Proceedings of the 11th Symposium on Applied Science, Business & Industrial Research – 2019
ISSN 2279-1558, ISBN 978-955-7442-27-3

Implementing a Data Infrastructure for Telecommunication Provider: A Case Study

Vithanage LS, Vidanagama VGTN
Department of Computing and Information Systems, Wayamba University of Sri Lanka
lisharanline.ls@gmail.com

ABSTRACT

Data infrastructure is the key asset of the Telecom service provider to manage its operations. This paper showcases a case study with Design, Development and Implementation of a Data Infrastructure for a mobile operator to handle massive input data load from dispersed data sources in a near real time manner without any data loss in order to facilitate analytics related works with visualizations to understand customer behaviours in operators region.

Keywords: Analytic, Data infrastructure, Massive input data, Near real time

1 INTRODUCTION

In Kenyan region, the second largest Telecommunication (Telco) provider needed an infrastructure to manage huge amounts of user data with zero data loss in real time. Data is the most important asset for Telco providers because every call, messages, voice mails generates data. If a data loss occurs, the operator will be unable to get clear idea about customer behaviors. The case study involves the Design, Development and Implementation of a Data Infrastructure in handling massive input data load from dispersed data sources in a near real time manner; in order to facilitate analytics related work.

There are a number of technologies used to implement this architecture. The implementation also requires all data to be visualized. Some data need to be visualized in real time; so, the architecture needs to handle all data in real time. This paper will cover the Technology, methodology and architecture requirements in the above sections.

1.1 Problem Definition

The second largest telecommunication provider in Kenya region has a huge customer base with them. One of the largest telco software providers in Sri Lanka created two software products for their requirement. They were Voice Mail Service (VMS) and Voice Short Message Service (VSMS). These systems generate huge Call Detail Records (CDR's) because of their customer base of 200000+ average active users in every second. These active users are generating 45,000 CDR's per second so, that is a huge amount of data per second. The data gathering success rate needs to be 100% because if any data lose happens, that will directly affect billing and pricing.

The clients need highly scalable fault tolerance data architecture to manage their operations. Normal relational databases are inefficient for this kind of situation because when the database size increases, the data queries get slower, or might even won't be executed due to process complications. Thus, there has to be a good database and data stream processors to manage the database.

The telco provider itself didn't have a proper architecture for this kind of situation; therefore, a new architecture had to be developed to suit their needs. This provider has a larger customer base compared to others thus, for the Telco software provider; it was a challenge to develop such a thing. For other providers the Common Reporting System (CRS) architecture had only apache spark, apache flume and MySQL. But for this kind of problem that architecture was not enough.

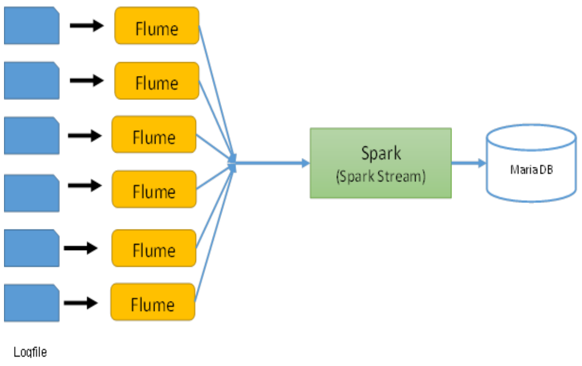

Figure 1: Early CRS (common Reporting System) Architecture

In this architecture, it obtains the data from the log files through Apache flume and the spark stream will save the data into MariaDB table. This architecture is working fine with small scale data loads. But when it comes to the massive data load this will fail. The MariaDB also gets slow when the data input is really quick (Kafka Architecture, 2019).

2 DESIGN METHODOLOGY

In Telco provider side they had two sites in Kenya which needed the new application. It was planned to manage the entire data load with one database cluster. Several technologies were chosen to implement such a data infrastructure.

- Apache Flume
- Apache Kafka (cluster)
- Apache Spark (cluster)
- Cassandra (cluster)
- MariaDB
- Grafana
- Java
- Python

Apache Mesos is a resource management tool and that is optional for our need because Spark cluster performed well in our scenario. Spark cluster worked for both streaming jobs as well as batch processing jobs. This tool required High memory (RAM) in order to perform well (The Distributed SQL Blog, 2019).

Our proposed design was pretty straight forward. We proposed our high-level Architecture as follows:

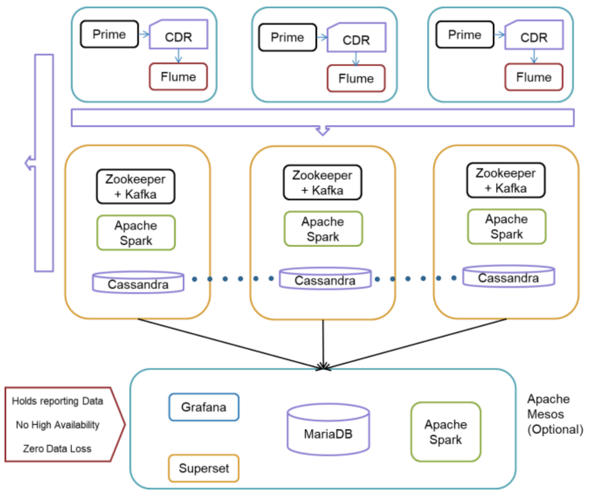

Figure 2: High-level Architecture of System

This Architecture could manage a huge amount of stream data and also manage a batch of data hourly. Java was used to develop stream processor and batch processors (Vaseekaran, 2019). The stream processors were used to store data to Cassandra database in real-time. Batch processors were used to retrieve data from Cassandra table and aggregate those data to get information and store it in MariaDB (MariaDB (MySQL), 2019).

There were two central sites of the Telco provider. For each site there were a separate Kafka cluster and Spark cluster. There was one Flume agent for one application server to pass data to Kafka cluster from application server. There was only one Cassandra cluster for both sites. Both Spark Streams were Inserting data into Cassandra cluster (Apache Kafka, 2019).

MariaDB was installed with master-master configuration to ensure that even when one database was crashed there would be no loss in data. Grafana is the visualization tool that was used to visualize the aggregated data. It was a time series visualization tool to visualize results according to a timeframe (http://msutic.blogspot.com, 2015).

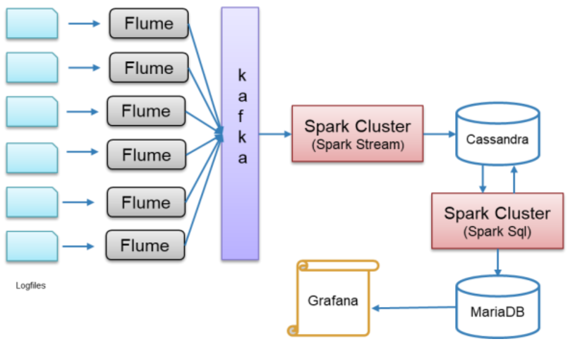

Figure 3: One Site Configuration

3 SUMMARY OF FINDINGS/ RESULTS

Results of this system contain two parts namely Stream processor results and Batch processor results. To measure Stream processor results it was required to confirm that all data were stored in Cassandra table without any loss. Batch processor results were evaluated using the Grafana dashboard

Overall performance was measured based on load testing using generated CDR records. These records were generated continuously so the DataStream's matched for 45,000+ CDR's per second. The memory usage of the architecture was considered as Apache spark was a high memory usage application. But with the given configurations, the system performed exceptionally well.

Grafana Dashboards also provided very high accurate results of the given data.

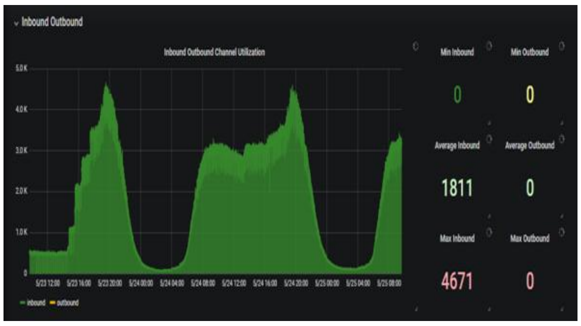

Figure 4: Sample Grafana dashboard -1 (852 x 252 px)

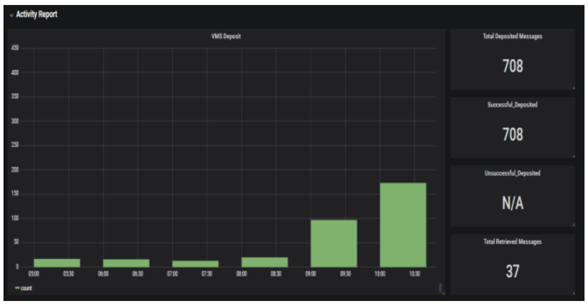

Figure 5: Sample Grafana dashboard -1 (852 x 252 px)

4 CONCLUSION AND RECOMMENDATION

Implementation of Data infrastructure for safaricom was developed and completed in 16-weeks of time, similarly over the same period Stream Processors Batch Processors were also completed and tested. The end result was to implement the infrastructure on Safaricom and test with live data. It worked fine in live environment. The systems received massive data load per second, but the proposed architecture managed that load smoothly.

Although the current system shows good performance, there are areas of risk which may require solutions in future.

1. If flume agent malfunctions in the application server, there will be a break in data stream so the application server data will not come to the Cassandra table.

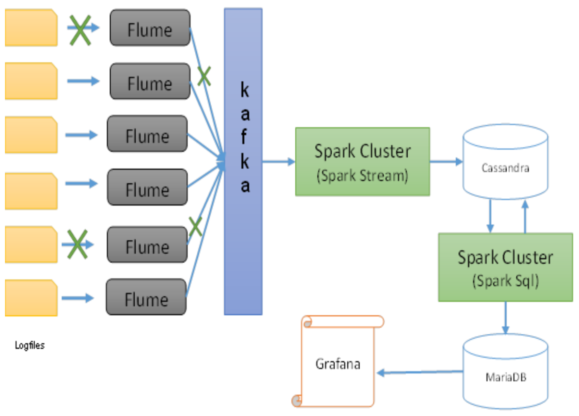

Figure 6: Problem in flume

2. In Spark documentation they have explained that Spark default cluster is not suitable for production environment. They recommend that Mesos is good to manage resources. But in our case Spark inbuilt cluster works fine. Sometimes, this might make some issues in later stage.
3. If the CDR per second count will increase later; for example, 100000+ CDR per second. Spark Streaming is not the answer for that. We have to replace Apache flink for Stream processing jobs.
4. MariaDB is used as a Master - Master replication. So, every aggregate data is replicating again. This can be waste of data in some point. We can use mariaDB cluster to utilize the resources.

REFERENCES

- Apache Kafka - Dzone Refcardz, 2019, [ONLINE] Available at: https://dzone.com/refcardz/apache-kafka?chapter=1.
- Vaseekaran, G. 2019, Big Data Battle: Batch Processing vs Stream Processing [ONLINE] Available at: https://medium.com/@gowthamy/big-data-battle-batch-processing-vs-stream-processing-5d94600d8103.
- Kafka Architecture, (2019), Kafka Architecture, [ONLINE] Available at: http://cloudurable.com/blog/kafka-architecture/index.html.
- MariaDB (MySQL) Master-Master Replication | Marko Sutic's Database Blog, 2019, MariaDB (MySQL) Master-Master Replication | Marko Sutic's Database Blog, [ONLINE] Available at: http://msutic.blogspot.com/2015/02/mariadbmysql-master-master-replication.html.
- The Distributed SQL Blog, 2019, Apache Cassandra: The Truth behind Tunable Consistency, Lightweight Transactions & Secondary Indexes - The Distributed SQL Blog, [ONLINE] Available at: https://blog.yugabyte.com/apache-cassandra-lightweight-transactions-secondary-indexes-tunable-consistency/.