Geospatial Tracking:
Development of Alert Software for Real-time Aerial and Maritime Tracking System

CS4991 Capstone Report, 2021

Kayla Lewis
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ksl3fs@virginia.edu

Abstract
A proper alert system is an important aspect of any geospatial tracking system because most of the data that is tracked and displayed is streamed in real-time. Even though some data is stored for a short time and can be indexed through, real-time alerts are the best way to ensure that events of interest are saved.

I addressed the need for alert notifications in a geospatial tracking system by creating a backend application that continuously compared the filter criteria to live data observations from the tracking system. When data satisfied a given criteria, the application created an alert object with information about the filter criteria and the observation that triggered the alert. The alert object was then written to an output data store. This double input, single output process for the backend alert software allowed for reliable and configurable user notifications. This application can be improved in the future by sending alerts through email and increasing the scalability of some of the components.

1 Introduction
Geospatial data and tracking systems are crucial for commercial transport routes, disaster response, and military planning. The geospatial system I worked with was built to track and display global aerial and maritime routes. Because of the scope, large amounts of data have to be fed into and analyzed by the tracking system. This data becomes visible on the graphic interface of the tracking system, and users are able to search for and observe the current locations of airplanes and ships as well as the route they have taken since the beginning of their course. This all happens in real-time, meaning that if an event of interest occurs, such as a ship passing through a certain set of coordinates, it is easy to miss if not being specifically looked for at the right time.

When I began working with this geospatial tracking system, there was a method in place for users to set up criteria for which they should be alerted. These criteria for alerts were called saved queries, and these saved queries were stored in an external database that was unutilized, leaving a significant portion of the geospatial tracking system undeveloped. Users needed a way to be quickly alerted once an event defined by their saved query occurred. The solution to this problem was to create a separate backend application that reads from the database of saved queries and that could run in parallel with the geospatial tracking system to create alerts that could be sent to users.

2 Related Work
This project relied on concepts of Big Data stream processing. This is a model of data analytics that is growing in popularity because it

allows for a high throughput of large volumes of processed data. This model is enabled by streaming frameworks such as Apache Kafka, and studies into the architecture and advantages of Apache Kafka in Big Data streaming processing were used for the development of this project [1].

3 Project Design

The three main elements of this project design consist of the tools used to create the alert application, the background knowledge of the existing geospatial tracking system, and the system design of the alert application and its interconnected components.

3.1 Tools Used

I was introduced to many new tools and software libraries when developing the alert application for this geospatial tracking system. For compatibility purposes and company standards the main tools and libraries I used for the alert application were the same as those used by previous engineers to build the tracking system, namely Scala, Apache Maven, Apache Kafka, and GeoMesa.

Scala is a statically typed high-level programming language that supports both functional and object-oriented programming and integrates seamlessly with Java and JavaScript [2]. The versatility of Scala as well as the compatibility with Java libraries are the main reasons the language was used in my project as well as the existing tracking system. Apache Maven is tool that is used to primarily build and manage Java projects, but can also be used with other interpreted programming languages such as Scala, C#, and Ruby [3]. Since I had previous experience working with Java and using project management tools, it was not difficult to utilize Scala and Maven to implement the alert application.

Apache Kafka is an event streaming platform that allows systems to push and subscribe to streams of events that can come from a wide range of sources including databases, sensors, and cloud services [4]. An important aspect of Kafka in the context of the tracking system in this project is the ability to use Kafka as a way to reliably collect and analyze large amounts of data that is streamed in real-time. For this project, Kafka is used in conjunction with a suite of tools called GeoMesa. GeoMesa allows for spatial-temporal indexing on top of Kafka and other data management systems such as Accumulao, HBase, and Spark [5].

3.2 Background

The existing geospatial tracking system that I worked with was a project that had been in development for around four years and was nearing the end of its development cycle. The tracking system had many different implementations, but the one relevant to my project used Kafka Data Stores from the GeoMesa tool suite to manage the flow and analysis of live location data. Each Data Store was configured to read in data from a variety of sources. The unclassified implementation of the tracking system I worked with used about 10 sources that contained public aerial and maritime traffic. Each entry of data from the Data Store was called an observation, and each observation consisted of the coordinates, time the aircraft or ship was observed at those coordinates, and some unique identifier.

In the graphic interface of the tracking system, the user is able to select which data sources they would like to observe, and after a short delay they are able to view the location of each observation in the form of a small dot on a global map. The user can then zoom in and click on individual observations and view any data related to that observation. Another way for the user to interact with the tracking system is to make a real-time query or save one to track future observations. A query is a request for observations that meet a certain set defined parameters or criteria, and within this tracking system, queries are often based on retrieving

observations of objects within a certain polygonal region. Saved queries are stored in an external database, and the non-utilization of this database was the motivation for the alert notification project.

3.3 System Design

The alert application is designed to process large amounts of data in real time. To accomplish this, the application uses three main components which operate simultaneously to process the live observation data and send alerts. The components are the Data Store listener, the saved query cache and the alert constructor. These components are connected to a main controller. This is seen in Figure 1 which shows the general flow of data between components with the components inside the main square being the components that are a part of the alert application.

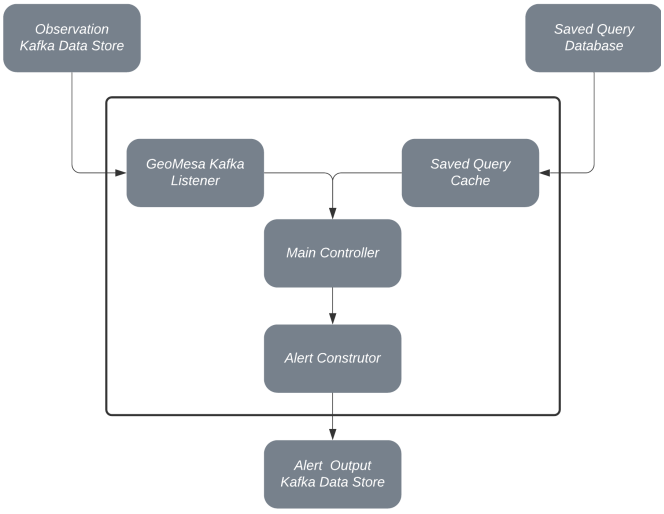

Figure 1: Alert Application Components

The Kafka listener is built over a GeoMesa tool that subscribes to a Kafka topic and outputs changes depending on how the listener is configured. For the case of this system those changes would be the addition of a new observation to the Data Store. It is important to note that the listener reports the addition of an observation in real-time and this report is not repeatable so the observation must be acted on immediately. Because of this, each new observation is passed to the main controller so the computation can be performed, making the Data Store listener the first source of input.

The second source of input is the saved query cache. This cache is an in-memory replication of the saved query database that is used in the tracking system. The cache is configured to refresh at a given interval to stay updated with any new saved queries that are added to the database. An in-memory caching system is used for reading saved queries in order to cut down on external database query times. Since all the saved queries are stored in memory they can be processed quickly in order to keep up with the Kafka Data Store stream.

The alert constructor functions as the output component of the alert application. This component takes data from an observation and the saved query that matched that observation and creates an alert object. The structure of the alert object is configurable so that this application as a whole is compatible with any sort of output destination. While I worked on this application, the desired output location was a Kafka server.

All of these components are linked together by the main controller. The main controller is the entry point of the backend application. Every observation that is reported by the Data Store listener is passed to the main controller which then compares that observation to every saved query that is within the cache at that time. When the filter in the saved query matches an observation, the main controller sends the key information from both the observation and saved query to the alert constructor.

4 Results and Conclusions

Each component of the backend alert application was built using Scala, Maven, and GeoMesa tools. The application was intended to run separately and in parallel with the geospatial tracking system, and its connection to the tracking system lies in its use of the same input and output sources that are seen outside of the

central square in Figure 1. Each component communicates with those outside output sources based on a set of configurations, such as domain names, usernames, and passwords that are taken from a configuration file that is loaded in from the main controller. This means that the application could be configured to work with any set of Data Stores and databases. The application was tested using standard unit tests for basic component functionalities, and to test the interaction between components, I used an in-memory instance of Kafka as well as an in-memory database engine.

When the application starts, a startup log prints in the command line that shows any configuration information as well as the status of the three external connections. If any of the connections fail, the application will log that there was an error with establishing a connection and the application will terminate. Otherwise, the main controller will start accepting and processing data from the Kafka listener and the saved query cache, and sending that data along to the alert constructor if an alert should be triggered.

5 UVA Program Evaluation

The development of this project allowed me to gain soft skills relevant to software engineering as well as technical skills in software development and testing. Of all the computer science courses I've taken, the course that prepared me the most for this project was CS 3240 Advanced Software Development. This course had a focus on the agile software development methodologies as well as Git collaboration. Learning agile software development methodologies prepared me for the style of development that took place with this project. I was assigned the ticket for creating the application, and this ticket was divided into sub-tickets with the goal of completing a sub-ticket at the end of each sprint cycle. Similarly to CS 3240, these sprint cycles were a week long. I worked on this project primarily independently with help from a senior advisor, so I did not experience the same amount of team collaboration I did in CS 3240. Despite this, there were still many benefits to developing within a Git repository such as version control. CS 3240 also prepared me for creating high quality unit tests for my application. It would have been very helpful if CS 3240 incorporated more traditional stands up into the curriculum. From my experience, being able to quickly and effectively communicate the status of a sprint task seems very important in industry, so it would have been nice to have some practice of that in class.

Another course that prepared me for this project was CS 2150 Program and Data Representation. The alert application needed to quickly run comparisons on large amounts of data constantly, so it was beneficial for me to have a deep understanding of data structures and the time complexities around them.

6. Future Work

There are two main improvements that can be implemented for the alert application and its components. The first would be to extend the alert constructor to create and send emails to the address on the saved query involved in the triggered alert. This email generator could also be implemented as a separate application that sends emails based on the information in the data store that the alert constructor writes to. Another improvement to the alert application would be to improve the scalability of the saved query cache. Presently, the query cache component queries the external saved queries database once and copies the entirety of the database to memory because it would be very expensive to query the database for every processed observation. This was acceptable at the time of development because saved query objects were limited in number and in size, and the memory of the system the application was expected to be executed on was relatively large. In production, this could change so there should

be a way to repeatedly query and cache a subset of the external saved query database.

References

[1]Bhole Rahul Hiraman, Chapte Viresh M., and Karve Abhijeet C. 2018. A Study of Apache Kafka in Big Data Stream Processing. In *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, 1–3. DOI:https://doi.org/10.1109/ICICET.2018.8533771

[2]“Scala Language Specification | Scala 2.13.” https://scala-lang.org/files/archive/spec/2.13/ (accessed Oct. 06, 2021).

[3]“Maven – About Maven.” https://maven.apache.org/about.html (accessed Oct. 06, 2021).

[4]“Apache Kafka,” *Apache Kafka*. https://kafka.apache.org/intro (accessed Oct. 06, 2021).

[5]“What is GeoMesa?,” *GeoMesa*. https://www.geomesa.org/documentation/stable/user/introduction.html#what-is-geomesa