Mugla Journal of Science and Technology

DESIGNING A PIPELINE WITH BIG DATA TECHNOLOGIES FOR BORDER SECURITY

Fatih AYDEMİR[1], Aydın ÇETİN[2]*

[1] STM Savunma Teknolojileri Muhendislik ve Tic. A.S., Ankara, Turkey
faydemir@stm.com.tr
[2]Department of Computer Engineering, Faculty of Technology, Gazi University, 06500 Ankara, Turkey
acetin@gazi.edu.tr

Received: 29.04.2016, Accepted: 02.05.2016
*Corresponding author

Abstract

Developing technology, communication and transportation facilities have led borders to the gradual disappearance. The incidents taking place in different countries go beyond the limits and being felt by large audiences. Solutions to rapidly increasing terrorist attacks and internal turmoil caused security problems in recent years which have become very complicated. Protecting and controlling the borders, finding rapid and comprehensive solutions to illegal border passing problems have become a major problem in many countries. When considering the scope of the border security, it would be understood to examine the data which comes at different times from different sources and types in real time. This can be achieved by a collaboration of different principles. In addition, when large-scale unstructured data analysis also considered for a comprehensive solution, it would be a rational method to develop a pipeline using the Big Data technologies. In this study, we aim to achieve cost-effective, robust, scalable and flexible system to solve the border security problems. The use of Lambda Architecture which provides real time data processing and batch processing capabilities was investigated in border security applications. System development was explained and all information on its principles was provided.
Keywords: Apache Spark, Lambda Architecture, Apache Kafka, Cassandra, Real-time Data Processing, Batch Data Processing

SINIR GÜVENLİĞİ İÇİN BÜYÜK VERİ TEKNOLOJİLERİ İLE BORU HATTI TASARIMI

Özet

Gelişen teknoloji, iletişim ve ulaşımdaki kolaylıklar sınırların yavaş yavaş ortadan kalkmasına yol açmıştır. Bu sebeple farklı coğrafyalarda gerçekleşen olaylar, sınırların ötesine geçerek etkisini büyük kitlelere hissettirmektedir. Sınırları denetlemek, korunmak, kaçak geçişlere karşı hızlı ve kapsamlı çözümler bulmak birçok ülkenin ana problemi haline gelmiştir. Sınır güvenliğinin kapsamı düşünüldüğünde; farklı zamanlarda, farklı kaynaklardan ve farklı tiplerde gelen verinin, gerçek zamanlı olarak incelenmesi gerektiği anlaşılacaktır. Bu da farklı prensiplerin bir arada çalışması ile mümkündür. Bunların yanı sıra yapılandırılmamış büyük ölçekli veri analizi de göz önüne alındığında, kapsamlı bir çözüm için Büyük Veri (Big Data) teknolojilerinden yararlanarak boru hattı (pipeline) geliştirmek akılcı bir yöntem olacaktır. Bu çalışmada, sınır güvenliği için gerçek zamanlı akan veri ve yığın veri üzerinde analiz yapmayı sağlayan Lambda Mimarisi (Lambda Architecture) yetenekleri araştırılmaktadır. Sınır güvenliği problemi çözümüne yönelik geliştirilen uygulama açıklanarak sistemin çalışması hakkında bilgi verilmektedir.
Anahtar Kelimeler: Apache Spark, Lambda Architecture, Apache Kafka, Cassandra, Gerçek zamanlı veri işleme, Yığın Veri işleme

1 Introduction

The emergence of new threats and methods have led to the displacement of existing threats as a result of having the rapid developments in technology, communication, transportation and armament opportunities. Technical and technological changes in the military field, divergences in the management layer, problems such as negative effects of economic factors on individuals, societies and states have increased the crime initiatives from threats which put individuals in a difficult situation about ensuring the safety of their lives and properties. Therefore, units which are responsible of ensuring the security need new approaches and cautions to provide an effective service [[1]].

When security considered as a multi-layered architecture, the outermost layer would be the protecting the area where people live against external threats. Therefore, security could be called that starts from the 'borders'. Ensuring the security of a region is possible with the coordinated operation of the all related units which work in different areas and systems.

In studies about the border security, it's emphasized that the security can be ensured by the integration and collaboration of the underground and surface sensors, low and high resolution cameras, unmanned aerial vehicle (UAV), the satellite and the radar [[2]]. As a result of a collaboration of different disciplines, the data can be in different formats, in different sizes and obtained at different time intervals. It is difficult for widely used database management systems and architectures to store high volumes of data and analyze unstructured data with high speeds and variability. Big Data technologies can provide significant advantages in solving these addressed complex

problems. Big Data concept can be explained as storing and querying data in a meaningful, economic and scalable way [[3]].

A border controlling system should have the following features to solve the major problems:

- Doing the 'clean up' and low-latency configuration/interpretation on the flowing data,
- Performing the 'recording/storing' and 'query processing' in a high speed,
- To be able to give instant information to users and providing the re-examination of the records.

In this study, we aim to achieve cost-effective, robust, scalable and flexible system to solve the border security problems. The rest of the paper, structured as follows. In Section 2, Architectural Design and Methodology were given in detail. In Section 3, Experiments and Findings were presented and the Conclusions of the study were submitted in Section 4.

2 Architectural Design and Methodology

The preparation rules given below should be followed: It is recommended that you either use the template or stick to a sample file in order to meet the specifications for the format of MJST papers.

There are many available technologies, architectures and frameworks for developing a pipeline. A pipeline should possess the following properties: working with different sources, having low latency transport mechanism, avoiding loss of data during data transmission and recording, having a high data processing speed and presenting the results in a way that is readable and understandable manner.

The software components in the system should be separated in accordance with their purpose of use and be able to work without being aware of each other's work. In the framework of this logic, we have designed the main backbone of system with Event -Based Architecture (EDA) [[4]]. Thus, each component has been considered as a loosely coupled and singly-responsible structure. Data sender and data receiver do not have a direct relationship between components transmitting and receiving data in EDA. As shown in the Fig. 1, workflow is performed with events transmitted over Kafka.

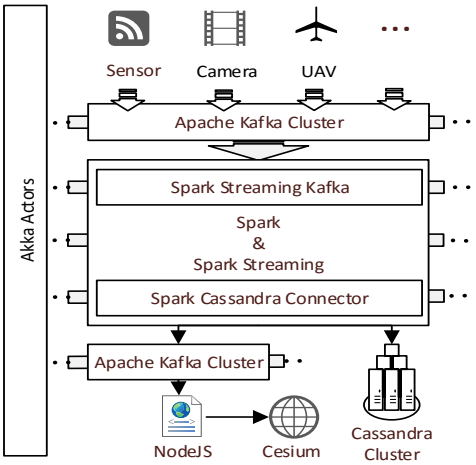

Figure 1. System architecture.

In the study, we aimed to build a distributed, robust, coherent, sensible, large-scale and low-latency data pipeline. To achieve this aim, a design based on the Lambda Architecture (LA) is proposed with following components;

- Apache Kafka for the data transmission,
- Apache Spark to process in real-time data and batch data,
- Apache Cassandra for storing the data,
- Nodejs to provide processed data to the end user,
- Cesiumjs for 3-dimensional representation of the object on the map.

Apache Kafka is an open-source framework which implements publish-subscribe approach over TCP/IP and records them in the same method with log registers [[5]]. Apache Spark is an open source cluster with the capability of computing framework build and ease of use, complex analytics and speed [[6]]. It provides an API based on micro batch style called the resilient distributed dataset (RDD). Cassandra is a NoSQL distributed database management system with column based structure, low latency, open source code and distributed database designed to handle large amounts of data [[7]]. Nodejs is a cross-platform runtime environment letting us to develop server-side applications using JavaScript programming language and having a mechanism with open source code, event based and non-blocking input/output (I/O) [[8]]. Cesiumjs is an open-source library which is used for 3D globes and 2D maps in web browser [[9]].

2.1 Real-time Data Flow

It is difficult to predict the magnitude and frequency of data flow in real-time processing. Therefore, during the transmission of data, bottleneck may occur between the sender and the receiver. Consequently, the J. Boner et.al, have claimed an asynchronous stream processing with non-blocking back pressure in the Reactive Manifesto [[10]] issued in 2014. After the Manifesto was published, many systems and tools which support Reactive Streams have been developed. One of the reactive streams tool has been developed with Akka Library is called as Akka Streams. The difference between sending and receiving data speed should be at minimum level for the system integrity and consistency. Thus, we choose the Akka Streams on the TCP Client/Server model which we use the retrieving data from the source. TCP server publishes the data via Kafka when received data. And also KafkaAkka Consumer and KafkaAkka Producer classes were implemented using Akka library providing explicit locking and thread management. These KafkaAkka classes were used for all messaging operations in the system.

2.2 Data Modelling and Decomposition

When it comes to border security, it won't be enough to make just real-time processing. Working on the batch data helps us to retrieve useful information by removing irrelevant data, grouping data and making sensible information from them. In data modelling and cleaning process to provide this capability to our system, we worked based on Lambda Architecture which is created by Nathan Marz and James Warren [11]. We preferred Apache Spark to make distributed computing on the large-scale data. These tools with real-time data and batch data processing capabilities keep requests in the queue and make them pending until the action function is called providing quick realization of the next process by writing the process results to the memory. The Spark Streaming is the component that processes the real-time data received from different sources. Spark Streaming creates Discretized Stream (Dstream) for each data flow. DStream may be expressed as an array made on micro batch series (RDD).

Spark Streaming can be easily integrated with many messaging system and can receive data via multiple topic simultaneously. There are two methods which can be used for data flow between Kafka and Spark. These are Direct Approach - no receivers and Receiver-based Approach [12]. In the study, Direct Approach has been chosen because of its ease of installation, data lossless operation and its parallel processing characteristics without custom settings.

2.2.1 Lambda Architecture

Lambda architecture (LA) is data processing architecture designed to handle large amounts of data using the advantages of stream processing and batch processing [13]. LA consists of three main layers. These are batch layer, speed layer and serving layer. The structure of the LA is given in Fig.2.

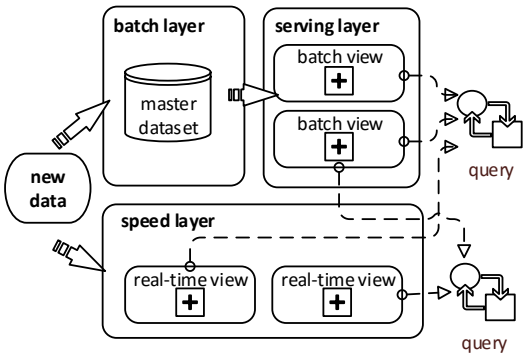

Figure 2. The LA structure.

As it can be seen from the Fig.2, all data entered into the system is sent to both speed layer and batch layers. There are two main tasks of the batch layer; namely, to check the master data set and to create batch pieces. Batch pieces are made available for querying by being indexed in the serving layer. Huge delays arising from the update process at the presentation layer are handled by speed layer. It acts on the last incoming data. Inquiries can be carried out by using the serving layer and speed layer.

2.3 Storage and Presentation

By using Spark Cassandra Connector, not only Spark RDDs can be managed like Cassandra tables, but also Cassandra tables can be managed like Spark RDDs. Therefore, RDDs which are formed from the Map/Reduce processing results can be written to Cassandra database. The Spark Application implemented to process data stores RDDs by using these abilities.

Processed data representation for the information to end-users is also included in this study. Spark saves the processed data in Cassandra and publish simultaneously with Kafka. The module developed using Nodejs serves as a web server as well as Kafka consumer. Kafka doesn't delete the records after sending them to receiver. However; it keeps them during the period of time specified in the configuration file. Thus, a mechanism such as long polling is established. When clients start listening to the related topic, it takes all historical records. That means, when Nodejs consumer is connected to Kafka, it takes all retroactive data. Socket IO has been chosen as a communication model between clients and web server. Web client converts the data received via Socket.IO into 3D Cesium object and allows the examination of historical data by keeping in the form of time series.

3 Experiments and Findings

An architecture designed to meet the needs of complex systems should be;

- fault-tolerant,
- stable against to human and hardware errors,
- supportive a wide range of different uses,
- scalable and extensible,
- fast to read, write and update operations.

Therefore, our focus in this study was to build an alternative infrastructure to support border security rather than the accuracy of analysis results or performance assessment. The system workflow is as follows:

- read events from Kafka
- deserialize data
- filter irrevalant data
- create a projection to related fields
- store and serve

The analysis of data received from sensors or cameras or unmanned aerial vehicles (UAVs) are important to ensure the area of security. Therefore, in order to test the functionality of the implemented pipeline, at least two different types of data should be processed. So, we have developed TCP client generating sensor data in JSON format as the primary source. As secondary source, we have developed a module which reads video frames and sends them via Kafka.

A sensor data includes twenty different information, such as warnings, altitude, latitude, longitude, remaining battery life, range, identity (id), description, time. The data in JSON format transmitted via Kafka is formatted that consists of sensorId, timestamp, latitude, longitude, range, motion detection status and time. Data modelled are grouped by the sensorId, and then are sorted by the time of submission (Map). Those which cannot be observed changes from the data ordered by time will be deleted (Reduce). As shown in the Fig. 3, a primary key is obtained adding time information to combination of sensorId and date information. Using the primary key is generated daily a row record for each sensor and thus the query operations happen quickly.

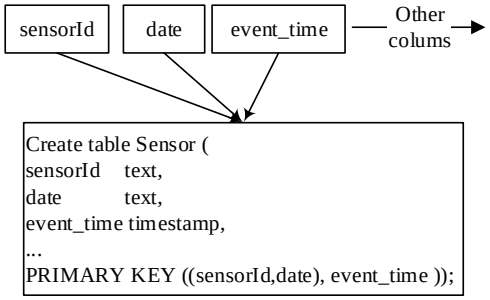

Figure. 3 Sensor Table.

We used JavaCV to analyze image data received as byte array. JavaCV is a library that uses wrapped classes from computer vision libraries via JavaCPP. In image analysis, it is aimed to find faces in each frame, there is no need for a procedure such as Map or grouping of frames. The input video stream is encoded with the H.264 encoder and has 23fps frame rate. Each frame at 1280×720 resolution is consumed by FaceDetector class implemented using Haar feature-based cascade classifiers. Frame which includes at least a face is stored, otherwise it is deleted. The image data is saved to Cassandra with primary key

compounded with the date portion of the timestamp and the sensorId.

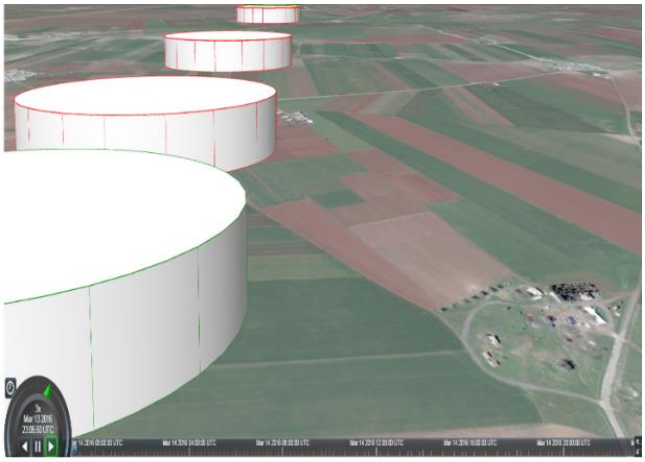

Figure 4. Map screenshot.

Sensor data, shown on the map with Cesium is kept in the form of time series according to the time of submission with their coverage. When updating sensors, changes are matched with time and are added to objects. Sensors are expressed by green. Sensors are expressed by red line, if motion detected.

Cesium has a module specialized for sensors. 3D sensor data that can be expressed in different shapes can stored in memory in the form of time series. Although this feature provides great convenience, it has to run on WebGL based browsers because Cesium is based on WebGL.

Spark is a powerful, fast and robust platform. One of the reasons for the preference is compatible work with the functional programming languages. However, there is a high probability to encounter problems in complex applications. We encountered problems about Spark's object serialization and management of life cycle during development process. We got the java.io.NotSerializableException in application stack trace.

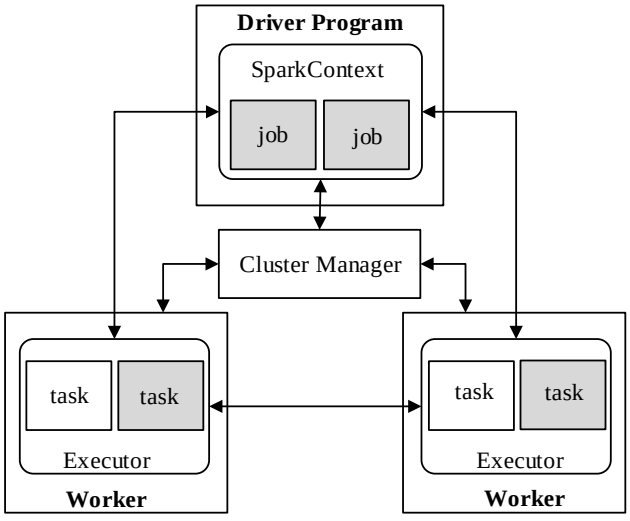

Figure 5. Spark distributed computing mechanism.

As it can be seen from Fig. 5, Spark jobs created on the driver are distributed to workers. The number of jobs is related to the edited logic. The number of tasks is related to the data partition. So, RDD processes work on driver, RDD partitions work on executer. We had to create Kafka objects at every partition to send data which through from Map/Reduce process. If it expressed mathematically; let's consider we had 1000 events per second, 2 second batch range and 16 partitions. Although Kafka will send only 125 posts, there will be instances created 16 times in every 2 seconds. To get rid of this transaction costs derived from Kafka manufacturer, Serializable interface class should be coded and investigated in future studies.

4 Conclusion

In this study, we have developed a system that is consistent, durable, scalable and capable of distributed computing, to help ensure border security. Moreover, the developed application has a significant potential to improve the current practice in emergency related operations as well. This system for real-time streaming data and applications requiring analysis of batch data is a set of interoperable technologies. The evaluation of the performance of the developed system under these circumstances including additional Hadoop Distributed File System (HDFS) for storing results of the image analysis is under progress and Druid will be integrated for storing sensor data in the form of time series for comparative analysis with Cassandra. A detailed report will be prepared about the performance differences before on site integration of the system.

5 References

[1] Aksu, M. and Turhan, F. "New Threats, Expansion of Security Dimensions and Human Security", *International Journal of Alanya Faculty of Business*, Vol. 4, 69-80, 2012

[2] Eker, G. and Yılmaz, G. "Providing Environmental Security Using Wireless Sensor Networks", *TBV BBMD*, 64-71, 2013

[3] Demchenko, Y., de Laat, C. and Membrey, P, *"Defining architecture components of the Big Data Ecosystem", Collaboration Technologies and Systems (CTS),* 2014, 104-112

[4] Engel, Y., and Opher, E., "Towards proactive event-driven computing", *5th ACM international conference on Distributed event-based system (DEBS '11)*, 2014, 125-136

[5] Kreps, J., Narkhede, N. and Rao, J., "Kafka: A distributed messaging system for log processing", *Proceedings of 6th International Workshop on Networking Meets Databases,* 2011

[6] Zaharia, M., Das, T., Li, H., Shenker, S. and Stoica, I., "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters", *4th USENIX conference on Hot Topics in Cloud Computing*, 2012

[7] Chebotko Kashlev, A. and Shiyong, L., "A Big Data Modeling Methodology for Apache Cassandra", *Big Data (BigData Congress), IEEE International Congress*, 2015, 238-245

[8] Nodejs, https://nodejs.org/en/about/

[9] Cesiumjs, https://cesiumjs.org/

[10] The Reactive Manifesto, http://www.reactivemanifesto.org/

[11] Nathan Marz and James Warren, Principles and best practices of scalable realtime data systems, Manning, 2015, 328 pages

[12] Spark Streaming + Kafka Integration, http://spark.apache.org/docs/latest/streaming-kafka-integration.html

[13] Kiran, M., Murphy, P., Monga, I., Dugan, J. and Baveja, S., "Lambda architecture for cost-effective batch and speed big data processing." *Big Data (Big Data Congress), IEEE International Conference*, 2015, 2785-2792