STREAMING LOG ANALYTICS TOOL

Nithin Veer Reddy Kankanti
University of Colorado Boulder
109540039
nithin.kankanti@colorado.edu

Janani Selvan
University of Colorado Boulder
109258879
janani.selvan@colorado.edu

Shruthi Sridharan
University of Colorado Boulder
109540345
shruthi.sridharan@colorado.edu

Veena Prasad
University of Colorado Boulder
109561006
vepr4844@colorado.edu

ABSTRACT

Logs are generated across multiple platforms mostly in the raw format. There are many types of logs generated but it is essential to analyze the error logs and rectify the faults for a system's successful flow. Most of the systems have their own style of log formats. Upon merging the logs, it becomes tedious to extract the important information. This creates a necessity to have a distinct pipeline for the log generator to merge these logs from various sources. Streaming Log Analytic Tool aims to create a log generator that extracts meaningful information from logs collected from multiple sources. To this end, this paper is divided into five sections. The first section discusses the introduction in brief. Then the paper describes the work previously done that is related to the Streaming Log Analytic Tool. The paper dives deep into the proposed work i.e., data set, architecture design and the modules involved in developing this tool. Then the paper sets forth the evaluation metrics and finally the milestones are provided.

KEYWORDS

log-analysis, logs, console-log, data sets, anomaly-detection, log-parsing, unstructured-logs

1 INTRODUCTION

Logs play an important role in the development and maintenance of software systems. It is a common practice to record detailed system runtime information into logs, allowing developers and support engineers to understand system behaviours and track down problems that may arise. The rich information and the pervasiveness of logs enable a wide variety of system management and diagnostic tasks, such as analyzing usage statistics, ensuring application security, identifying performance anomalies, and diagnosing errors and crashes. Despite the tremendous value buried in logs, how to analyze them effectively is still a great challenge.

2 RELATED WORK

[8] This paper presents LogDiver, a tool for the analysis of application-level resiliency in extreme-scale computing systems. The tool has been implemented to handle data generated by system monitoring tools in Blue Waters, the petascale machine in production at the University of Illinois' National Center for supercomputing applications. [11] It discusses about the complexities involved in automatic parsing of heterogenous logs as analysis tools prefer the logs in a unified format. It also presents an analysis to conclude that most, if not all, necessary information about a system for security and anomalies are available within logs if extracted properly. [12] Application logs contain a wealth of information that can be useful in aiding software system maintenance, and thus become an important data source for postmortem analysis. [9]- It proposes a fast, memory efficient, accurate, and scalable tool implemented in map-reduce framework for distributed platforms to process millions of heterogeneous log messages in seconds.

3 PROPOSED WORK

3.1 Architecture

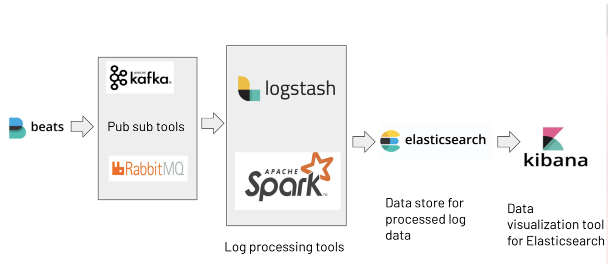


Figure 1: Architecture - Pipeline flow of logs

At uber level, the logs would be first collected from multiple resources such as github, open source logs etc. These logs are then processes to set of different pipelines which includes pub-sub tools such as rabbitmq, kafka etc which are divided based on the topic. These topics are based on the log category. These logs further move down to the batch processing tools such as spark, hadoop. Then the furnished logs would be dumped into elasticsearch where logs are indexed based on the log category. Elasticsearch [3] might further transform these logs into more meaningful categories. Lastly, the findings are projected onto kibana dashboard via multiple visualizations.

3.2 Components

Data collection (File watcher): -
We plan on using Beats[2], which is included in Elastic family that can be seamlessly integrated with Logstash, Elastic search and Kibana. Beats are essentially lightweight, purpose-built agents that acquire data and then feed it to Elasticsearch. The magic of Beats is the libbeat framework that makes it

easy to create customized beats for any type of data you'd like to send to Elasticsearch.

Message queue:
It enables asynchronous communication, which means that the endpoints that are producing and consuming messages interact with the queue, not each other. This allows the app scale seamlessly. Simply said; it is software where queues are defined, to which applications connect in order to transfer a message or messages. This message queue will act as an intermediary between the file watcher and log processing tool, where the file watcher adds messages of new log entry from log folders and waits for the log processing tool to consume the message and process the same.

Data processing (Log processing tool):
A critical aspect of a log that enables fast and accurate analysis is its structure. Recognizing the structure of a log greatly helps in easy extraction of specific system information, such as the type, time of creation, source of a specific event, the value of key performance indicators, etc. Without a known log structure, log analysis becomes a simple keyword based text search tool. So, this can be identified as the most important step of parsing the logs, identifying the kind of insights that we may have to derive from the logs and parsing the logs accordingly.

Data store:
We need a place to store the processed logs that will enable us to access them for our needs. These data stores need to provide faster search performance and more efficient data storage so as to add capabilities of building time-series charts and figures. So, identifying an effective choice that will satisfy our needs is an essential part of developing the tool.

Visualization tool:
This will be the last step of our pipeline where the processed metrics can be pulled up using beautiful visualization tools. This component should offer powerful and easy-to-use features such as histograms, line graphs, pie charts, heat maps etc.

3.3 Design Choices

Visualization tool: a) Kibana: Kibana is the 'K' in the ELK Stack, the world's most popular open source log analysis platform, and provides users with a tool for exploring, visualizing, and building dashboards on top of the log data stored in Elasticsearch clusters. Kibana's core feature is data querying and analysis. Using various methods, users can search the data indexed in Elasticsearch for specific events or strings within their data for root cause analysis and diagnostics. Based on these queries, users can use Kibana's visualization features which allow users to visualize data in a variety of different ways, using charts, tables, geographical maps and other types of visualizations. b) Grafana: Grafana is a free and open source (FOSS/OSS) visualization tool that can be used on top of a variety of different data stores but is most commonly used together with Graphite, InfluxDB, Prometheus, and Elasticsearch. As it so happens, Grafana began as a fork of Kibana, trying to supply support for metrics (a.k.a. monitoring) that Kibana (at the time) did not provide much if any such support for. Essentially, Grafana is a feature-rich replacement for Graphite-web, which helps users to easily create and edit dashboards. It contains a unique Graphite target parser that enables easy metric and function editing. Users can create comprehensive charts with smart axis formats (such as lines and points) as a result of Grafana's fast, client-side rendering — even over long ranges of time — that uses Flot as a default option.

Data store: We plan on using elasticsearch as the processed log data store. Elasticsearch allows you to store, search, and analyze huge volumes of data quickly and in near real-time and give back answers in milliseconds. It's able to achieve fast search responses because instead of searching the text directly, it searches an index. Since we were dealing with log data, that needed full-text search capabilities integrated into it, the obvious choice here was to go ahead with Elastic search. It is a free, open source software. We also have the options of running Elasticsearch on-premises, on Amazon EC2, or on Amazon Elasticsearch Service. Elastic search is also closely coupled with logstash in the ELK stack, which makes it easier to connect to the log processing tool - Logstash.

Log processing tool: a) Logstash: It has a larger footprint, but provides a broad array of input, filter, and output plugins for collecting, enriching, and transforming data from a variety of sources. Logstash helps you to collect data from multiple systems into a central system wherein data can be parsed and processed as required. Also, Logstash helps you to gather the data from multiple systems and store the data in a common format, which is easily used by Elasticsearch and Kibana. b) Apache spark: It is an excellent and ideal open source framework for wrangling, analyzing and modeling structured and unstructured data—at scale! It allows you to process, cheaply dump and store your logs into files on disk, while still providing rich APIs to perform data analysis at scale.

Message queue: a) Rabbitmq [6]: It is often summarized as an "open source distributed message broker." Written in Erlang, it facilitates the efficient delivery of messages in complex routing scenarios. Initially built around the popular AMQP protocol, it's also highly compatible with existing technologies, while its capabilities can be expanded through plug-ins enabled on the server. RabbitMQ brokers can be distributed and configured to be reliable in case of network or server failure. b) Apache kafka: It is described as a "distributed event streaming platform." Rather than focusing on flexible routing, it instead facilitates raw throughput. Written in Scala and Java, Kafka builds on the idea of a "distributed append-only log" where messages are written to the end of a log that's persisted to disk, and clients can choose where they begin reading from that log. Likewise, Kafka clusters can be distributed and clustered across multiple servers for a higher degree of availability.

3.4 Dataset Overview

This project proposes to use the collection of system logs maintained by LogHub [5]. The total size of the data set is approximately 77 Gigabytes and the website hosts only a small sample (2k lines) on GitHub for each data set [10]. Most of the data available are system generated logs and the logs are anonymized and sanitized to remove any confidential information at hand. The logs presented in the website has the data collected from distributed systems (HDFS, Hadoop, Spark, etc), Supercomputers (BGL, HPC, Thunderbird), Operating systems (Windows, Linux, Mac), Mobile systems (Android, HealthApp), Server applications (Apache, OpenSSH) and Standalone software (Proxifier) [10]. The data set is available at [4]: https://github.com/logpai/loghub.

4 ACCOMPLISHMENT

For this checkpoint we were able to successfully process one particular logset from the huge dataset that we have at hand. We were able to complete building one pipeline that would process logs end-to-end starting from the raw logs to visualization dashboards built on them. We set up infrastructure for the same on cloud and deployed it on Google Cloud Platform. The reason for us, choosing android logs is that, Mobile phones are ubiquitous today. The portable device has revolutionized the way we communicate with one another locally and globally. Hence, we decided to process android logs and derive insightful information out of these logs. We went about with mapping the raw logs to the android log format, which includes pre-processing the log severity(Info, Debug, Error etc) and other data attributes. This pipelines helped us parse roughly about 1.5 M logs are parsed so far. The framework that we have now can be easily adapted for other log types, which allowed us to push the formatted log data into elastic search. Kibana dashboards were built on top these Elastic search indices, enabling us to visualize the data better using time-series and other charts.

4.1 ELK - Accomplishment

The pipeline consisted of Elastic search for storing the processed logs, Logstash for processing/parsing the raw logs and Kibana for visualizing the same in meaningful dashboards. We processed 4 Major log groups - Android, Spark , Zookeeper, Hadoop logs end-to-end and putting down the specifics on the same in the table below.

Log type	# logs	Log file size (MB)	Processing time (in minutes)
Android	199799	192	270
Hadoop	180897	5.1	264
Zookeeper	74380	10	102
Spark	108291	2950	192

Figure 2: ELK run times
Run time numbers for 4 different log group processing

4.2 Spark - Accomplishment

The pipeline consisted of Elastic search for storing the processed logs, Apache Spark for processing/parsing the raw logs and Kibana for visualizing the same in meaningful dashboards. We processed 4 Major log groups - Android, Spark , Zookeeper, Hadoop logs end-to-end and putting down the specifics on the same in the table below.

Log type	# logs	Log file size(MB)	Processing time(in minutes)
Android	199799	192	297
Hadoop	180897	5.1	316
Zookeeper	74380	10	113
Spark	108291	2950	230

Figure 3: Apache Spark run times
Run time numbers for 4 different log group processing

5 OBSERVATION

Now that the load was not distributed across multiple servers, the simpler pipeline with Elastic search, Logstash and Kibana performed better. But, on doing an qualitative analysis on the runtimes when the applications scales, Apache Spark [7] would seem like a viable option. Ideally, its always better to go ahead with serverless architecture when it comes to big data processing over periodic events, where you pay for the only time your infrastructure was in use/live. While serverless cost is generally correlated with level of usage, there are architectural decisions that impact cost efficiency. The impact of these choices is more significant as your traffic grows, so it's important to consider the cost-effectiveness of different designs and patterns. But, when this pipeline needs to be adapted for log stream processing, where the servers need to be continuously running, Logstash would be a good choice, as we will need very less no of servers that would persist for a longer duration. In order to decide between the pipelines, we can do a simple cost benefit analysis with distributed/sequential load across servers. One other interesting observation was that the number of line items in the raw log files i.e the log count impacted the run times of these jobs over the actual log file size itself or the nature of logs. For example, we would be able to notice that Spark logs being almost 500x times larger in file size, it got processed faster, when compared to Hadoop logs. The reason here being the line item count in Hadoop logs where 1.6x times larger than that of the Spark logs.

6 EVALUATION

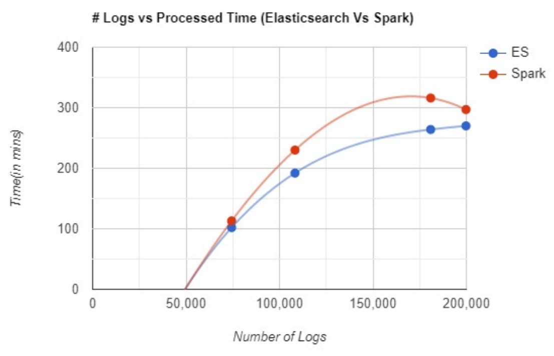


Figure 4: Graphical Comparison of ES and Spark
This graph represents the comparison of the logs processing time by Elastic search and Spark.

Rabbitmq vs Kafka: Kafka offers much higher performance than message brokers like RabbitMQ. It uses sequential disk I/O to boost performance, making it a suitable option for implementing queues. It can achieve high throughput (millions of messages per second) with limited resources, a necessity for big data use cases. RabbitMQ can also process a million messages per second but requires more resources (around 30 nodes). You can use RabbitMQ for many of the same use cases as Kafka, but you'll need to combine it with other tools like Apache Cassandra. For our usecase, where the log data can scale tremendously, its ideal to go about with Kafka. Also, one other pro's of going with Kafka is that many leading cloud service providers have an option of fully managed kafka service. Apache Kafka clusters are challenging to setup, scale, and manage in production. When you run Apache Kafka on your own, you need to provision servers, configure Apache Kafka manually, replace servers when they fail, orchestrate server patches and upgrades, architect the cluster for high availability, ensure data is durably stored and secured, setup monitoring and alarms, and carefully plan scaling events to support load changes. Managed Kafka service makes it easy for you to build and run production applications on Apache Kafka without needing Apache Kafka infrastructure management expertise. That means you spend less time managing infrastructure and more time building applications.

Kibana vs Grafana: The key difference between the two visualization tools stems from their purpose. Grafana's design for caters to analyzing and visualizing metrics such as system CPU, memory, disk and I/O utilization. The platform does not allow full-text data querying. Kibana, on the other hand, runs on top of Elasticsearch and is used primarily for analyzing log messages. If you are building a monitoring system, both can do the job pretty well, though there are still some differences that will be outlined below. If it's logs you're after, for any of the use cases that logs support — troubleshooting, forensics, development, security, Kibana is your only option. In this section, we plan to conduct a one-of-a-kind evaluation of certain metrics in terms of analyzing and comparing. The results will be evaluated and benchmarked to achieve robustness and efficiency. The aim is to achieve a higher performance with 3 representative log parsers to rank each component alongside with the power/time/cost and nature of the logs. These evaluations necessarily enlighten the characteristics of different representative log parsers to help in software development. The down-stream log analysis tasks include parsing textual log messages into a formatted text. This quality and reliability framework is expected to meet several research questions including anomaly detection, usage and performance analysis, fault and duplicate issue diagnosis. The streaming log analysis tool enables data mining of logs with improved searching, filtering and aggregation techniques.

Logstash vs Apache Spark: Apache spark offers rich APIs for data transformation making for very each to transform and prepare data in a distributed environment without worrying about memory issues. It's also provides faster in execution times compare to Hadoop and PIG Latin. But Logstash design is definitely perfect for the use case of ELK. Logstash has "drivers" using which it can inject from virtually any source. This takes the headache from source to implement those "drivers" to store data to ES. It is fast, very fast and can handle data in different shape, size, and formats. It lets you write simple rules to programmatically take decisions real-time on data. This lets you can change your data on the fly. The concept is similar to Kafka streams, the difference being the source and destination are application and ES respectively. So, it does seem like an option to go ahead with Logstash if the usecase demands Elastic search, as it can be easily integrated with it.

7 VISUALIZATION - INSIGHTS INTO THE LOG DATA

Visualizations are prepared on Kibana dashboard to provide valuable insights to rectify and help in software production. We present the charts from the kibana dashboards that we built for android, hadoop, spark and zookeeper logs and to derive insights by analyzing the underlying data.

We aimed at completing the implementation of streaming pipelines using Logstash and Spark based on the log categorization. Thereby, we ingested the standalone logs from the pipelines into the Elasticsearch. Pictorial representation of the insights from the processed logs are presented to understand any errors, irregularities using graphs, charts, word counts and reports.

The following graph depicts insights from the Hadoop processed logs.

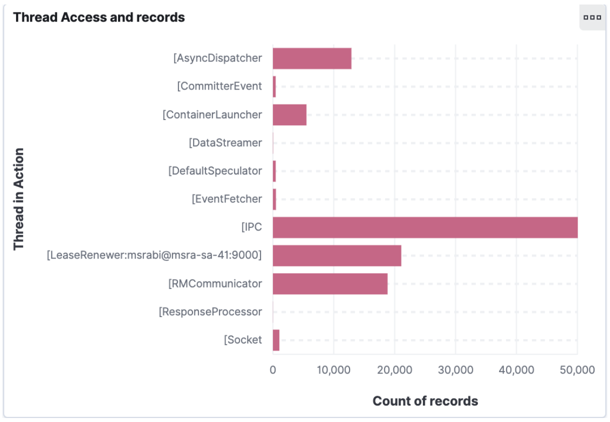
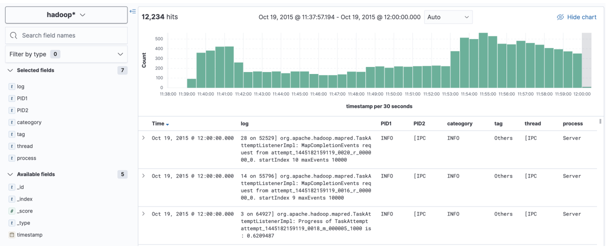


Figure 5: Hadoop Thread Access
This graph represents the Thread in Action and the count of records for each thread in Hadoop logs.

The above graph gives us a picture of various threads in action while Hadoop is processed and the number of records each thread logged. It is clearly visible that 'IPC' thread has hit the most number of records.



```
Time                        log                                        PID1   PID2   cateogory  tag     thread  process
> Oct 19, 2015 @ 12:00:00.000  28 on 52529] org.apache.hadoop.mapred.TaskA  INFO   [IPC   INFO       Others  [IPC    Server
                              ttemptListenerImpl: MapCompletionEvents req
                              uest from attempt_1445182159119_0020_r_0000
                              00_0. startIndex 10 maxEvents 10000
> Oct 19, 2015 @ 12:00:00.000  14 on 55796] org.apache.hadoop.mapred.TaskA  INFO   [IPC   INFO       Others  [IPC    Server
                              ttemptListenerImpl: MapCompletionEvents req
                              uest from attempt_1445182159119_0016_r_0000
                              00_0. startIndex 9 maxEvents 10000
> Oct 19, 2015 @ 12:00:00.000  3 on 64927] org.apache.hadoop.mapred.TaskAt  INFO   [IPC   INFO       Others  [IPC    Server
                              temptListenerImpl: Progress of TaskAttempt
                              attempt_1445182159119_0018_m_000005_1000 is
                              : 0.6209487
```

Figure 6: Hadoop Insights from Kibana Dashboard
This graph represents the variance of IPC thread's action along with its timestamp

Underlying data has to be analyzed to narrow down to the root cause as to why IPC had the highest count of records from Hadoop logs. By analyzing the above graph, it is very clear that the combinations of having PID2 as 'IPC', category as 'INFO' and server process has given us a huge hit in the number of records during the Map completion events.

The following graph depicts insights from the Android processed logs.

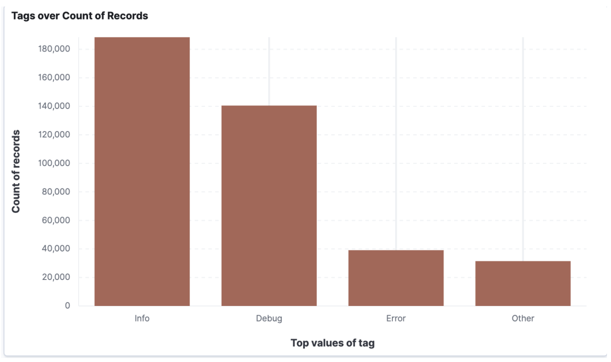
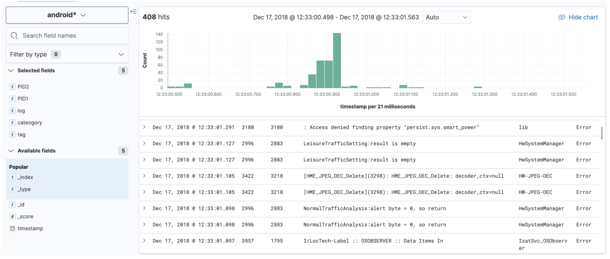


Figure 7: Severity of Android logs
This graph represents the distribution of categories in the Android processed logs.

The above graph gives us a picture of various log severity's through which logs are generated and shows the distribution of Android logs across various severity levels namely Info, Error, Warning, and Debug.



```
> Dec 17, 2018 @ 12:33:01.291  3180  3180  : Access denied finding property "persist.sys.smart_power"    lib               Error
> Dec 17, 2018 @ 12:33:01.127  2996  2883  LeisureTrafficSetting:result is empty                         HwSystemManager   Error
> Dec 17, 2018 @ 12:33:01.127  2996  2883  LeisureTrafficSetting:result is empty                         HwSystemManager   Error
> Dec 17, 2018 @ 12:33:01.105  3422  3218  [HME_JPEG_DEC_Delete](3298): HME_JPEG_DEC_Delete: decoder_ctx=null  HW-JPEG-DEC  Error
> Dec 17, 2018 @ 12:33:01.105  3422  3218  [HME_JPEG_DEC_Delete](3298): HME_JPEG_DEC_Delete: decoder_ctx=null  HW-JPEG-DEC  Error
> Dec 17, 2018 @ 12:33:01.098  2996  2883  NormalTrafficAnalysis:alert byte = 0, so return               HwSystemManager   Error
> Dec 17, 2018 @ 12:33:01.098  2996  2883  NormalTrafficAnalysis:alert byte = 0, so return               HwSystemManager   Error
> Dec 17, 2018 @ 12:33:01.097  3957  1795  I/LocTech-Label :: OSOBSERVER :: Data Items In                IzatSvc_OSObserv  Error
                                                                                                         er
```

Figure 8: Android Insights from Kibana Dashboard
This graph represents the timestamp and the description of each error in the android logs

Underlying data has to be analyzed to narrow down to the root cause and identify why those few error logs are attained. The various reasons for hitting a error log are listed in the image. If these errors are rectified, then the error count in Android logs can be significantly mitigated.

The following graph depicts insights from the Zookeeper processed logs. It also gives us a picture of the trend of various Zookeeper logs through which the timestamp of these generated logs shows that the Zookeeper Server is busy during the midnight.

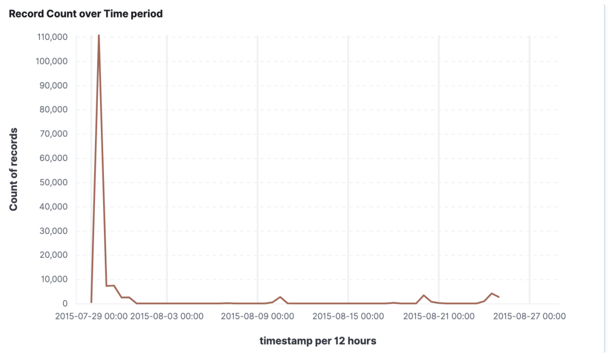
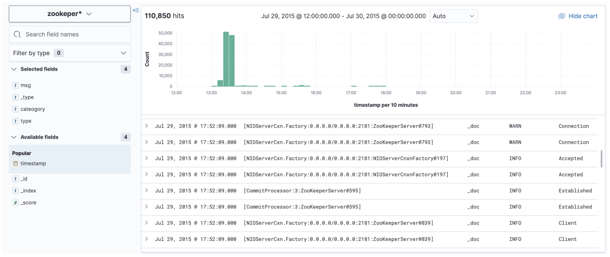


Figure 9: Record count in various timestamps for Zookeeper
This graph represents the spike in Zookeeper log count in various timestamps.



```
110,850 hits    Jul 29, 2015 @ 12:00:00.000 - Jul 30, 2015 @ 00:00:00.000    Auto

> Jul 29, 2015 @ 17:52:09.000  [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:ZooKeeperServer@793]        _doc  WARN  Connection
> Jul 29, 2015 @ 17:52:09.000  [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:ZooKeeperServer@793]        _doc  WARN  Connection
> Jul 29, 2015 @ 17:52:09.000  [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:NIOServerCnxnFactory@197]   _doc  INFO  Accepted
> Jul 29, 2015 @ 17:52:09.000  [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:NIOServerCnxnFactory@197]   _doc  INFO  Accepted
> Jul 29, 2015 @ 17:52:09.000  [CommitProcessor:3:ZooKeeperServer@595]                                _doc  INFO  Established
> Jul 29, 2015 @ 17:52:09.000  [CommitProcessor:3:ZooKeeperServer@595]                                _doc  INFO  Established
> Jul 29, 2015 @ 17:52:09.000  [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:ZooKeeperServer@839]        _doc  INFO  Client
> Jul 29, 2015 @ 17:52:09.000  [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:ZooKeeperServer@839]        _doc  INFO  Client
```

Figure 10: Zookeeper Insights from Kibana Dashboard
This graph represents the reason for the spike in midnight for zookeeper logs

Underlying data has to be analyzed to narrow down to the root cause and identify why maximum logs are attained by the Zookeeper server at mid-night and not otherwise. The various reasons for hitting the highest log count are listed in the image. It is clear that significant number of connection request are made to the Zookeeper server and they are successfully accepted and established during the interval from 13:00 to 16:00.

The following graph depicts insights from the Spark processed logs.

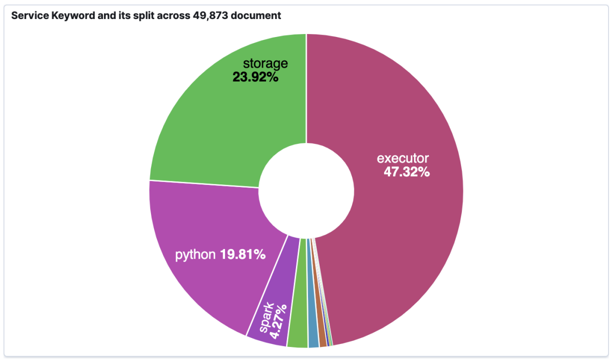
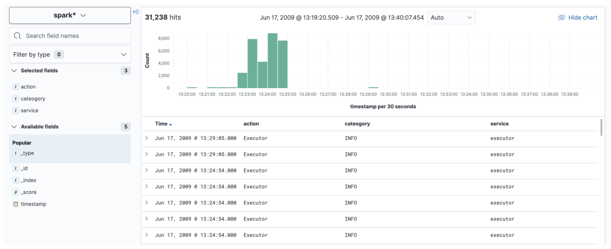


Figure 11: Record count in various timestamps for Spark
This graph represents the percentage of Services distribution in Spark logs.

The above graph gives us a picture of various services in action while Apache Spark is processed and the percentage of records each service logged. It is clearly visible that 'Executor' service has hit the most number of records.



```
31,238 hits    Jun 17, 2009 @ 13:19:20.509 - Jun 17, 2009 @ 13:40:07.454    Auto

  Time                          action     cateogory   service
> Jun 17, 2009 @ 13:29:05.000  Executor   INFO        executor
> Jun 17, 2009 @ 13:29:05.000  Executor   INFO        executor
> Jun 17, 2009 @ 13:24:54.000  Executor   INFO        executor
> Jun 17, 2009 @ 13:24:54.000  Executor   INFO        executor
> Jun 17, 2009 @ 13:24:54.000  Executor   INFO        executor
> Jun 17, 2009 @ 13:24:54.000  Executor   INFO        executor
> Jun 17, 2009 @ 13:24:54.000  Executor   INFO        executor
```

Figure 12: Spark Insights from Kibana Dashboard
This graph represents the huge chunk of logs generated at a particular timestamp for Apache Spark

Underlying data has to be analyzed to narrow down to the root cause as to why Executor had the highest count of records from Spark logs. By analyzing the above graph, it is very clear that the combinations of having action as 'Executor', category as 'INFO' and service "executor" process has given us a huge hit in the number of records around the timestamp 13:00.

7.1 Kibana Results and Snapshots

A collection of visualizations obtained by parsing 4 categories of logs: Android, Hadoop, Zookeeper, Spark were injected into Elastic Search and visualized using Kibana dashboard. Results and snapshots can be located in the Source code Zip submitted as part of submission and present in our project Git repository available at [1]

In this report, we have provided a detailed description of Android logs parsed, injected into Elastic Search and visualized using Kibana.

.

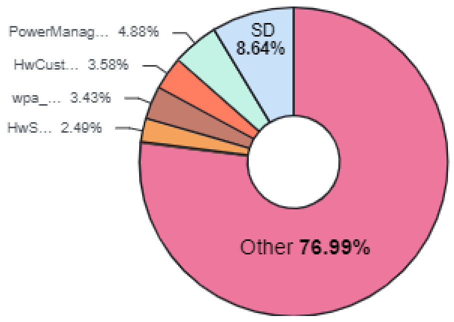


Figure 13: Android Category Chart
This graph represents the percentage of each category of android log types. The SD contributes to 8.84 percentage of logs compared which is 2 times higher than PowerManagement, and 3 times higher than wpa and HwS categories.

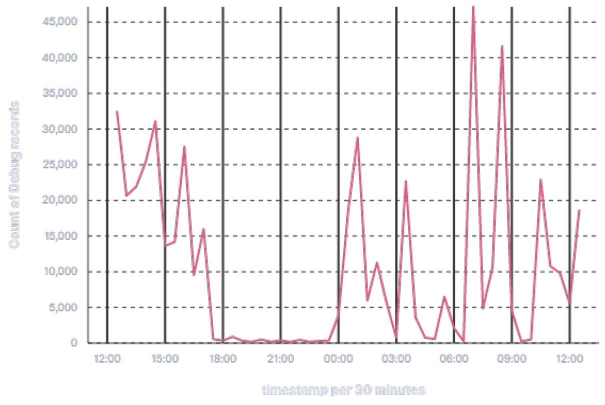


Figure 14: Android Debug Chart
The trend of the debug severity of each android log for a given time span can be observed using this chart.

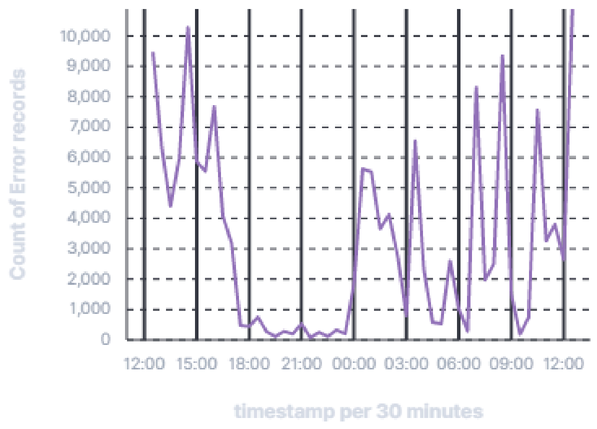


Figure 15: Android Error Chart
The various anomalies detected as part of the android log categories from the given time frame are represented here. The error severity of the android logs at the time of 15.00 is alarming.

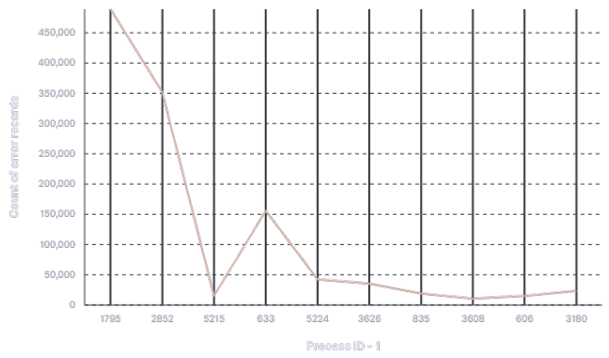


Figure 16: Android PID1 Chart
The number of logs whose severity is Error is contributed by the Process-ID1 1795, with a count of 488930 errors.

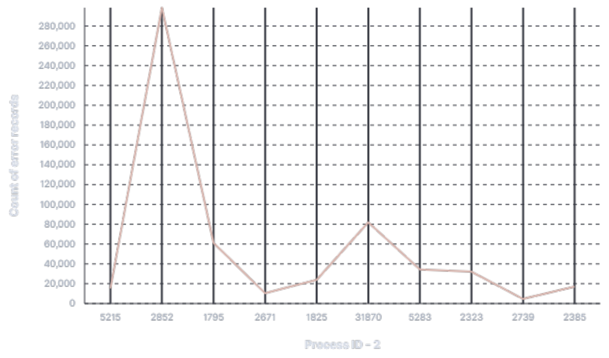


Figure 17: Android PID2 Chart
The number of logs whose severity is Error is contributed by the Process-ID2 2852, with a count of 298454 errors.

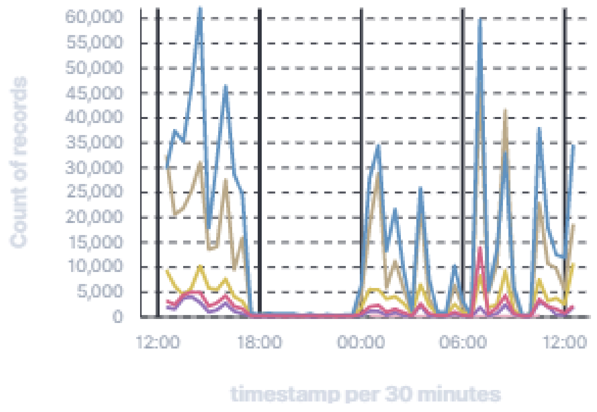


Figure 18: Android All Tag Chart
The trend of the severity of each android log for a certain time span can be observed using this chart.This indicates that monitoring is not required when the processes are not used between the time 18.00 and mid-night 12

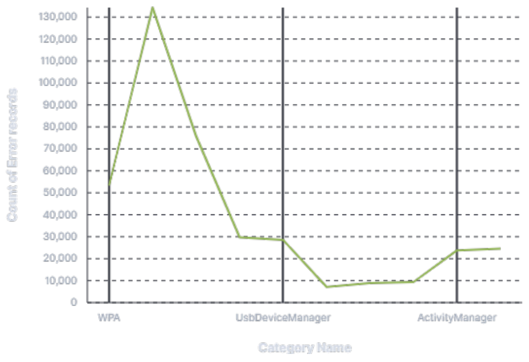


Figure 19: Android Tag Category Chart
The number of logs whose severity is Error is contributed by the SD android log category, with a count of 134307 errors.This is an indication to the developer to monitor SD android category more closely to avoid anomalies.

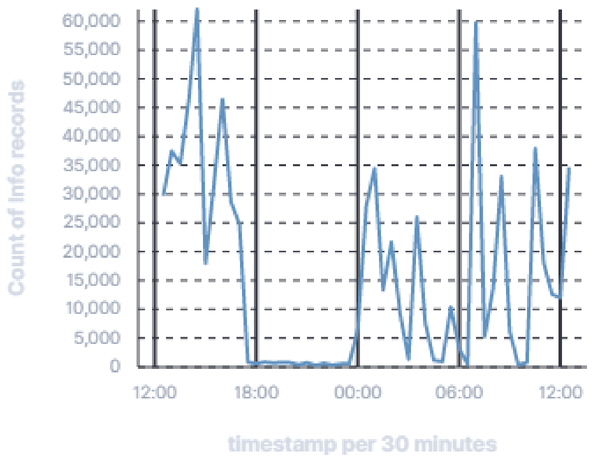


Figure 20: Android Tag Info Chart
The trend of the Info severity of each android log for a given time span can be observed using this chart.

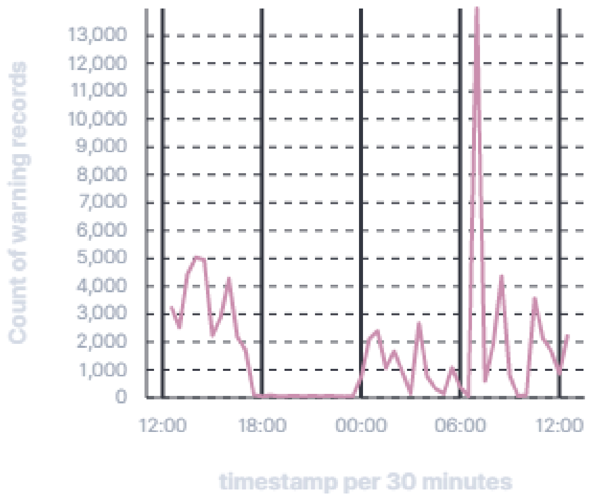


Figure 21: Android Warning Chart
The trend of the warning severity of each android log for a given time span can be observed using this chart.

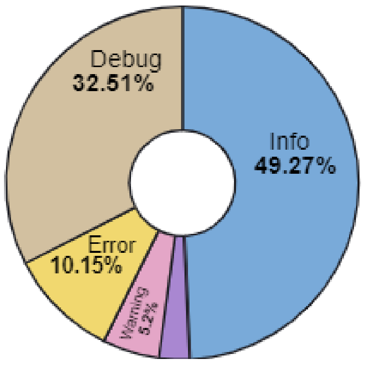


Figure 22: Android Tag Chart
This graph represents the percentage of each severity level of android log types. The Info contributes to 49.27 percentage of log severity and forms the highest followed by Debug with 32.51 percentage which is 3 times higher than Error, and 5 times higher than warning severity's.

8 CONCLUSIONS

To summarize it all, we set up experiments to analyze performance of ELK and Apache Spark based pipeline based on 3 V's (Volume, Velocity and Variety). We processed 4 different log groups end-to-end (Android, Spark , Zookeeper, Hadoop). We recorded processing time for these experiments and did a quantitative/qualitative analysis of the above runtimes and other aspects. On comparing the processing times of ELK and Apache Spark based pipelines, it can be seen that ELK has lesser processing time. One other interesting observation was that the number of line items impacted the runtimes over the actual file size or nature of logs. On qualitative aspects of growing the application and bringing a scalable solution, Apache spark seems like a better option.

9 FUTURE WORK

We plan on building the entire pipeline that would support streaming data using the components listed above in the proposed work section. This will essentially include a data collection(file watcher) - watches a folder or set of folders for new or updated files. When it finds one, it passes the filename to a script that you designate or a message queue - to perform whatever action you desire. File Watcher allows all user settings to be saved and opened so that the same tasks can be repeated in the future without re-keying.

We assume since this is going to be a distributed framework, having a message queue such as Rabbitmq or Kafka would help us streamline the newly found data items to the actual log processing unit. This will enable us in collecting and delivering high volumes of log data with low latency. The pipeline will include the actual log processing unit that would typically be based on a big data platform - Apache Spark. Apache spark is fast because computations are carried out in memory and stored there. Thus there is no picture of I/O operations as discussed in the Hadoop architecture.

After processing data, processed log gets pushed into Elastic search. Elasticsearch has become the de facto standard open source logging database. Kibana provides a browser-based interface to logs and other time-associated data in Elasticsearch. This will help us get answers in real time: Stream your data into Elasticsearch and use Kibana to explore events in real time. No need to predefine your questions. It also provides powerful search syntax: Slice and dice your data on the fly - no complicated setup process required. Point-and-click queries opens Kibana up to your whole organization.

REFERENCES

[1] [n.d.]. BDA. https://github.com/nithinveer/BDAProject.
[2] [n.d.]. Beats. https://www.elastic.co/beats/.
[3] [n.d.]. Elastic. https://www.elastic.co/.
[4] [n.d.]. LogHub. https://github.com/logpai/loghub.
[5] [n.d.]. Logpai. https://logpai.github.io.
[6] [n.d.]. rabbitmq. https://www.elastic.co/guide/en/logstash/current/plugins-inputs-rabbitmq.html.
[7] [n.d.]. spark. https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html.
[8] Catello Di Martino, Saurabh Jha, William Kramer, and Ravishankar Iyer. 2015. LogDiver. 11–18. https://doi.org/10.1145/2751504.2751511
[9] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. LogMine: Fast Pattern Recognition for Log Analytics. https://doi.org/10.1145/2983323.2983358
[10] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. 2020. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. arXiv:2008.06448 [cs.SE]
[11] Andriy Miranskyy, Abdelwahab Hamou-Lhadj, Enzo Cialini, and Alf Larsson. 2016. Operational Log Analysis for Big Data Systems: Challenges and Solutions. *IEEE Software* 33 (01 2016), 1–1. https://doi.org/10.1109/MS.2016.33
[12] Adam Oliner, Archana Ganapathi, and Wei Xu. 2012. Advances and Challenges in Log Analysis. , 55-61 pages. https://doi.org/10.1145/2076796.2082137

A HONOR CODE PLEDGE

On our honor, as the University of Colorado Boulder students, we have neither given nor received unauthorized assistance.