Development of a Data Pipeline for Real-Time Network Analytics

Internship Experience

CS4991 Capstone Report, 2021

Derek Johnson
Department of Computer Science
University of Virginia
Charlottesville Virginia USA
Dej3tc@virginia.edu

ABSTRACT

My summer internship centered around creating a system for analyzing network logs with machine learning to determine outages and automate fixes. I developed a pipeline that would consume data from a variety of sources, analyze said data using models created by data scientists, and relay the results to a final data sink. I used Kafka stream processing to make the pipeline distributed and fault-tolerant. Additionally, I developed a batching service capable of formatting incoming data to the specifications of the models. Although I was not able to complete the project, my contributions provided a robust and scalable solution for deploying machine learning models. As more models are added to the system, this data pipeline will need to become easily configurable and deployable. This will allow for rapid integration of new features.

1 INTRODUCTION/BACKGROUND

Traditionally, managing networks has been the responsibility of system administrators and IT professionals. Once they reach a certain size, they require round-the-clock supervision to prevent failure. Most network outages are caused by human error or machine failure. Network usage increases proportionally with the size and complexity of an organization. Network outages can lead to financial losses as well as loss of trust from users. New tools for preventing these outages have the potential to help companies avoid these negative repercussions.

Machine learning (ML) is a general term to describe techniques that use computers to identify patterns in training data. It is most effective with regular, well-structured data **[1]**. Various network devices, such as routers and switches, log data about the connections they make in order to enable humans to detect network problems. On the surface, network logs seem like a perfect application for ML. Some challenges, however, have prevented ML from fully mastering this domain. Each network is unique and constantly evolving. It is also difficult to establish ground truth for network data. If a machine learning system is to provide useful insight to a network administrator, it must be trained on the network where it will be deployed, and it will need a large amount of training data.

Processing a large amount of data in real time is an issue that has received a lot of attention in recent years. The stream processing system has risen to solve this issue. At its core, stream processing is a programming paradigm that allows for simple parallel processing. Computation on data occurs in a pipeline. This is essentially a series of connected data processing steps. There are several popular frameworks that abstract some of the complexity of building a stream processing system. These include Spark, Fink, and Kafka**[2]**.

2 RELATED WORKS

Currently, much of the tooling available for network administrators is based around providing useful metrics about the performance of the network. Some of the products that are available currently include SolarWinds and Splunk. Both tools can be integrated into existing networks and can provide insights into the performance and health of a network. Additionally, they both market machine learning capabilities for diagnosing problems in a system. It is not clear what types of models are deployed for this detection. There are many other smaller companies entering the network tooling space. This seems to be in response to growing excitement about the field of AI/ML.

Despite the growing number of companies offering AI analysis on network data, there are some who are skeptical of how novel these products truly are. Jagjeet Gill, principal in Deloitte's strategy practices said "We're probably overusing the term AI, because some of these things, like predictive maintenance, have been in the field for a while now,"**[3]**. An issue facing ML systems is vendor lock in. Cisco and other network device manufacturers offer ML analysis tools. This will work well if a network is made up entirely of Cisco products, however, it can become a problem in multi-vendor environments**[3]**.

One area of interest in the ML network space is traffic prediction. It is essentially forecasting future traffic in and out of a network. This problem, if solved, would allow traffic to be routed more efficiently through a network, resulting in quicker response times,

due to routing optimization. Supervised neural networks have shown high prediction accuracy on low complexity networks. The difficulty in applying this to more complicated, high-speed networks is due to an inability to accurately measure traffic.

3 PROJECT DESIGN

My summer internship revolved around creating a data pipeline to run machine learning models on network log messages. I was not responsible for creating the models, only for supplying them with data and collecting the results.

3.1 Requirements

- The system must be able to pull data from multiple types of databases.
- The system should be able to run data on multiple ml models concurrently
- The system should be able to run on both streaming models and batch models.
- Results from the models should be collected and placed in a data sink for further analysis.

3.2 System Architecture

Data Source: The pipeline consumes data from an Elasticsearch database. Elasticsearch is a NoSQL database that provides a highly efficient search engine.
Stream Processing: Kafka was chosen as the stream processing platform. Kafka allows data to be written to and read from streams of events called topics. Additionally, it has an API to process events in a stream as they occur **[6]**.
Model Deployment: The machine learning models would be run as docker containers on EC2 instances.
Development: The data pipeline was written in Python. The confluent_kafka API was used for producing to and consuming from Kafka topics **[7]**. The faust library was used to develop stream processing services.

3.3 System Design

The design of the system can be thought of as a data wrapper around the ML model. This data wrapper consumes network logs from elasticsearch using a time query. When enough new logs are present, they are ingested into a Kafka topic using the Consumer API **[8]**. While the data wrapper was being developed, all the models required batched data. Thus, items that are added to the ingest topic are grouped for batch processing. When a new batch was prepared, it was sent to a batch ingest topic. When batches were written to this topic, it would trigger the model to run on that batch of data.

Communication between the data wrapper and the models was done using a REST API. Each model had an API endpoint and would be triggered to start running by a call to that endpoint. The data to run on the model was passed in the body of a JSON request. When the model was finished running, it would return a status code and the results of the run in the JSON response. Creating these API calls and waiting for their responses was handled by a stream processor. Finally, this stream processor would send the response data to a "sink" Kafka topic. This "sink" topic would be subscribed to by any user of the system who wanted to process these results.

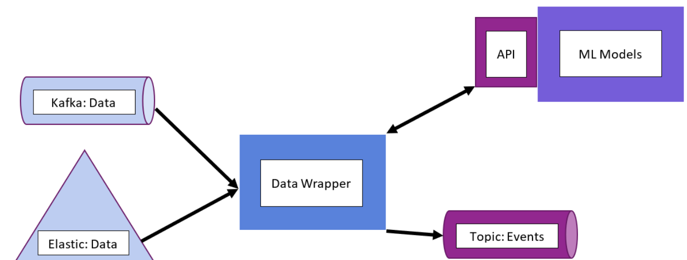

Figure 1: System Design for the data pipeline.

Figure 1 displays this system design. On the left are the two sources of log data: Kafka and Elasticsearch. In the center is a box representing the data wrapper. Arrows represent communication between two pieces of the architecture. Tubes represent Kafka topics.

3.4 Challenges

The biggest design challenge was trying to marry the streaming system that I was supposed to deliver with the existing changepoint detection models. The models deployed on the system would scan a time window of logs and try to identify regions of rapid change. This change would often signal a failure in the system or anomalous behavior by some user. For the models to work they needed to receive batches of logs. This caused issues because the streaming model is designed to perform analytics on individual messages.

To get around this challenge, I developed and deployed a streaming batch creation process. This process would listen for logs and batch them together to be processed. Faust was used to listen to topics receiving logs, order them by timestamp, create a batch, and produce them to a Kafka topic.

4 RESULTS/FUTURE WORK

At the conclusion of my internship, the pipeline was set up and data could run through every step of the process. The Kafka topics were deployed to servers and could be remotely configured. Although I will not be involved with the project going forward, other members of the team will continue the work outlined in this paper. There are several important next steps in the deployment process for this project. This first is to deploy the stream processing code. When I left the team, this section of the pipeline was run on local machines. In my last week I began working on deploying my code on a container. This container would run on the same network as the model API and the Kafka topics.

Another important step in this project will be to create detection models that can identify changepoints in real time. This will require a more adaptive model that can constantly be retraining. Once this is deployed, it will remove the need for a buffer. This

was identified as an area of focus for the future as the goal of the project is to deliver real time analytics. By deploying a model that relies on batches of data, the project will not be able to deliver on this goal.

5 CONCLUSIONS

During my internship, I played a role in developing a data pipeline that could analyze network logs and deliver insights into changes in network activity and performance. The data pipeline that I created will be able to be used as a reference for others working on the project who are trying to deliver data to and receive data from machine learning models. The implementation that I developed is both efficient and fault tolerant. Additionally, it is designed to be modular. Adding additional data sources can easily be accomplished by reading data from the data source and writing it to the input Kafka topic. The system can also accommodate data coming in from many different sources at the same time.

Networks will only become more complex in the coming years. It will be important to arm network administrators with the tools to identify issues in real time. Recent advancements in machine learning and event streaming architectures allow for tools that can adapt to individual networks and identify issues as they happen.

6 UVA COMPUTER SCIENCE EVALUATION

All the classes that I have taken at UVA prepared me for my internship. However, there are three that I have identified as being especially beneficial.

The first is Software Development Methods with Professor Mark Sherriff. This class introduced me to the git workflow. As a new software developer, I was familiar with GitHub as a place to store code, but I knew little about how to use it to collaborate with other developers. This class forced me to work with older students who had more experience coding collaboratively. Through this experience I learned about the power of feature branches and agile development. This made the transition to working in a professional development role smoother. I learned that building software involves being able to communicate your ideas to others. Getting in the habit of working on a codebase with others shows the importance of writing maintainable code.

The second class that helped me in my summer internship is Operating Systems with Professor Charles Reiss. During this class I learned how to design for parallel processing and safety. These concepts were reinforced in the assignments given, specifically the twophase and pool homework. In working on the twophase assignment I was asked to design for fault tolerance. I had never written software that had to anticipate hardware failure. This experience was incredibly helpful when designing my data pipeline. The pool assignment forced me to write asynchronous code. The Kafka stream processing API is essentially a series of asynchronous function calls. The transition to writing code for Kafka would have been more difficult without this class introducing me to the fundamentals of asynchrony.

The last class I would like to identify as being especially helpful was Compilers with Professor Matthew Dwyer. Although the information covered in the class had nothing to do with my project this summer, I improved greatly as a developer. When I started the class, I had no experience working on a large codebase. The only code I had interacted with was code that I had written myself. The central assignment for this class was to extend the feature set of an existing compiler. This necessitated exploring documentation and spending time reading other people's code. This proved to be a valuable learning experience. The first two weeks of my internship were primarily getting familiar with the codebase I would be working with. Becoming adjusted to working in a new codebase would have been more daunting if I had not already had the experience gained in Compilers.

This leads me to some areas I think the computer science curriculum could be improved upon. During my internship I improved my coding style. While this was partially due to writing a large volume of code, what really forced me to be better were the code reviews I took part in and the linting tool built into our deployment pipeline.

Working with a more experienced developer opened my eyes to mistakes I did not know I was making. Integrating the code review concept into lower-level computer science courses would have helped me to avoid some bad habits that I picked up as a new computer science student. It has been demonstrated that groups of three to four students working with a trained moderator to check each other's code see an improvement in code quality **[4]**. This concept could easily be incorporated into CS 1110. The major limiting factor would be the number of TAs required to run these small groups.

In my internship, before committing any code to the master branch, my code had to pass a set of linting rules. A linter is a tool that checks code for programmatic and style issues **[5]**. Although they do not find bugs in code, they help to ensure that programmers avoid common antipatterns which can lead to more maintainable code. It may be hard for students in the introductory CS classes to understand the importance of linting. However, by the 2000 level courses, students are proficient enough to think about how to write their code, not just what their code will accomplish. The automated grading servers could incorporate a linting check when they are scanning submissions. If code style was worth five to ten percent of a homework grade, it would encourage students to stop using bad practices in their assignments. By introducing linting tools early in the computer science curriculum, students would be forced to write code with better style.

REFERENCES

[1] Raouf Boutaba, Mohammad A. Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M. Caicedo. 2018. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *J. Internet Serv. Appl.* 9, 1 (June 2018), 16. DOI:https://doi.org/10.1186/s13174-018-0087-2

[2] Paris Carbone, Marios Fragkoulis, Vasiliki Kalavri, and Asterios Katsifodimos. 2020. Beyond Analytics: The Evolution of Stream Processing Systems. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (SIGMOD '20), Association for Computing

Machinery, New York, NY, USA, 2651–2658. DOI:https://doi.org/10.1145/3318464.3383131

[3] Jon Gold. 2020. Machine learning in network management has promise, challenges. *Network World*. Retrieved October 26, 2021 from https://www.networkworld.com/article/3587131/machine-learning-in-network-management-has-promise-challenges.html

[4] Christopher Hundhausen, Anukrati Agrawal, Dana Fairbrother, and Michael Trevisan. 2009. Integrating pedagogical code reviews into a CS 1 course: an empirical study. *ACM SIGCSE Bull.* 41, 1 (March 2009), 291–295. DOI:https://doi.org/10.1145/1539024.1508972

[5] Florian Obermüller, Lena Bloch, Luisa Greifenstein, Ute Heuer, and Gordon Fraser. 2021. Code Perfumes: Reporting Good Code to Encourage Learners. In *The 16th Workshop in Primary and Secondary Computing Education* (WiPSCE '21), Association for Computing Machinery, New York, NY, USA, 1–10. DOI:https://doi.org/10.1145/3481312.3481346

[6] Apache Kafka. *Apache Kafka*. Retrieved October 23, 2021 from https://kafka.apache.org/intro

[7] confluent_kafka API — confluent-kafka 1.7.0 documentation. Retrieved October 24, 2021 from https://docs.confluent.io/platform/current/clients/confluent-kafka-python/html/index.html

[8] KafkaConsumer (kafka 2.6.0 API). Retrieved October 23, 2021 from https://kafka.apache.org/26/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html