Real-Time Web Server Log Processing

CSC 591 - Data Intensive Computing - Fall 2018

Group 6

Name	Unity ID
Daxkumar Amin	dkamin
Khantil Choksi	khchoksi
Riken Shah	rshah9

Table of Contents

1. Definition

Building a data intensive pipeline to process web-server log data and provide feedback, take necessary infrastructure actions and provide visual analysis. This helps engineers to take necessary actions with their deployed applications on servers and also automate certain infrastructure horizontal scaling real-time.

2. Justification

The purpose of this project is learning. All the members of the team are new to the domain of Big Data and Data Intensive Computing. With this project we are looking forward to learn the concepts around Data Engineering and implement them for a real world use-case, i.e. web server log analysis.

Throughout our academics and during internships we have witnessed the buzz around tools such as Apache Kafka, AWS Kinesis, Apache Flink and likewise. We were always looking for a chance to learn and build applications using these tools and with this project anticipate to do so.

In the initial lectures, class has discussed certains traits of Big Data - volume and velocity, and characteristics of Data Intensive Applications - scalability, availability & reliability. During this project we are eager to build a system with these characteristics and delve deeper into the domain, expanding our knowledge base.

3. Overview

i. *D*escribe what will be built.

A system to provide dynamic heterogeneous analysis of simulated real-time web server log data. The input to the system is a fast stream of web server logs and the output is the processed analysis of various parameters like number of web-page hits, regional involvement, system performance across various microservices, etc. The goal of the system is

to handle large volume of high velocity data, providing resiliency and availability at the same time.

As the feedback of the processed data, we can change error threshold of web-serivce in real-time to get less paged to engineers. Moreover, depending on number of requests per API call in a given timeframe and high/low web response time, we can increase/decrease number of servers for that dedicated services to achieve load-balancing. Also, if we see increasing 4xx & 5xx errors in logs, we can switch the real-time traffic back to stable server / staging server or blue to green cluster switch. This way of providing feedback will help servers automatically handle to large amount of service requests, less error prone and high availability to end user. After each action. we should be able to see

ii. Explain how the above will be built. Introduce the major tasks.

The three main modules of the system are the data generation, data processing, and monitoring dashboard. There are various submodules too, including but not limited to, automatic alerts and notification, dynamic cluster management, etc. The main functionalities of each module is described as follows.

Data generation module will use the base data set of web logs and with some random sampling continuously keep on generating data at high speeds (based all configurable parameters). This will be used for the ingestion of the data by the data processing module. The main component of the data processing module is the multi-level fault-tolerant distributed queue system (kafka/kinesis) which can scale easily. Lastly, there is an analysis module associated which will carry out the processing like Apache Flink / Spark. The dashboard monitoring module is where all the aggregated information as well as notifications can be visualized.

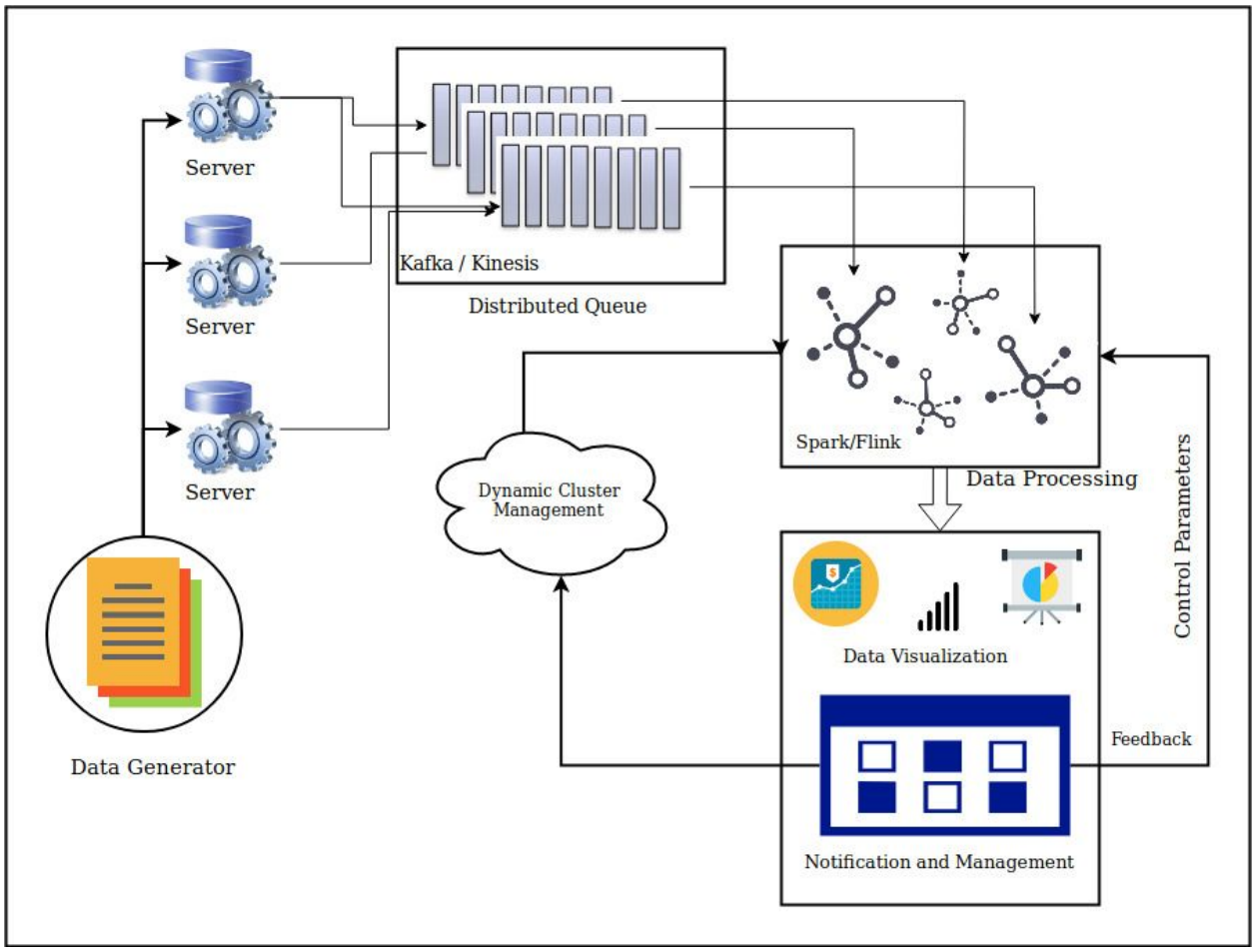
iii. *Verification* Explain how to tell if the project has been accomplished.

Following are the set of deliverables that we plan to accomplish.

- The architecture should be able to process high volume of server logs, with high velocity.
- Process efficiently increasing load of server logs by auto-scaling
- Correct feedback from processed web-server logs

- Deployment of resources in cloud and easily pluggable to real-time servers

4. Architecture



The major components of the architecture are as follows.

I. Data generator module:

We use a base dataset of 26000 rows containing IP addresses, services, response status and date/time stamp. We use dynamic generation methods with some randomization to continuously generate a stream of data that will be used as an input to the

queuing and the processing module. All the communication happens via RESTful APIs and there are tunable parameters to control the flow, delay etc. of the data.

II. Data Queuing and Processing:

A. Data Queueing: For data queuing we use the multilevel, distributed fault tolerant queuing service to handle high volume, high velocity streaming data. Kafka could be used which has message channels known as topics and a publisher subscriber model to access the data in FIFO manner. There could be windowing applied, with caching and fault tolerant techniques to make sure we provide a highly available system. The publishers will be the data generator module servers which will continuously publish the data based on preconfigured parameters and subscribers will be the spark/flink nodes in the data processing module.

B. Data Processing: Once the data is ingested in the queue, the data processing nodes subscribe to the topics or channels to fetch appropriate data continuously. The main function of the processing nodes is to do aggregations on the streaming data and find patterns in the incoming data based on predefined set of rules and parameters. These rules will be used to detect or filter data which will be then provided to the visualization module.

III. Visualization and Management:

A. Visualization: We provide different types of analysis based on number of hits, number of failures or errors, and regional data on one axis as compared to different services on other axis. These type of analysis will be useful for comparing various system parameters.
B. Management: There would be alerts and notifications for major services and tasks which would be useful for the developer to take appropriate actions based on current system state. Also there are various control parameters for various functions that could be used as a feedback loop to modify system state. There is an automated cluster management module as well, which would be configured to manage load and assign green/blue clusters based on the system state.

This is how the different components of the system would work and communicate with one another.

5. Timeline

List major milestones. You should have between 4 and 7 major milestones. In a 15-week semester, that implies milestones will not be more than 4-5 weeks apart and not closer than 2 weeks.

	Milestones	Major Deliverables	Deadline
1	Project Idea	- Project ideas - Large dataset to work on	Sept 8
2	Proposal	- Prepare for project proposal	Sept 13
3	Design	- Design infrastructure - Setup main pipeline components (e.g. kafka, cluster)	Sept 25
4	Implementation	- Data generation for high volume - Develop data ingestion module with high velocity - Build infrastructure and connect all the components - Process, analyze web-server logs and demonstrate the results	Oct 17
5	Deployment	- Deploy the application on AWS / VCL - Handle auto-scaling based on increasing/decreasing flow of web-server logs	Oct 31
6	Testing	- Black-box test the application - Stress the infrastructure	Nov 14
7	Documentation	- Presentation - Documentation - Screencast	Nov 21

6. Background

Real-time log analysis is a process of getting information that is more meaningful and human-readable from web-server log data, in real time. Traditional approach can be of storing log data into some static form such as relational database and then analyzing it in large chunks. But to maintain high availability of web applications it is necessary to stream the logs in real-time data pipeline and analyse it. The data pipeline should be scalable to handle the variability of web traffic.

And for such use case, we need to generate high volume of data as the dataset has 26k rows of log. We need to randomize IP address by country-region, also considering adding the column for “web-response time” for each web-server log. This way, it will help to generate good analysis, considering industry application and also help to determine feedback to the system for auto-scaling of cloud resources. We also need to generate mapping between ip groups and country regions for easier results, analysis and feedback.

7. Resources

i. Data - Web log data set, 26k rows
ii. Data Storage - S3 bucket
iii. Data streaming platform - Apache Kafka / AWS Kinesis
iv. Data processing framework - Apache Spark / Flink
v. Deployment environment on Cloud - AWS EC2 / VCL
vi. Programming resources - Python3, standard py libraries
vii. Pipeline trigger (for feedback) - AWS lambda

These resources are tentative and are subject to change in future.

8. References

- Jay Kreps, Neha Narkhede, Jun Rao. *Kafka: a Distributed Messaging System for Log Processing.* NetDB'11, June 12, 2011, Athens, Greece.
- Matei Zaharia, et al. *Spark: Cluster Computing with Working Sets.* HotCloud 10.10-10 (2010)
- https://www.kaggle.com/shawon10/web-log-dataset
- https://kafka.apache.org/documentation/
- https://aws.amazon.com/documentation/kinesis/
- https://spark.apache.org/documentation.html
- https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/python.html