HDFS Log Analysis using ELK Stack

Aarya Parikh
Concordia University
Montreal, Canada
aarya.parikh@mail.concordia.ca

Avaneesh Kanshi
Concordia University
Montreal, Canada
avaneesh.kanshi@mail.concordia.ca

Apurba Das
Concordia University
Montreal, Canada
apurba.das@mail.concordia.ca

Pankaj Sharma
Concordia University
Montreal, Canada
pankaj.sharma.20241@mail.concordia.ca

Smridhi Verma
Concordia University
Montreal, Canada
smridhi.verma@concordia.ca

ABSTRACT

The idea of this project is to perform log analysis of the Hadoop Distributed File System (HDFS). This analysis is critical for understanding system performance and diagnosing issues on a large scale, and ensuring operational efficiency. The proposed system leverages the Elasticsearch, Logstash, Kibana (ELK) stack to ingest, process, and visualize HDFS logs, enabling comprehensive insights into system behavior. Moreover, it makes use of Nginx and Redis for query routing and asynchronous processing, respectively. This approach helps to improve horizontal scalability for increasing users and make our system highly fault tolerant to database failures. This report details the methodology, architecture, and results achieved, demonstrating the effectiveness of ELK in large-scale log analysis.

HDFS Log Analysis Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/includeavaneesh/HDFS-Log-Analysis-using-ELK[1]

1 INTRODUCTION

Across a distributed network, the Hadoop Distributed File System (HDFS) is essential for controlling the processing and storing of large datasets. It creates numerous logs that document a wide range of actions, including file operations, block-level processes, and system diagnostics. System administrators need these logs in order to identify malfunctions, maximize resource use, and preserve high availability.

However, manual analysis is impractical due to the enormous volume of HDFS data. We use the ELK stack, an open-source framework for large-scale log management, to solve this. The ELK stack provides a scalable solution for log analysis by enabling automated log collecting, processing, and visualization.

The architecture and methodology for integrating HDFS log analysis with the ELK stack are described in this report. We concentrate on developing visualizations in Kibana and building Logstash pipelines for analyzing different kinds of HDFS logs. We also demonstrate how the scalability, concurrency, and fault-tolerance characteristics of the ELK stack enhance the robustness and effectiveness of the log analysis system through the use of a distributed system design. This method's effectiveness is illustrated using real-world datasets from TraceBench and Loghub.

2 ABOUT THE DATASET

The dataset used in this study is from Loghub [2, 5], which is a large collection of system log datasets for AI-driven log analytics. Additionally, TraceBench [3, 4] provided valuable resources for trace-oriented monitoring. Specifically, we used HDFS_v3_TraceBench for our project.

TraceBench is an open data set for trace-oriented monitoring, collected using MTracer on a HDFS system deployed in a real IaaS environment. When collecting, it considered different scenarios, involving multiple scales of clusters, different kinds of user requests, various speeds of workloads, etc. In addition to recording the traces when the HDFS runs normally, it also collected the traces under the situations with various faults injected. There are 17 faults injected, including function and performance faults (and real system bugs). The total collection time of TraceBench is more than 180 hours, resulting 364 files that record more than 370,000 traces.

The data was collected through instrumenting the HDFS system. For our project and report purposes, we are using trace, operation, edge and event csv files.

3 SYSTEM ARCHITECTURE

The system architecture makes use of an NGINX load balancer for fault tolerance and effective query routing, as well as Docker containers to deploy the ELK stack components (Elasticsearch, Logstash, and Kibana). High-throughput data processing can be handled by the architecture while preserving system dependability and speed.

- **Elasticsearch Cluster:** To store, index, and query data, the system uses a distributed database method. One node is initially chosen at random to be the master node in this master-slave architecture. Furthermore, the other nodes serve as replicas and are slave nodes. All write operations are handled by the master node. The master node receives any write requests sent to slave nodes. Any node, though, is capable of performing read operations. This architecture avoids split-brain issues and increases failure tolerance and data availability. The slaves use consensus to choose the new master node in the event that the current one fails. But only a maximum of (N/2) + 1 nodes can stop the system from working at a certain moment.

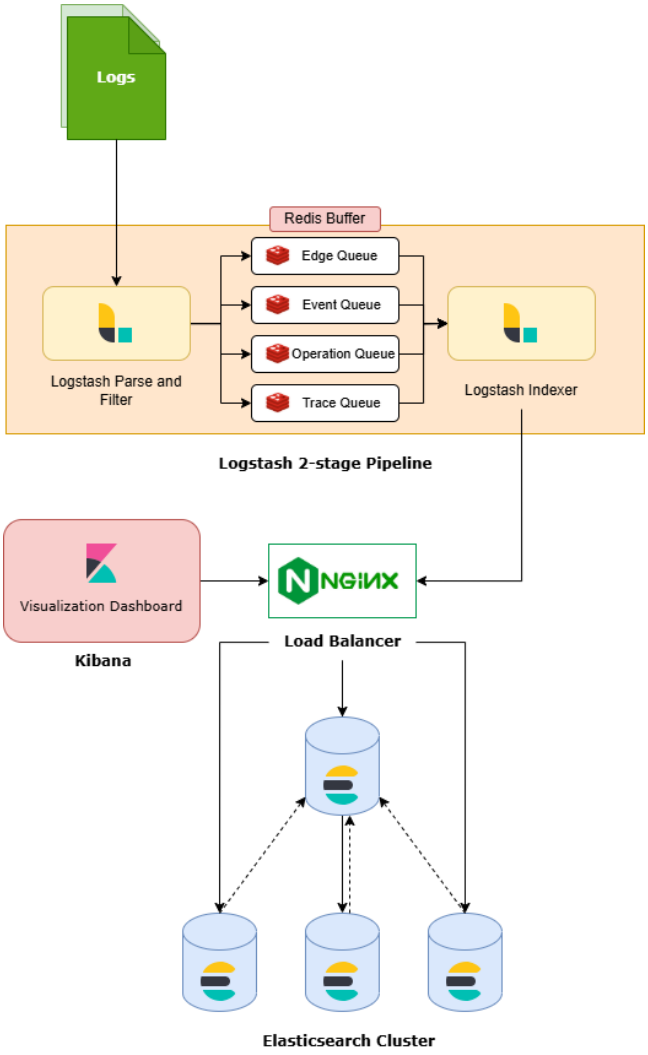

Figure 1: System Design for Distributed ELK

- **Logstash Pipelines:** This consists of a 2 stage pipeline, where the first stage is responsible for parsing and filtering raw input .csv files. Upon processing the log data, it is transferred to the respective message queues hosted on a dedicated Redis service. The second stage of the logstash pipeline pulls the subscribed messages from the buffer and feeds it into the Elasticsearch cluster. These pipelines are configured to handle various types of HDFS logs—edge logs, event logs, operation logs, and trace logs. Each log type is parsed and enriched with relevant metadata, such as timestamps and operation details, before being indexed in Elasticsearch.

- **Redis Buffer:** This service acts as a middleware in the Logstash pipeline. The implementation provides 4 message queues dedicated to handle a certain type of log to provide data separability and asynchronous transfer. The parsed data is fed into the Redis buffer and segregated into message queues to prevent any blocking of data. The Logstash service can pull the required data asynchronously as required without waiting for other tasks to finish.

- **Kibana:** The architecture's visualization layer is supplied by Kibana. Through a range of configurable dashboards, it enables users to engage with the data kept in Elasticsearch. Kibana visualizations used in this project include event distribution pie charts, delay analysis bar charts, and task duration histograms.

- **NGINX Load Balancer:** To distribute Kibana queries to the relevant Elasticsearch nodes, NGINX serves as a load balancer. By doing this, bottlenecks are avoided and system responsiveness is increased because no single node is overloaded with requests.

4 RESULTS AND DISCUSSION

Our project demonstrates a distributed system because it showcases multiple independent nodes working together to provide a unified service.

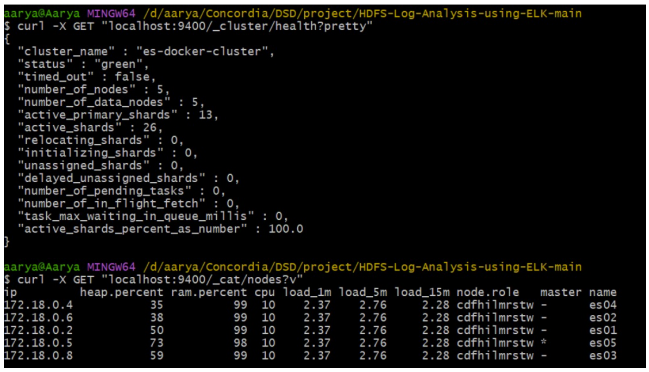

Figure 2: Demonstration of the Distributed System Design in our project

Explanation of the system's design:

- There are 5 nodes (es01, es02, es03, es04, es05) in the cluster.
- Each node has specific resources (CPU, RAM, and disk) contributing to the cluster's overall capacity.
- Elasticsearch distributes data (shards) and tasks across these nodes, allowing the system to handle larger datasets and higher query loads than a single machine could.
- The status: green indicates that all primary and replica shards are active.
- Replication (via active replicas) ensures that if one node fails, the data remains accessible from another node.
- active_primary_shards: 13 and active_shards: 26 indicate that the data is split into 13 primary shards and 13 replica shards.
- Each shard is stored and managed on different nodes in the cluster.

We also tried intentionally failing some nodes, but our service still worked even after 2/5 nodes failed, which is what it is expected to do.

```
aarya@Aarya MINGW64 /d/aarya/Concordia/DSD/project/HDFS-Log-Analysis-using-ELK-main
$ docker-compose stop es02
time="2024-12-15T20:29:33-05:00" level=warning msg="D:\\aarya\\Concordia\\DSD\\project\\HDF
S-Log-Analysis-using-ELK-main\\docker-compose.yml: the attribute `version` is obsolete, it
will be ignored, please remove it to avoid potential confusion"
[+] Stopping 1/1
 ✓ Container es02  Stopped                                                    1.5s

aarya@Aarya MINGW64 /d/aarya/Concordia/DSD/project/HDFS-Log-Analysis-using-ELK-main
$ curl -X GET "localhost:9400/_cluster/health?pretty"
{
  "cluster_name" : "es-docker-cluster",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 4,
  "number_of_data_nodes" : 4,
  "active_primary_shards" : 13,
  "active_shards" : 21,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 5,
  "delayed_unassigned_shards" : 5,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 80.76923076923077
}

aarya@Aarya MINGW64 /d/aarya/Concordia/DSD/project/HDFS-Log-Analysis-using-ELK-main
$ curl -X GET "localhost:9400/_cluster/health?pretty"
{
  "cluster_name" : "es-docker-cluster",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 4,
  "number_of_data_nodes" : 4,
  "active_primary_shards" : 13,
  "active_shards" : 26,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0
}

aarya@Aarya MINGW64 /d/aarya/Concordia/DSD/project/HDFS-Log-Analysis-using-ELK-main
$ curl -X GET "localhost:9400/_cat/nodes?v"
ip         heap.percent ram.percent cpu load_1m load_5m load_15m node.role   master name
172.18.0.4           55          94   6    1.90    2.42     2.21 cdfhilmrstw -      es04
172.18.0.2           71          94   6    1.90    2.42     2.21 cdfhilmrstw -      es01
172.18.0.5           55          94   6    1.90    2.42     2.21 cdfhilmrstw *      es05
172.18.0.8           70          94   6    1.90    2.42     2.21 cdfhilmrstw -      es03
```

Figure 3: Intentionally failing some of the nodes

```
aarya@Aarya MINGW64 /d/aarya/Concordia/DSD/project/HDFS-Log-Analysis-using-ELK-main
$ docker-compose stop es05
time="2024-12-15T20:31:33-05:00" level=warning msg="D:\\aarya\\Concordia\\DSD\\project\\HDF
S-Log-Analysis-using-ELK-main\\docker-compose.yml: the attribute `version` is obsolete, it
will be ignored, please remove it to avoid potential confusion"
[+] Stopping 1/1
 ✓ Container es05  Stopped                                                    1.1s

aarya@Aarya MINGW64 /d/aarya/Concordia/DSD/project/HDFS-Log-Analysis-using-ELK-main
$ curl -X GET "localhost:9400/_cluster/health?pretty"
{
  "cluster_name" : "es-docker-cluster",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 3,
  "number_of_data_nodes" : 3,
  "active_primary_shards" : 13,
  "active_shards" : 19,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 7,
  "delayed_unassigned_shards" : 7,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 73.07692307692307
}

aarya@Aarya MINGW64 /d/aarya/Concordia/DSD/project/HDFS-Log-Analysis-using-ELK-main
$ curl -X GET "localhost:9400/_cat/nodes?v"
ip         heap.percent ram.percent cpu load_1m load_5m load_15m node.role   master name
172.18.0.4           74          83   7    2.19    2.43     2.22 cdfhilmrstw -      es04
172.18.0.2           58          83   7    2.19    2.43     2.22 cdfhilmrstw *      es01
172.18.0.8           37          83   7    2.19    2.43     2.22 cdfhilmrstw -      es03

aarya@Aarya MINGW64 /d/aarya/Concordia/DSD/project/HDFS-Log-Analysis-using-ELK-main
$ curl -X POST "localhost:9400/test-index/_doc" -H 'Content-Type: application/json' -d'
{
  "test_field": "This is a test after node failure"
}'
{"_index":"test-index","_type":"_doc","_id":"ewoYzZMBjkJaQMuUEwzD","_version":1,"result":"c
reated","_shards":{"total":2,"successful":2,"failed":0},"_seq_no":0,"_primary_term":1}
```

Figure 4: Proof of system functioning even after some of the nodes failed

Figure 5: Proof of failure data reflection on kibana

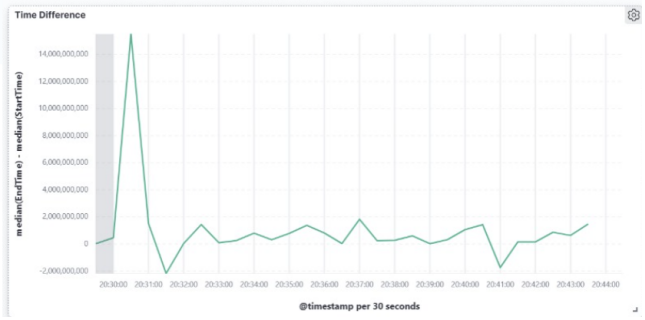

Figure 6: Screenshot of a time graph on kibana

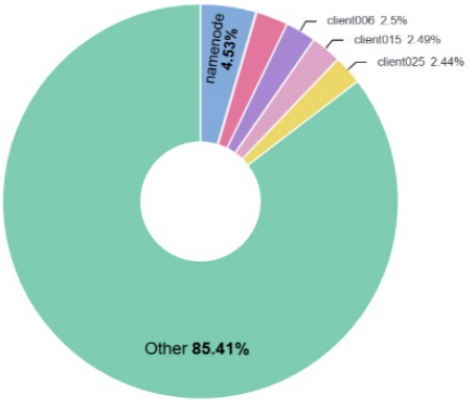

Figure 7: Screenshot of a donut chart of results on kibana

Our project also has a load balancer that redirects to different nodes every time.

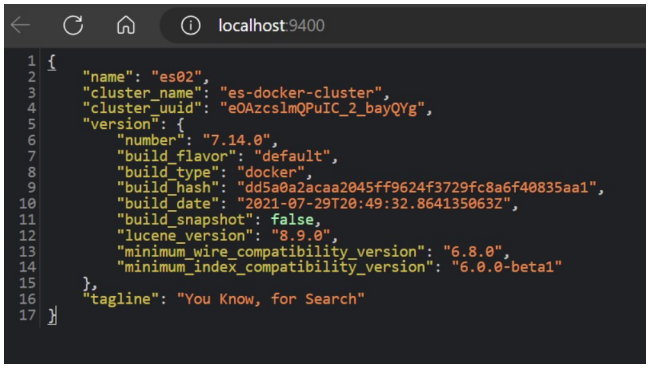

Figure 8: Load balancing - redirects to es02

In this way, the system can scale effectively by using our architecture, which includes a master-slave distributed database design and a 2-stage Logstash pipeline. Redis buffers guarantee asynchronous processing, which lets the system manage big datasets without interfering with later phases.

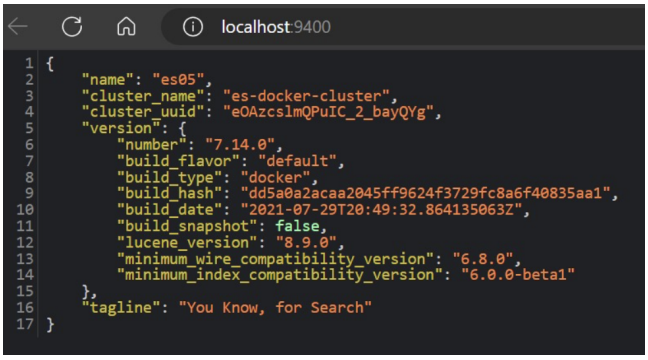

Figure 9: Load balancing - redirected to es05

5 CONCLUSION AND FUTURE WORK

The ELK stack's ability to automate and provide a reliable HDFS log analysis was showcased in this study. We developed a system that can process massive amounts of log data in real time by utilizing the ELK stack's scalability, fault tolerance, and concurrency.

Future work will concentrate on integrating health tools, autoscalers, and additional Logstash pipelines to further improve the system's monitoring capabilities as our study investigates new facets of distributed systems. With these enhancements, the system will continue to be reliable, flexible, and able to manage even heavier workloads while still operating at peak efficiency.

ACKNOWLEDGMENTS

We, the report's authors, would like to express our gratitude to our professor, Dr. Essam Mansour, as well as our teaching assistants, Omijkumar Pravinbhai Mangukiya and Hussein Abdallah, for their assistance in providing the resources, guidelines and inputs required for this study. We also thank the Hadoop community and the ELK stack authors for their documentation and tools.

REFERENCES

[1] Apurba Das Pankaj Sharma Smridhi Verma Avaneesh Kanshi, Aarya Parikh. 2024. HDFS-Log-Analysis-using-ELK. https://github.com/includeavaneesh/HDFS-Log-Analysis-using-ELK.

[2] Loghub Dataset. 2023. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. https://github.com/logpai/loghub.

[3] Mtracer. 2014. TraceBench: An Open Data Set for Trace-oriented Monitoring. https://mtracer.github.io/TraceBench.

[4] Jingwen Zhou, Zhenbang Chen, Ji Wang, Zibin Zheng, and Michael R. Lyu. 2014. TraceBench: An Open Data Set for Trace-oriented Monitoring. In *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*.

[5] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R. Lyu. 2023. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE)*.