch log file processing. Next is an example of Logstash configuration file for normalization of SQL queries using Logstash mutate filter. This filter allows performing of regular expression pattern matching and replacement for general transformation of event fields. Following is the piece of the Logstash filter configuration file for SQL query normalization:

```
mutate {
 # Set the entire query lowercase
 lowercase => [ "statement" ]
 gsub => [
  # Remove comments
  "statement", "\/\/\*(.*?)\*\/", "",
  # Remove extra space, new line and tab
  "statement", "[\t\s\r\n]+", " ",
  # Remove start of transaction
  "^\s*begin\s*;\s*/,"",
  # Remove string content
  "statement" , "\\'", "",
  "statement" , "'[^']*'", "''",
  "statement" , "''('')+", "''",
  # Remove NULL parameters
  "statement" , "=\s*null", "=''",
  # Remove numbers
  "statement" , "([^a-z_\$-])-?([0-9]+)","\ 10",
  # Remove hexadecimal numbers
  "statement" , "([^a-z_\$-])0x[0-9a-f]{1,10}", "\10x"
 ]
}
```

Other useful pre-processing plugging is the merge filter that lets us combine two events that occur within a period into a new single event. This can be helpful if information for a single SQL query is split into several log events. In our case, Postgres logs two events for a single query, first containing the SQL query, and the second containing the duration of the query execution. Merge plugin has the following options that are used:

- key => Unique identifier, used to match the two events you want to merge.
- order => 'first' or 'last', the order the events should arrive
- merge_tag => Tag(s) to add on the *new* event.
- period => Max length of time between events(seconds).

In the example below if the event is the first event to be merged we execute the following merge plugin. This can be controlled using conditional filter processing. The merging of events is based on the key values, i.e. in this case "session_id" and "session_line_num".

```
merge {
 key => [ "session_id", "session_line_num" ]
 order => 1
 period => 1
}
```

Finally for the second event, if that event contains the durraion of the SQL query, and matches the key fields "session_id" and "session_line_num", the event fields specified are merged. In this case we merge only the "duration" field.

```
merge {
key => [ "session_id", "session_line_num" ]
fields_to_merge => [ "duration" ]
 order => 2
}
```

At the end of pre-processing we remove the log message key-value for personal data protection, and further calculate a hash of the normalized SQL statement in order to optimize the analysis process so it will not involve the full complex SQL statements.

```
mutate {
 add_field => [ "sql_hash", "%{statement}" ]
 remove_field => [ "message" ]
}
anonymize {
 algorithm => "MD5"
 fields => [ "sql_hash" ]
 key => "<some seed>"
}
```

Analytics filter . In order to perform statistical analysis of the performance of SQL queries, we found that the use of the metrics filter [3] can be practical. The metric filter produces an aggregation metrics from the log events based on the selected key values. The metrics filter is invoked periodically (flush_interval), can filter the processed events based on a time frame (clear_interval) and can produce statistics of both event occurrence (count, rate of events) and event values (ex. sql statement duration). The timer parameter of the metrics filter gives us a variety of information as follows:

- “thing.count” - the total count of events
- “thing.rate_Xm” - the X-minute rate of events
- “thing.min” - the minimum value seen for this metric
- “thing.max” - the maximum value seen for this metric
- “thing.stddev” - the standard deviation for this metric
- “thing.mean” - the mean for this metric
- “thing.pXX” - the XXth percentile for this metric

Following is the Logstash configuration that uses the metrics filter in order to produce statistics on every 60 seconds, based on SQL events in the past 300 seconds. The statistics contain count, rate_1m and rate_5m for the event occurrence, and duration statistics per SQL query type (sql_hash). The statistics are produced as a separate log event.

```
metrics {
 add_tag => "metric"
 timer => [ "%{sql_hash}", "%{duration}" ]
 flush_interval => 60
 clear_interval => 300
 rates => [1,5]
}
```

Analyze of log files in real time can signalize and detect errors, track CPU usage, monitor parameters and similar. If some of parameters rise above expected values, or error occurs, built in (in future) triggers will indicate or even prevent possible problems in real time. Figure 2 shows how these metrics filters present results in Kibana.

0091e3390f763d542da76ea18d56717d.count	832
0091e3390f763d542da76ea18d56717d.rate_1m	4.50225627
0091e3390f763d542da76ea18d56717d.rate_5m	4.21929311
0091e3390f763d542da76ea18d56717d.min	0.008
0091e3390f763d542da76ea18d56717d.max	0.283
0091e3390f763d542da76ea18d56717d.mean	0.0620601
0091e3390f763d542da76ea18d56717d.stddev	0.20703094
0091e3390f763d542da76ea18d56717d.p1	0.009
0091e3390f763d542da76ea18d56717d.p5	0.009
0091e3390f763d542da76ea18d56717d.p10	0.01
0091e3390f763d542da76ea18d56717d.p90	0.113
0091e3390f763d542da76ea18d56717d.p95	0.124
0091e3390f763d542da76ea18d56717d.p99	0.15367
0091e3390f763d542da76ea18d56717d.p100	0.283

Fig. 2. Result fields of metrics (meter and timer) event in Kibana

4 Real time analysis of e-Dnevnik database log file

To illustrate possibilities of our infrastructure, we have analyzed log files generated from e-Dnevnik. e-Dnevnik is the electronic system for managing the student records of elementary and high schools in Macedonia. There is a huge number of requests in real time and we would like to take some statistics based on the traffic that is generated in a defined periods of time. The data analyzed are SQL queries saved in log file. In the Figure 3 below we present two histograms produced by Kibana. The first chart displays the distribution of the number of events in the system, calculated per 30 seconds intervals in the time period from 14:26 untill 15:16, having 2207880 hits all. The second chart shows the calculated mean duration of SQL queries execution time for the same period and intervals. This shows that the mean of the query duration is higher at the specific period of time. The higher mean duration time of SQL queries in this example is the consequence of the Postgres server restart and warming up of Postgres shared buffers.

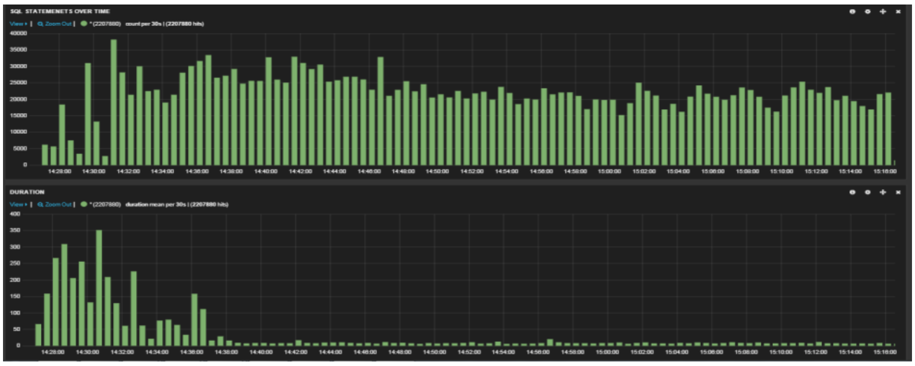

Fig. 3. e-Dnevnik number of hits and duration mean per 30 seconds intervals, in selected 14:26-15:16 period of time

4.1 Result Visualization

For now we didn't implement any specific tool or make our own, beside Kibana that will visualize the statistical data produced by the metric filter. In future work we plan to implement such a tool that visualizes result automatically. To visualize and compare results for various time intervals, and multiple parameters obtained by metrics filters in single picture, we export results in JSON format and import them in Excel. Figure 4, 5 and 6 present results acquired by the metrics filters count, rate_1m, rate_5m, min, max, mean, p1 (1 percentile), p5 (5 percentile), etc. Illustration is done for the five types of SQL queries. On the Figure 4 the 1 and 5 min rates of each query for the consecutive 1 min intervals in 10 min period are shown. Figures may indicate that the rates of query types 5 and 3 are slightly higher. Further investigation should be done (for their duration) and eventual optimization of these queries can be suggested.

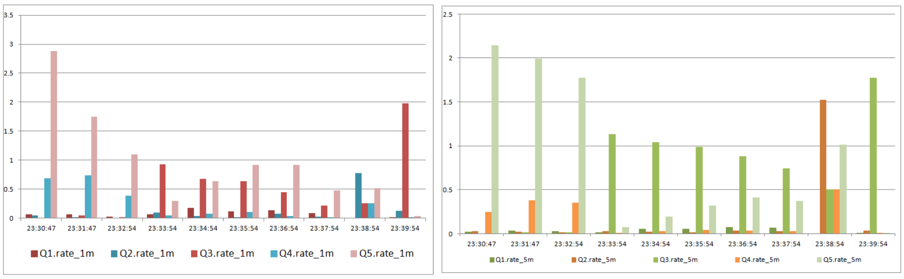

Fig. 4. Comparison of the 1 and 5 min rates for the five types of queries (parameters: rate_1m, rate_5m) for the consecutive 1 min intervals in 10 min period.

On the Figure 5 below the number of occurrences for the same five types of queries for the consecutive 1-min intervals in 10 min period is shown..

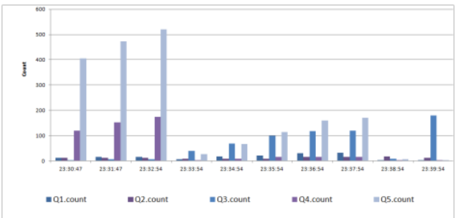

Fig. 5. Comparison of the number of occurrences for the five types of queries (parameter: count) for 1 min intervals in 10 min. period.

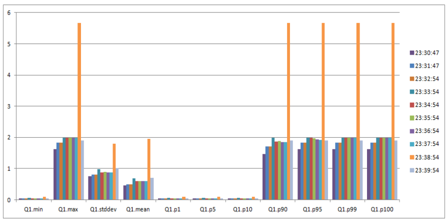

Fig. 6. Change of the several statistical parameters for the one type of query, for 1 min intervals in 10 min. period.

Figure 6 shows the change of the several statistical parameters for one type of query in 10 min period. The idea is to monitor these and other parameters in order to detect anomalies.

5 Conclusion

Analyzing Big Data in real time is a challenging process but the need for this analytics is emerging with enormous growth of incoming data and need of their fast analyze. In this paper, we propose infrastructure we have adjusted in order to analyze big log files in real time and demonstrate related analytics we made on system “e-Dnevnik” big log files that are produced daily by its PostgreSQL server.

The main components of the infrastructure are open source and free software tools, Redis, Logstash, Elasticsearch and Kibana. The infrastructure design is based on the pipeline event processing, divided in phases: input (collects and manages events and logs), buffering, decode/pre-process (extract structured data into variables, parse), filter (modify, extract information) and output (ship the data for storage, index, search and visualize). Proposed architecture is capable to scale up/out depending on the input stream size and rate, by running one or more of its components as separate threads/servers. Flexibility is achieved by possibility of adding various further components as Hadoop, Cassandra, statistical or graphical tools like Statsd, Graphite, or deploying extension of functionalities in each phase by using own plugins.

We illustrate the SQL queries database transaction logs analytics with implementation of the filters that produce various statistics enabling detections of anomalies in query performance on an operational level. This means that we are able to detect performance degradation of SQL queries in real time and alert adequately in order to remove the possible causes. In the same time in real time we do the pre-processing of the logs in order to reduce the amount of content of SQL queries that are necessary to be saved for further analyze.

In the future work we plan to extend the usage of available Logstash filters and include our own. The main goal will be to build-in triggers or similar mechanism that will automatically act in case of detected problems. We plan to extend the pre-processing of the incoming logs by parameterization of the SQL queries to lower further the volume of the stored data and to enable easier future analyses. Depending on the input stream of data we will experiment with scale up/out of the system components/servers and including other (batch appropriate) components as Hadoop and visualization tools.

References

1. pgBadger. Retrieved April 04, 2015, from http://sourceforge.net/projects/pgbadger/.
2. Ian Delahorne. Postgresql Metrics With Logstash. Retrieved April 04, 2015, from http://ian.delahorne.com/blog/2014/06/10/postgresql-metrics-pipeline
3. Logstash. Retrieved April 05, 2015, from http://logstash.net/docs/1.4.2/filters/metrics.
4. James Turnbull. *The Logstash Book Log management made easy*. January 26, 2014.
5. Radu Gheorghe and Matthew Lee Hinman. *Elasticsearch in action*. Manning Publications 2014.
6. Mitchell Anicas. How To Use Logstash and Kibana To Centralize Logs On Ubuntu 14.04. Retrieved April 06, 2015, from https://www.digitalocean.com/community/tutorials/how-to-use-logstash-and-kibana-to-centralize-and-visualize-logs-on-ubuntu-14-04.
7. Zirije Hasani, Margita Kon-Popovska, Goran Velinov. Survey of Technologies for Real Time Big Data Streams Analytic. 11th International Conference on Informatics and Information Technologies. April 11-13, 2014 – Bitola, Macedonia.
8. Zirije Hasani, Margita Kon-Popovska, Goran Velinov. Lambda Architecture for Real Time Big Data Analytic. ICT Innovations 2014 Web Proceedings ISSN 1857-7288
9. Zirije Hasani. Performance comparison throw running job in Hadoop by defining the number of maps and reduces. 12th International Conference on Informatics and Information Technologies 2015. April 24-26, 2015 – Bitola, Macedonia.
10. Zirije Hasani. Virtuoso, System for Saving Semantic Data. 12th International Conference on Informatics and Information Technologies 2015. April 24-26, 2015 – Bitola, Macedonia
11. Apache Lucena. Retrieved April 30, 2015, from https://lucene.apache.org/.
12. Redis. Retrieved April 30, 2015, from http://redis.io/.