Turkish Journal of Science & Technology
19(1), 161-168, 2024

Research Paper
https://doi.org/10.55525/tjst.1364046

Migration of a Vehicle Tracking System Running on Relational Database to Big Data Environment

Ferhat KOÇER[1], Selim BAYRAKLI[2*]
[1] R&D/Wireless Department, Huawei Telecommunications, Istanbul, Türkiye
[2] Department of Computer Engineering, Air Force Academy Turkish National Defence University, Istanbul, Türkiye
[*1] ferhatkocer25@gmail.com, [2] hbayrakli@hho.msu.edu.tr

(Geliş/Received: 21/09/2023; Kabul/Accepted: 22/03/2024)

Abstract: Building a high-performance and scalable system has always been a challenge in tracking systems. At the root of this problem lies the excessive and real-time data overload. This paper aims to replace traditional approaches with big data approaches. In this study, a new big data ecosystem design for vehicle tracking system architecture is presented. The aim is to process real-time and extremely fast-generated location/tracking data very fast and increase the overall system performance. The process speed performance of the newly developed big data ecosystem is compared with the table query speed performance of a relational database. As a result of the comparison, the query speed of the big data ecosystem was found to be much faster than that of the relational database management system.

Key words: vehicle tracking system, satellite tracking system architecture, big data, cassandra, kafka.

İlişkisel Veri Tabanında Çalışan Araç Takip Sisteminin Büyük Veri Ortamına Taşınması

Öz: Yüksek performanslı ve ölçeklenebilir bir sistem oluşturmak, takip sistemlerinde her zaman bir sorun olmuştur. Bu sorunun temelinde aşırı ve gerçek zamanlı veri yoğunluğu yatmaktadır. Bu makale geleneksel yaklaşımların yerine büyük veri yaklaşımlarını uygulamayı amaçlamaktadır. Bu çalışmada araç takip sistemi mimarisi için geleneksel yöntemlerin dışına çıkarak yeni bir büyük veri ekosistem tasarım ortaya koyulmuştur. Bunu yapmadaki amaç; gerçek zamanlı ve aşırı hızlı üretilen konum/takip verilerinin çok hızlı işlenmesi ve genel sistem performansının arttırılmasıdır. Yeni geliştirilen büyük veri ekosisteminin hız performansı, ilişkisel veri tabanın tablo sorgulama hız performansıyla karşılaştırılmıştır. Karşılaştırma sonucunda büyük veri ekosisteminin sorgulama hızının ilişkisel veri tabanı yönetim sistemine kıyasla çok daha hızlı olduğu görülmüştür.

Anahtar kelimeler: araç takip sistemi, uydu takip sistemi mimarisi, büyük veri, cassandra, kafka.

1. Introduction

The cumulative developments in recent history occupy an important position in human life. Technological developments brought on remarkable changes in our lives and there was a considerable increase in the number of technological software and hardware. This increase led to competition among organizations participating in the race for technology, and this in turn resulted in the introduction of the concept of the internet of things (IoT), which aims for customer satisfaction, to our lives. The internet of things can be defined as network of devices with communication channels which makes it possible to collect or send the data on media, as well as tracking and performing actions over the data [1]. Smart home systems and GPS systems can be given as examples for IoT. Since all this communication network data is borne by the internet, a massive amount of data has been generated and processing of this data has brought up a new problem [2]. This problem, in turn, has introduced us to the concept of big data. This concept was articulated for the first time by Michael Cox and David Ellsworth at the IEEE 8th Conference on Visualization, during which the researchers stated that the excessive amount of data in their study entirely filled their computer's memory as well as external hard drives, referring to the incident as the "Big Data Problem" [3].

The popularity of vehicle tracking systems has gradually increased in recent times. In general, vehicle tracking systems operate over Global Navigation Satellite Systems (GNSS). Determination of location and speed through these systems is useful in search and rescue, target detection, measurement in areas of limited sight, tourism, animal agriculture and many other fields [4].

In order to resolve the 3 main issues related to tracking of vehicles and collection and analysis of data at a single hub, the essential requirements that should be met by the data analyses are as follows:

- Real-time cleansing, addition, organization and interpretation of collected data.
- High speed in data input to the system and data output from the system.

[*] Corresponding author: hbayrakli@hho.msu.edu.tr. ORCID Number of authors: [1] 0000-0002-5313-7903, [2] 0000-0003-3115-6721

- For purposes of customer satisfaction, real-time transfer of information and historical data to users as well as delivery of instant alerts when necessary.

Since data producers generate data at varying formats and intervals, the size of the generated data differs as well. The storage of these varying data types and analysis of real-time streaming data is very difficult through the database management systems and architectures that are currently used. The definition of big data is storage and reading of data in a significant and scalable manner [5].

Vehicle tracking systems are technological systems which make it possible for vehicles and vessels used in road, air and marine transportation to be located and tracked via electronic hardware and software. Tracking devices calculate the vehicle's speed by determining the latitude and longitude values of the vehicle, and send the data acquired from sensors such as RFID, navigation and panic button through the TCP/IP protocol.

In [6], the performance of store and query operations in NoSQL databases is described, with estimates made for the read and write speed using both basic and sophisticated queries. The results show that the performance NoSQL based Apache Cassandra's query read/write processes outperform relational database system.

2. Design of Vehicle Tracking Systems

The structure of vehicle tracking systems is very similar to that of IoT (Internet of Things). It requires for many hardware components to be controlled and used as data producers through the internet. It may not be possible to predict the scaling for the data generated by data producers. Therefore, it was intended for the designed system to be fault-tolerant and solution-oriented. In the 2016 article by Fatih Aydemir and Aydin Cetin which refers to a structure designed for border security, a similar configuration was prepared for border security vehicles [6]. This study, however, intends to address the data intensity that has occurred in vehicle tracking systems through current technology and innovative solutions.

The closest study that has been conducted so far involves the pipeline design by Fatih Aydemir. This study aims to ensure border security and includes a pipeline design in which a data stream is provided between data resources and servers [8]. However, its design is more suited to military purposes. Our study, on the other hand, rather focuses on the field of vehicle tracking. The study is based on the common protocol and modes of operation of vehicle tracking systems.

3. Real-Time Reading of Data from Multiple Resources on The Speed Layer

Many solutions have been created to provide optimization for enabling data transfer, and Apache ActiveMQ is one of these. The data is read and distributed by Apache ActiveMQ and the distributed data is captured by Apache Spark; it is then introduced to the Map/Reduce application and served over ActiveMQ to NodeJS as a topic associated with the JSON language for analysis of processed data such as Web service, mobile access and Web interface.

3.1 ActiveMQ

The ActiveMQ structure involves producers, consumers and brokers. Data producers provide the resource for the data and transmit the data to brokers. Depending on the state of the system, brokers send the data to consumers. Messages can be sent in the queue structure or with a topic. Flow control is performed at the brokers and can be shut off if desired. When flow control is on, the producer enables data delivery by receiving a confirmation message from the consumer. This ensures for the data to be retained at the broker in unfavorable conditions such as a full cache at the receiver. When flow control is off, however, the brokers remain connected to any slow consumers and stopping of the system is prevented. If a fast receiver sends a message to a slow or blocked receiver, the sent message changes direction and the system continues to operate through data transmission to the fast consumer [9].

Devices which are interconnected over TCP/IP and send data in seconds are kept in parallel to Java. Data blockages are prevented through processing at a minimum level. The processing of the collected raw data is carried out with Apache Spark. The data captured with simultaneously operating threads is sent with a queue that is associated with ActiveMQ. In this case, port listeners act as producers.

3.2 Apache Spark

Real-time reading and processing of data in vehicle tracking systems does not always yield all of the necessary solutions. Various capabilities such as interpretation, grouping and customization of the read data, as well as

interventions on it, are required. In order to achieve these, it was intended to perform distributed processing on large amounts of data through the use of Apache Spark. Apache Spark possesses the capability of real-time data processing and retains incoming requests until the action function is recalled; it keeps the acquired data in temporary storage and continues to operate.

The data flow library of Apache Spark is Apache Spark Streaming. In addition to locally supporting many messaging systems such as Kafka, Flume, Kinesis and Sockets, Apache Spark Streaming also supports data reading from external resources [10].

Javed Awan et al. aimed to support retail companies create personalized deals and promotions for their customers, using a big data framework that allows them to handle huge sales volumes with more efficient models [11]. Using the Apache Spark big data framework, they trained two machine learning models: linear regression and random forest to forecast future pricing and sales. At first, they implemented the two models without using the Spark framework and achieved accuracies of 68% and 74%, respectively. Later they trained these models on the Spark machine learning big data framework and achieved an accuracy of 72% for the linear regression model and 81% for the random forest model.

3.3 NodeJS

NodeJS is a popular server-side programming language which can work asynchronously, has a language based on Javascript as well as a powerful, fast and flexible structure, and is integrated with many current libraries [12].

3.4 Openlayers

Openlayers has an open source and powerful mapping structure. It can be directly integrated into maps such as Google Map, BingMap and OpenMap, as well as an external map service. It has a structure designed on layers. Openlayers is a map layering library developed with Javascript. Data which is broadcast with NodeJS is captured with Javascript on the client side and instantaneously reflected on maps through use of the required APIs of Openlayers.

4. Recording Meaningful Data in The Database

The data which is read through being distributed according to the server load status at the consumer with Spark is both broadcast over NodeJS at the speed layer and recorded in the database. At the stage of recording, it is subjected to the Map/Reduce processes of Apache Spark. Apache Cassandra, which carries out column-oriented recording in the NoSQL structure that serves as a big data solution, was selected as the database. Compared to relational databases, NoSQL databases are more advantageous in terms of speed [13]. They are more suitable for batch processing and the internet of things. Various industrial requirements, continuous data production, digitalization of existing data types, consolidation and each similar factor form different resources for big data [14].

4.1 Apache Cassandra

Apache Cassandra is a scalable, open source NoSQL database management system which is designed on the basis of the Apache Cassandra column structure and can be applied on distributed systems. Apache Cassandra is also used by some large companies such as Apple (10 PB of data storage with 75,000 nodes), Netflix (420 TB of data and over 1 trillion daily requests with 2,500 nodes) and eBay (100 TB of data with over 100 nodes) [15].

Apache Cassandra is designed to process big data over multiple data. The multiple node connections of data established between clusters ensures for it to be fault-tolerant. Each node communicates by using the Gossip protocol between pairs and status information is updated continuously. Apache Cassandra allows for an authorized user to access and query a node at any data center and write data by means of the CQL language.

4.1.1 Key Structures of Apache Cassandra

Node: The building block of Cassandra which retains the information as to where the data is kept.

Data center: The group formed by related nodes. Data centers can be virtual or real data centers. Different workloads should use different data centers; this applies for both physical and virtual data centers.

Cluster: Contains one or more data centers. Commit log: All data is first written in the commit log for purposes of durability. Following transfer to SSTables, they can be recorded, deleted or retained for reuse.

CQL Table: Tables formed by rows that are entered in the columns with the primary key.

4.1.2 The Components Required for Configuration of Apache Cassandra

Gossip: The peer-to-peer protocol over which the status and location information of nodes are periodically shared among themselves for exploratory purposes.

Partitioner: Determines in which clusters and nodes the data segment copies of nodes will be kept. A partitioner is the function which subjects the relevant row and its primary key to the hash process.

Replication factor: The total replication number for clusters. A replication factor indicates only one-time replication for each row in the node. The replication amount increases in proportion to this value. All copies are equally important and there are no primary or backup copies. In general, the number of copies is higher than 1 and lower than the number of clusters.

System operating tables: These allow for changes to be made on the main tables in order to modify the storage properties through use of the CQL language.

Replica placement strategy: Recording copies are kept in multiple nodes. Replica placement strategy determines the location of the nodes in which the copies will be kept.

Snitch: Defines the machine groups at the data center and the topologies in which replica placement strategy is used. Cassandra.yaml configuration file is the main configuration file. It is the file in which many adjustments such as property identification for clusters, cache parameters for their tables, resource use parameters for performance, timeout, client connection limitations, backup and security are carried out.

5. System Design and Modules

The basis of the structure design relies on adaptability, flexibility and sustainability of the system. The properties which the system should possess in the structure design are as follows:

- minimizing the interaction time between clients and devices,
- performing a higher number of tasks with a lower amount of load through correct use of system resources,
- increasing data reliability,
- performing a higher number of tasks with a lower amount of work through correct use of system resources,
- facilitate system growth in parallel to data growth by forming a horizontally scalable system.

Vehicle tracking systems require real-time and bi-directional processing on real big data. Therefore, Apache Spark was chosen to process the real-time data. The data of vehicle tracking hardware, which is sent with TCP over GSM, was received with multiple sockets via ports and the use of parallel programming capabilities. Due to being open source, powerful, free of charge and fault-tolerant, Apache ActiveMQ messaging service was chosen to distribute the received data over servers. For analysis of the raw data captured with ActiveMQ, Apache Spark was chosen due to being the most suitable structure for real-time parsing and easy integrability with Cassandra, which is being used to store the data.

5.1 Design of the Structure

In structure design, each segment should perform its duty in the form of small modules and the system should be as sustainable and scalable as possible. The stages of the structure completed according to this principle are as follows.

Stage 1: The raw input received from multiple data providers with ActiveMQ is assimilated with the queue and in a manner which is independent from the receiver.

Stage 2: The data which is read over the queue is combined with Apache Spark and subjected to the Map/Reduce process.

Stage 3: The data which is analyzed and interpreted with Spark is assimilated and broadcast if it has a receiver, and all data is recorded in the Cassandra database.

Stage 4: The parsed data is read at the consumer with NodeJS and sent to the relevant user with the Socket.IO library over WebSocket. NodeJS allows application development on the server side with the JavaScript programming language and is suitable for asynchronous tasks. Its powerful structure ensures for it to process instantaneous operations with low resource consumption and high performance.

Stage 5: Multiple map integration is performed through use of Openlayers within the web page with a user interface. Openlayers is an open source, free of charge map layering library which is integrated with multiple maps

and developed with Javascript. Data which is broadcast with NodeJS is captured with Javascript on the client side and instantaneously reflected on maps through use of the required APIs of Openlayers.
The stages provided above show the data analysis and its reflection on the user panel; the flow chart of the whole process is given in Figure 1.

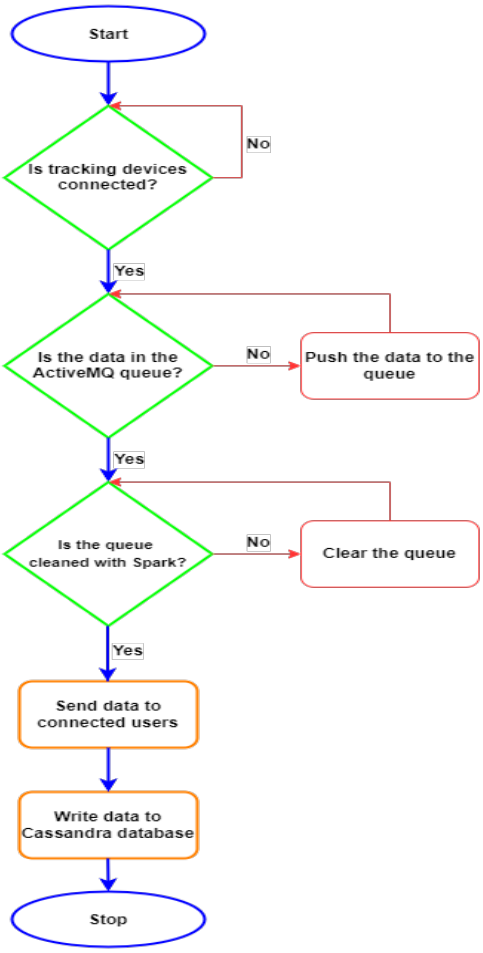

Figure 1. Flow chart of the migration process

5.2 Instantaneous Delivery of Data

In instantly delivered data, it is possible for there to be congestion and bottlenecks in the system due to changes in the data transmission frequency of data providers. In order to prevent this, the listener processes each received message at the minimum level before transferring it to the queue. Each connection on the port is assigned to the independent thread and the producer initiates data flow by using the capabilities of parallel programming.

The data from the data providers is received through use of ServerSocket and Socket classes with TCP. Each device that establishes a connection continues the process as an independent thread with a derivative object from its own Socket class. The data which is read over the port is transferred to the queue in a similar manner to threads. A failover operation is performed during data transfer with ActiveMQ to ensure data delivery to servers in situations such as interruption of instantaneous communication or timeout among servers.

5.3 Processing, Interpretation and Cleansing

In vehicle tracking systems, it is not always sufficient for the data to be instantaneously read and processed. Various capabilities such as interpretation, grouping, customization and intervention on the read data should also be provided. In order to achieve these capabilities, it was intended to perform distributed processing on large-scale

data through the use of Apache Spark. Apache Spark keeps incoming requests until the action function is recalled (lazy evaluation). It keeps the acquired findings in temporary storage and continues to operate without waiting.

Spark Stream forms a DStream (Discretized Stream) for the streaming data in the Apache ActiveMQ and Apache Spark integration. DStreams are series of the same type in which RDDs (Resilient Distributed Datasets) are continuously created. The data on ActiveMQ are deleted from the queue after the required operations are carried out. This also ensures cleansing in the queue.

5.4 Data Storage and Broadcasting

Apache Spark and Cassandra integration was provided by the Spark Cassandra Connector library. Due to operating in coordination with Apache Spark, this library also carries out recording processes with Map/Reduce. In order for the data of end users to be shown in real time over the system, the data is broadcast in JSON format over ActiveMQ before being recorded in the database. When broadcasting is performed with a topic, delivery to the subscribers is only carried out if the broadcast has any current subscribers. In order to capture the broadcast data, NodeJS acts as an ActiveMQ client. In order to prevent unnecessary use of system resources through broadcasting of all data, the user information which is active over the system is delivered to the software which interprets data over ActiveMQ with NodeJS.

NodeJS pairs the relevant data with the relevant users. Socket.IO was used to provide communication between the end user and the web client. With Socket.IO, a request can be returned in case of data arrival on the server side when there are no requests on the user side. Socket.IO allows bi-directional and simultaneous communication. Location data captured on the client side is positioned and reflected on the map through use of Openlayers.

Requests such as speed limit adjustment, region identification and data transmission to sensors connected to vehicle tracking hardware (VTH) are required for VTH on the data client side. It is required for the system to send a request to the server on the web client side. In this case, the operations are carried out in the opposite direction to the process of data acquisition from VTH. The web client sends its message to NodeJS with Web Socket. NodeJS attaches a queue with ActiveMQ and performs broadcasting. The message is read over Spark and recorded in Cassandra with Resilient Distributed Datasets (RDDs); delivery to the relevant devices is simultaneously carried out over ports and the TCP listener.

5.5 Configuration

After the reading and interpretation operations in the data sources are concluded, the data is transmitted in JSON format. The data acquired from VTH and other sensors is recorded in the table of locations through use of the IMEI (International Mobile Equipment Identity) number of VTH and the time at which the incident occurred as a primary key. The data which is sent to web clients on the NodeJS side with Socket.IO is reflected on the maps through use of the Javascript APIs of Openlayers. The map shows the location, speed, status and direction information for vehicles with icons (Figure 2).

Figure 2. Openlayers map view

6. Results and Conclusion

The conducted research led to Apache Cassandra, a column-oriented NoSQL solution, to be chosen as the database for recording the data. Apache Cassandra was preferred due to being fault-tolerant, distributed, scalable and compatible with many software products. This study involves a comparison between Apache Cassandra and the relational database MYSQL. The same tables were generated on the two databases. MYSQL and Cassandra were installed on two different virtual machines of the same properties. The virtual machines in question ran the Ubuntu 14.04 LTS operating system. Cassandra was chosen because it was fast in adding, deletion and reporting processes, as well as allowing operations over the columns. It is designed to provide high write and read throughput. It excels in write-heavy workloads and can handle a large number of concurrent writes across the distributed cluster. With this feature, Apache Cassandra is perfect for targeted work.

The results of the INSERT SQL-CQL queries ran on the databases are shown in Figure 3. It is impossible for the INSERT operation intensity to be predicted for IoT structures such as vehicle tracking systems. It is critical for the operations to be performed in the lowest amount of time possible in order for the system resources to be used correctly and data loss to be prevented. The INSERT operations which were carried out indicated that Cassandra yielded better results in terms of speed for each query directed at the two database systems installed on the same type of system.

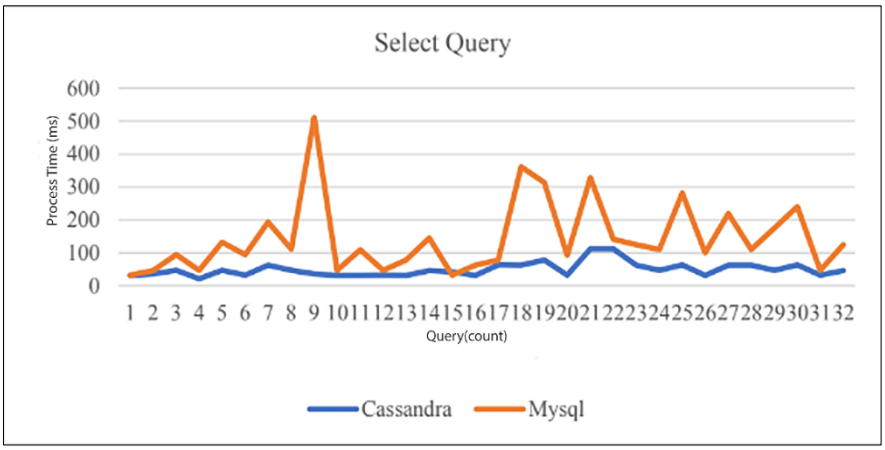

Figure 3. MYSQL-Cassandra record fetch time comparison

Data records from 16 different devices were kept for a period of 6 months for both systems. The following queries were ran in the two systems on completely identical data records and the relevant time was measured (Figure 4).

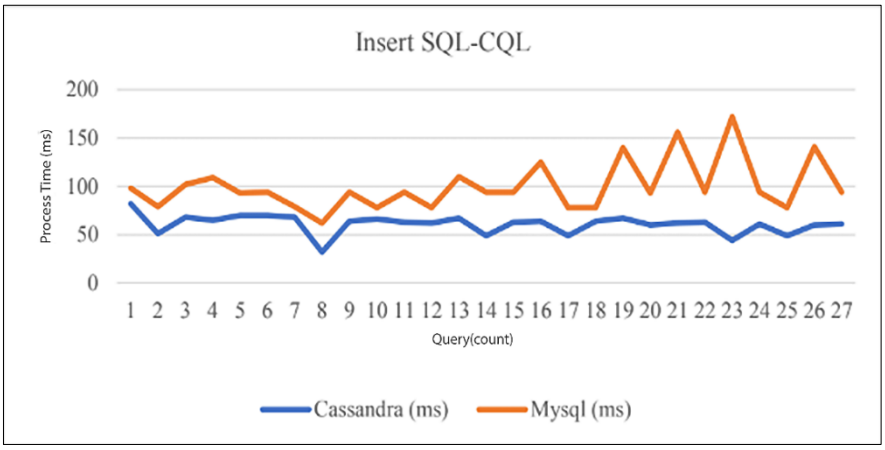

Figure 4. MYSQL-Cassandra insert record comparison

References

[1] Giusto D, Iera A, Morabito G, Atzori L. (Eds.). The internet of things: 20th Tyrrhenian workshop on digital communications. New York, USA: Springer Science & Business Media, 2010.

[2] Goes PB. Design science research in top information systems journals. MIS Q.: Manag. Inf. Syst. 2014; 38(1): iii-viii.

[3] Cox, M., & Ellsworth, D. Application-controlled demand paging for out-of-core visualization. In: Proceedings. Visualization'97 (Cat. No. 97CB36155); 1997 October; Phoenix, AZ, U.S.A. New York, USA: IEEE. pp. 235-244.

[4] Koca B, Ceylan A. Uydu konum belirleme sistemlerindeki (GNSS) güncel durum ve son gelişmeler [Current Status and Recent Developments in Global Positioning Satellite Systems (GNSS)]. Geomatik 2018; 3(1); 63-73.

[5] Eger Ö. "Big Data'nın (Büyük Veri) Endüstriyel Kullanımı". Türkiyenin endüstri 4.0 platformu. 2017. https://www.endustri40.com/big-datanin-buyuk-veri-endustriyel-kullanimi (accessed March 28, 2018).

[6] Kumar, M. Sandeep. "Comparison of NoSQL database and traditional database-an emphatic analysis." JOIV: International Journal on Informatics Visualization 2.2 (2018): 51-55.

[7] Aydemir F, Cetin A. Designing a Pipeline with Big Data Technologies for Border Security. Mugla Journal of Science and Technology 2016; 2(1): 98-101.

[8] Aydemir F. Sınır Güvenliği İçin Büyük Veri Teknik ve Teknolojileri ile Boru Hattı Tasarımı [Designing a pipeline with big data Techniques and technologies for border security] (Unpublished Master Thesis). Gazi University, Ankara, Turkey, 2012.

[9] Apache Active MQ, 2015. https://activemq.apache.org (accessed October 28,2018).

[10] Apache Spark. Apache Spark-Unified Analytics Enginefor Big Data. Apache software foundation. 2018. https://spark.apache.org/ (accessed November 11, 2018).

[11] Javed Awan M, Mohd Rahim MS, Nobanee H, Yasin A, Khalaf OI. A big data approach to black friday sales. Intell. Autom. Soft Comput. 2021; 27(3): 785-797.

[12] Linux foundation. Node.js. 2018. https://nodejs.org (accessed March 29, 2018).

[13] Öztürk S, Atmaca HE. İlişkisel ve ilişkisel olmayan (NoSQL) veri tabanı sistemleri mimari performansının yönetim bilişim sistemleri kapsamında incelenmesi. [The examination of relationaland non-relational (NoSQL) database system's architectural performances in terms of management of information systems]. Bilişim Teknolojileri Dergisi [Journal of Information Technologies] 2017; 10(2); 199-209.

[14] Aktan E. Büyük veri: Uygulama alanları, analitiği ve güvenlik boyutu [Big data: Application areas, analytics and security dimension]. Ankara Üniversitesi Bilgi Yönetimi Dergisi [Ankara University Journal of Information Management] 2018; 1(1): 1-22.

[15] Apache Cassandra. "Apache Cassandra Unified Analytics Engine for Big Data". Apache software foundation. 2016. http://cassandra.apache.org/ (accessed November 29, 2018).