Real Time Generalized Log File Management and Analysis using Pattern Matching and Dynamic Clustering

Bhupendra Moharil
Student, Computer Engineering
MIT College of Engineering
University of Pune, India

Chaitanya Gokhale
Student, Computer Engineering
MIT College of Engineering
University of Pune, India

Vijayendra Ghadge
Student, Computer Engineering
MIT College of Engineering
University of Pune, India

Pranav Tambvekar
Student, Computer Engineering
MIT College of Engineering
University of Pune, India

Sumitra Pundlik
Assistant Professor, Computer Department
MIT College of Engineering
University of Pune, India

Gaurav Rai
Senior Software Engineer
GS Lab Pvt. Ltd.
Pune, India

ABSTRACT

The past decade saw an exponential rise in the amount of information available on the World Wide Web. Almost every business organization today uses web based technology to wield its huge client base. Consequently, managing the large data and mining pertinent content has become the need of the hour. This is where the field of big data analytics sows its seeds. The linchpin for this is the process of knowledge discovery. Analyzing server logs and other data footprints, aggregated from clients, can facilitate the building of a concrete knowledge base. Querying the knowledge base can help supplement business and other managerial decisions.

The approach herein proposes a real time, generalized alternative to log file management and analysis. It incorporates the development of a sustainable platform which would enable the analysts to understand the essence of the data available.

General Terms:

Design, Experimentation

Keywords:

Log file, Levenshtein distance, Real time, Pattern matching, Dynamic clustering

1. INTRODUCTION

The main motivation for the project is the work done at Google Flu Trends by Jeremy Ginsberg et. Al. [6]. A few years back there was an outbreak of influenza like diseases. An exigency of finding the places of origin was felt in order to control the spread of the epidemics. However it was not feasible to visit each and every medical facility to collect statistics.To overcome this, the ideology of analyzing aggregated search data generated at the Google servers was put forth.

It was found that the number of patients suffering from influenza like diseases, visiting a medical facility, was directly proportional to the frequency of certain search queries fired to the server. Accordingly some keywords were identified and the source was traced based on the location from where the queries were fired. On similar lines, a log is another type of search data which can help mine huge amount of useful information.

1.1 Contents of a log file

Basically a log is a footprint of any activity performed by the user. Logs contain huge amount of information stored within them. If a highly productive environment is running then a large amount of logs would be produced per second. It is not feasible to manually read line by line and hence there is a requirement of a tool which can automate this process. Typically, a log line may include fields like IP Address, Timestamp, zone, http method, client request, protocol, status code, response size, execution time, memory consumed, or custom fields [4]. An example of an access log line is as shown in the Table 1

There are many other kinds of logs such as application logs, error logs, server logs, system logs, proxy server logs, gateway logs, database logs and other custom logs.

1.2 Requirement of real time platform

There are two types of data, one which has value at the given instance only and other whose value remains forever. For example, considering the scenario of a share market, if someone is able to predict the market spike based on the current data (stock) then it is beneficial. If the spike goes off before the prediction then the analysis becomes worthless. As opposed to this, analyses like top users, top product in demand, etcetera require historical data. Mining the instantaneously valued data requires a real time platform. In this paper, a method of dynamic pattern identification for logically clustering log data, has been proposed. The method is a real time and generalized solution to the process of log file management and analysis. A methodology to foster faster querying

Table 1. Sample log line

Field	Value
IP Address	192.168.2.20
Date	28/Mar/2014
Time with zone	10:27:10 -0300
Client request	”GET /cgi-bin/try/ HTTP/1.0”
Status code	200
Size in bytes	3395

on the data has been proposed. The paper covers the platform required for the implementation.

2. LITERATURE SURVEY

Since the last decade a lot of work has been done on developing business analytics solutions. Demiriz [1] has proposed webSPADE, a parallel sequence algorithm to analyze web log data. Peng Zhu et al. [13] have proposed a session identification algorithm for web log mining, which uses timing thresholds and makes dynamic adjustment to identify sessions. Zhixiang Chen et al. [19] have proposed two algorithms for frequent traversal path patterns from very large web logs. Both these algorithms employ suffix tree constructions. Jayathilake [5] has proposed a framework for multi-format log processing and a set of recurring procedures for automation of log analysis. Meichun Hsu [3] has proposed a system that supports multiple heterogeneous data store and query/ compute platforms (structured and unstructured), integrates real time analysis over streaming data and brings parallel computation closer to data. The platform presents a unified interface for applications to create and execute analytics data flows over a diverse collection of data engines. Karuna Joshi et al. [8] have designed a system which uses warehouses to analyze web log. The central idea behind all the above methodologies is a centralized data store combined with use case specific implementations. However, all of the methodologies introduce a lot of overhead when considered for real time applications. Hence a technique which reduces the overhead in processing of the logs has been proposed.

Along with the researches mentioned above, there are some log analytics software available in the market. A comparative study of these existing log analytics software was made based on some key parameters and the observations were as shown in Table 2.
The parameter, ’Customization’, in Table 2 pertains to module changes in the code. The platform proposed herein would enable the user to replace modules in the code with custom modules to have additional functionalities such as security (encryption) implemented. The parameter, ’Real time’ refers to log handling as and when the log is produced (however it includes network and buffer latencies). The parameter ’Generalized’ refers to the multiple type of log formats supported by the software. Considering the limitations of the tools mentioned in Table 2, the following objectives were identified for the platform proposed herein:

(1) Optimizing data analytics process
(2) Supplementing user with data analyses, survey and statistics collection at real time
(3) Providing a generalized system for analytics
(4) User friendly and customizable application
(5) Supporting applications with real time decision making on basis of live statistics

The following components were identified for log analysis process:

2.1 Log aggregation layer

The main requirement for the log analytics activity is a data collection utility. Most of the existing systems use a Software as a Service (SaaS) platform wherein the product is delivered as a Service to the clients. Logs are collected from the clients using either agent systems or by utilizing cloud storage facilities [15].

2.2 Orchestration and Execution layer

This phase deals with processing of the logs. For this, either batch or event stream processing may be used. Details regarding the same are mentioned in proposed methodology 3.

2.3 Log Stores

Most of the software store the data first and process later. But for real time data it is essential that data is processed first and stored later as real time data mostly resides in data pipelines (queues) and cache. Since real time data is considered, it is beneficial to process first and store later. Table 3 shows a comparative study of databases considered.
The various perspectives to data storage [15] with respect to real time data are as follows:

2.3.1 Perspective 1. SQL databases can be used to store data. However there is a limit on the amount of data that can be stored. Also maintaining the relational dependencies is an overhead. Hence it is not preferred for big data analytics.

2.3.2 Perspective 2. Another approach is to use NoSQL database to store data. Assuming one wants responses within a latency bound, has more data than can fit on a single machine and wants some reasonable guarantee or fault tolerance, there isn’t a single NoSQL backend that would satisfy these requirements. The alternative is to build a separate layer on the top of the data store for batching data. Thus it is advisable to have a live buffer for storing the temporary data for better availability.

2.3.3 Perspective 3. Directly index data using search engines.Maintaining the replication and consistency between the live buffer and NoSQL database as mentioned in perspective 2 is quite difficult and creates an extra overhead on the system. Hence it is more efficient to index the data using a search engine.

2.4 Visualization

Visualization of the results may be done as required, by developing User specific Interfaces for the platform.

3. PROPOSED METHODOLOGY

The proposed architecture as shown in Figure 1 has two main components, the client side and the server platform.

3.1 Client Side

At the Client side, the logs would be collected and sent to the Server side. The logs may either be tailed from plaintext files or pulled from the target application. For providing this functionality tools such as Fluentd, Flume and Scribe were analyzed. Flume may provide better support or compatibility only if one intends to use the Hadoop-family of products especially CDH (Cloudera Distribution for Hadoop) because it is written in Java.

Meanwhile, Fluentd [2] is backend-storage agnostic, has less memory footprint, and is easy to use and extend. Fluentd is

Table 2. Comparison of existing log analytics software

Parameter	Loggly[10]	Splunk [14]	SumoLogic [11]
Open Source	No	No	No
Whether Generalized	Choose log type first	Yes. Not restricted to logs alone.	Not all log types supported unless notified
Real time	Yes	Yes	Yes
Cluster analysis	No. Index based.	No. Index based	Yes
Software as a service	Yes	No	Yes
Data collection	Cloud, agentless	Cloud, agentless	Sumo logic collectors
Customization	Restricted	Restricted	Restricted

Table 3. Comparison of databases[12]

Parameter	MongoDB	Cassandra	Oracle	HBase	ElasticSearch
Major Users	Craigslist, Foursquare, Shutterfly	Facebook, Twitter, Digg	Acxiom, Allstate, Amazon, ATT	Facebook	Github, Mozilla
Storage Type	Document	Column	Relational database	Column	Indexing
Key Points	Retains SQL properties	Cross of BigTable, Dynamo.	Object relational data model	Huge data store	RESTful search
Language	C++	Java	C	Java	Java
Usage	Voting, banking, logging	E-Business Suite,	PeopleSoft, SAP Applications	Live messaging	Logging
Data Storage	Disk	Disk	Disk	Hadoop	Disk Storage

a log collector daemon for semi-structured JSON-based logs implemented in Ruby. It deals with machine-readable logs. It is simply a mechanism for receiving, buffering, and forwarding data to another destination and hence is lightweight as compared to Scribe. The lines of code in Scribe and Flume is also quite high as compared with Fluentd[2]. The ruby plugin functionality in Fluentd, also boosts the customizability. On the client side the Fluentd daemon would tail the logs and forward them to the Server side Fluentd daemon. The Fluentd tail plugin would pick up the log lines and pass them through a regular expression. The regular expression is as follows:

```
(?<log>((?<ip>((\d)+[.](\d)+[.](\d)+[.](\d)+)*)
(?<pre>(.)*)(?<times>([\s]|[:])(\d{2})[:]\(\d)+
[:](\d)+)(?<post>(.)*)))
```

The proposed platform would take in any type of log. To facilitate this, the regular expression was generalized. The regular expression contains optional fields such as IP address and timestamp. Data before the timestamp is the 'pre' part whereas data after the timestamp is the 'post' part. If these fields are not identified then the entire log would be passed into the pre part. If only ip address is recognized then rest of the part would be in pre part. If timestamp is identified, data before it would be in the 'pre' part and data after the timestamp would be in the 'post' part. The recognized fields along with the entire raw log line are combined into a single JSON (JavaScript Object Notation) object.

This JSON would be sent to the forward plugin. The forward plugin would send the data to other Fluentd daemons through port 24224. This completes the flow on the client side. Figure 2 shows the complete client to server flow.

3.2 Server Side

On the Server side, the Fluentd daemon is responsible for continuously listening on port 24224 for log lines sent from clients. The received lines are then pushed into a message queue. For the message queue, Apache Kafka and Kestrel were considered. Kestrel gives better support with Hadoop family products. As Hadoop products are not used, Apache Kafka [7] is used. Apache Kafka runs over a Zookeeper cluster. The Zookeeper cluster facilitates data distribution and offset storing. The log lines forwarded from Fluentd daemon are distributed over five Kafka partitions for better availability during parallel retrieval. For distributed parallel processing, Storm [16] [9] and Hadoop [17] were compared. Hadoop does batch processing. Hence, every time the topology has to wait for a batch of data to arrive. On top of it, the map-reduce jobs increase latency of the system. As opposed to it Storm introduces the concept of event stream processing. There are basically three component files in Storm viz. spout for picking up data, bolt for actual processing and the topology for control. The Storm topology runs over a Zookeeper cluster. Out of the machines in the cluster, one is appointed as the Nimbus host and the others are termed as Supervisors. The Nimbus host performs the work of distributing task and code amongst the Supervisors. The Storm topology is an 'always alive' topology.

The Storm spouts would pick up data from the Kafka partitions, in parallel, in a round-robin fashion. Each log line picked up from the message buffer would generate an event for the topology. A stream of such events is simultaneously picked up by the Storm spouts and forwarded to the bolts. Each log line would pass through three bolts in succession. The first bolt would be responsible for de-serializing the JSON object and identifying the fields from the line. The second bolt would then identify the cluster to which the log line belongs. The advantage of clustering is to narrow down the search. Logging produces 'big' data. Almost thousands of log lines are generated in a second. Instead of searching in thousands of gigabytes of log data, it would be advisable to search in smaller chunks of data. Using algorithms like K-Means Clustering creates overhead as the clusters needs to be recalculated every time a new line is entered. Hence, for identifying the cluster to which the log line belongs, Levenshtein distance, a string similarity metric is used. Levenshtein distance [18] basically gives the character distance between two strings. Instead of plainly using the character difference, the algorithm was extnded to give percentage character difference to avoid length dependency and also for giving us the threshold value to control the number of clusters. The process of cluster identification is as shown in the Figure 3 The cleaned log line would be matched with each pattern from the pattern database(the knowledge base) until a matching pattern is found. Based on the threshold value (by default 10, but can be changed as needed) it would be identified whether the line belongs to the same cluster or not. If no matching pattern is found, then the cleaned log line itself is put up as a pattern and is assigned a new id. In this

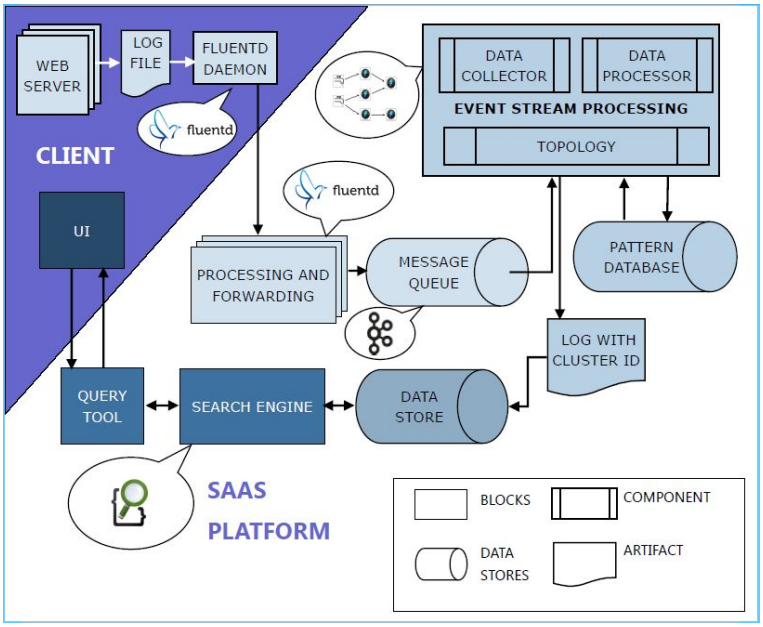

Fig. 1. Architectural Overview

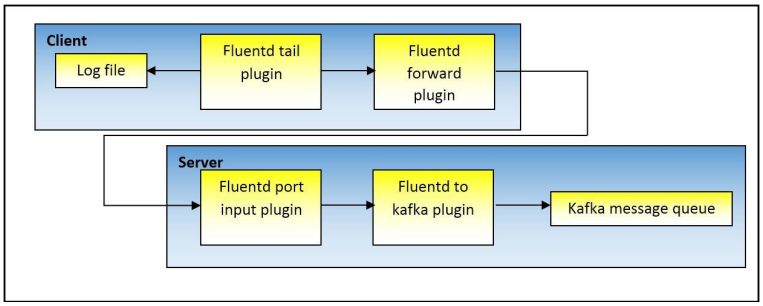

Fig. 2. Client to Server

Table 4. HBase versus ElasticSearch

Sr.No.	Number of lines	Time required	Data loss	Data store
1	1550	17.92	12	HBase
2	3100	28.92	16	HBase
3	1550	11.23	0	ElasticSearch
4	3100	20.14	0	ElasticSearch

The results were observed on a local topology. Data loss was due to the threads dying before acknowledgement.

manner the knowledge base goes on building. Lesser the threshold, more accurate the clusters formed. Hence it is advisable to keep threshold value lesser than 35.As the number of clusters go on increasing the performance of the algorithm might get degraded since every incoming log line is to be compared with all the existing patterns. As a solution to this the Levenshtein distance can be used as a metric to club(group) the clusters in order to reduce them. In this process the clusters with lesser inter-cluster distances are combined together. In this way the performance of the algorithm can be maintained or regained.The third bolt is responsible to store the log line along with the cluster id into the data store. Out of the databases in Table 3, consulting the database perspectives, HBase and ElasticSearch were identified as suitable candidates. The data was put into HBase as well as ElasticSearch. It was found that indexing data using ElasticSearch was faster than HBase writes. The same is shown in Table 4. Also ElasticSearch is provided with many inbuilt HTTP API's to foster faster search on the data. Hence ElasticSearch was chosen as the data store and also as a search engine.

The client may view and analyse his data from the User Interface provided. The User Interface is an NGINX hosted website having use cases like finding user navigation patterns, top users, location based user identification by map plotting, total logs and clusters in database and also ElasticSearch cluster overview. The user may also fire all the ElasticSearch queries on the Interface.

4. ANALYSIS

The processing topology was run locally and the results observed were as shown in the Table 5. The parameters considered for the test cases were:

(1) Number of executors and tasks: In order to achieve parallelism, Storm uses the concept of executors and tasks. The Nimbus

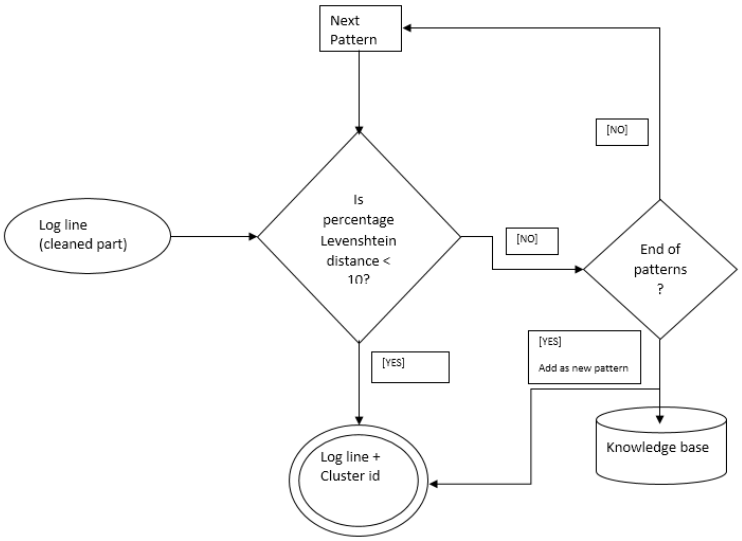

Fig. 3. Process of cluster identification

Table 5. If necessary, the tables can be extended both columns.

Sr. No.	Bolt1		Bolt2		Bolt3		Workers	Spouts	Lines tailed	Time(in sec)
	Executors	Tasks	Executors	Tasks	Executors	Tasks				
1	32	1	16	1	32	1	1	5	1550	22.3
2	32	1	16	1	32	1	1	5	3100	33.7
3	32	4	16	4	32	4	1	5	1550	12.9
4	32	4	16	4	32	4	1	5	3100	14.5
5	16	1	8	1	16	1	1	5	1550	18.6
6	16	1	8	1	16	1	1	5	3100	32.5

All the results were observed on a Storm local topology

Table 6. Observations

Allowed percentage Levenshtein distance (less than or equal)	Processing time	lines tailed	Clusters formed
75	6.42	1550	72
50	7.55	1550	225
25	8.26	1550	317
12	12.32	1550	553
6	18.38	1550	667

These results were found for Apache access logs.

host as described in the section , is responsible for code distribution across the Supervisors. The Supervisors have multiple parallel executors. Each executor is further assigned tasks. The tasks are serially executed.

(2) Workers: Number of Worker threads.

(3) Spouts: The number of spouts is dependent on the number of kafka partitions. Each spout is responsible for one partition.

(4) Lines tailed: The number of log lines handled in each test case.

(5) Time: Time required for the end to end flow when executed in Storm's local configuration.

The Table 6 compares the parameters, allowed percentage Levenshtein distance, number of clusters formed, line and time.

It was observed that the number of clusters formed is inversely proportional to the percentage similarity between the log lines. This confirms the fact that lesser the percentage Levenshtein distance more is the accuracy of clusters formed, that is, lesser clusters are formed.

5. CONCLUSION

The normal process of log information retreival includes store-process-store and then visualize, however, the proposed platform does preocess-store and directly visualize, thereby decreasing overall latency. Making use of the proposed platform, real time data analytics can be made possible on large data sets, thus facilitating prompt insights into the data. The platform can take in any type of log files, which gives it a generic capability to analyze different logs simultaneously. The platform can also be extended as a basemodule to serve other applications for decision making on the basis of real-time analytics. The clusters, which are created on the fly depending on the patterns identified, help shortening the search time. Thus the searching is done in a single cluster instead of the whole data store. This tremendously reduces the lag between request and response. As an added functionality the platform can be extended to serve any type of data other than logs, if the data matches the regular expression in Fluentd.

6. ACKNOWLEDGEMENT

This research work was undertaken as a part of the Bachelor of Engineering, final year project, under the guidance of GS Lab Pvt. Ltd. (www.gslab.com), Pune. The authors thank GS Lab Pvt. Ltd. for their valuable guidance and support.

7. REFERENCES

[1] Demiriz A. webspade: a parallel sequence mining algorithm to analyze web log. In *Proceedings of the International Conference on Data Mining*, pages 755–758. IEEE Proceedings 2002, 2003.

[2] Fluentd. Faq. http://docs.fluentd.org/articles.

[3] Meichun Hsu. Enabling real-time business intelligence. *6th International Workshop, BIRTC. Published by Springer. ISBN Print: 978-3-642-39871-1, ISBN Online: 978-3-642-39872-8*, pages 109–117, 2012.

[4] jafsoft. A web server log file sample explained, 2005. http://www.jafsoft.com/searchengines/logsample.html.

[5] Jayathilake. Towards structured log analysis. *IEEE International Joint Conference on Computer Science and software engineering*, pages 259–264, 2012.

[6] Rajan S. Patel Jeremy Ginsberg, Matthew H. Mohebbi. Detecting influenza epidemics using search engine data. *Nature*, 457, 2009.

[7] Apache Kafka. homepage. http://kafka.apache.org/.

[8] Yelena Yesha Karuna Joshi, Anupam Joshi. On using a warehouse to analyze web logs. *Distributed and Parallel Databases*, pages 161–180, 2003.

[9] G. E. D. S. Jonathan Leibiusky. Getting started with storm. O'Reilly Media.

[10] Loggly. How loggly works. http://www.loggly.com/product/how-loggly-works/.

[11] Sumo Logic. Meeting the challenge of big data log management: Sumo logic's real-time forensics and push analytics. Sumo Logic white paper.

[12] NoSQL. List of nosql databases, 2011. http://nosql-database.org/.

[13] Ming-Sheng Zhao Peng Zhu. Session identification algorithm for weg log mining. *IEEE International Conference on Management and Service Science(MASS)*, pages 1–4, 2010.

[14] Splunk. Home page. http://www.splunk.com/.

[15] Stackexchange. Perspectives on real time data. http://programmers.stackexchange.com/questions.

[16] Storm. home page. http://storm-project.net.

[17] Tom Whiter. Hadoop : The definitive guide, 3rd edition. O'Reilly Media.

[18] Wikipedia. Levenshtein distance. http://en.wikipedia.org/wiki/Levenshteindistance.

[19] Chunyue Weng Zhixiang Chen, Fowler R.H. Linear and sublinear time algorithms for mining frequent traversal path patterns from very large web logs. In *Proceedings of the Seventh International Conference, IEEE Database Engineering and Applications Symposium*, pages 117–122, 2003.