International Journal of Smart Device and Appliance
Vol.8, No.1 (2020), pp.1-8
http://dx.doi.org/10.21742/ijsda.2020.8.1.01

Data Processing System Using CQRS Pattern and NoSQL in V2X Environment

Sangkon Han[1]and Jung-In Choi[2]

[1]Department of Computer Science Engineering, Pusan National University, Busan, Republic of Korea
[2]Software Education Center, Pusan National University, Busan, Republic of Korea
[1]sangkon@pusan.ac.kr, [2]jungin.choi@pusan.ac.kr

Abstract

The vehicle-to-everything (V2X) environment comprises a variety of devices that exchange or share vehicle-to-vehicle, vehicle-to-infrastructure, and vehicle-to-pedestrian information while the vehicles are being driven. In V2X environments, large amounts of unstructured data are recorded by various devices. Storing large amounts of unstructured data in a relational database would require all the data generated by the different types of devices to be normalized. In addition, the transaction-based Atomic, Consistency, Isolation, and Durability (ACID) characteristics of relational databases would not be suitable for processing large amounts of data of this nature. In this study, we apply the Command and Query Responsibility Segregation (CQRS) pattern to separate commands and queries and to process large-scale data efficiently. NoSQL, which provides specialized functions for handling unstructured data, as well as the commands and queries, can be separated; thus, NoSQL and RDBMS can be selectively used depending on the V2X environment and system characteristics. We propose a data processing system suitable for the V2X environment using a CQRS pattern and the NoSQL-based database. By applying the CQRS pattern, NoSQL and RDBMS can be used together. When storing large amounts of data, use NoSQL, and when providing information or statistics to users, select an RDBMS to use together. In addition, both repositories can be easily scaled up when processing large amounts of data, making efficient use of resources.

Keywords: *Command and query responsibility segregation pattern, CQRS, RDBMS, vehicle-to-everything environment, V2X*

1. Introduction

The vehicle-to-everything (V2X) environment comprises a variety of devices that exchange or share vehicle-to-vehicle, vehicle-to-infrastructure, and vehicle-to-pedestrian information while the vehicles are being driven [1]. Domestic and overseas telecommunications companies and the manufacturing and service industries related to the battlefield business are devoting considerable effort to applying information exchange to the V2X environment in various ways by combining IT technologies.

V2X refers to all forms of communication related to the vehicle. Because the technology related to V2X has been developing rapidly, a standard has not yet been established. Furthermore, because data are exchanged among various types of devices, not only is the

Article history:
Received (February 2, 2020), Review Result (March 10, 2020), Accepted (April 14, 2020)

Print ISSN: 2288-8977, eISSN: 2207-404X IJSDA

amount of data large but the data format is not standardized. Therefore, the collection and processing of these data would be expected to be difficult [2].

This paper proposes a design for a data processing system for efficient data storage and processing by referring to the command and query responsibility segregation (CQRS) architecture pattern [3] to store and process the data of various formats that are generated in the V2X environment.

2. Data processing system using CQRS and NoSQL

2.1. Necessity of CQRS

When developing and operating an application, it is often necessary to incorporate a wide variety of business logic into a variety of examples and behavioral variations. This motivated us to define and introduce a domain model to efficiently reflect complex business logic such as the data structure, association, and system constraints used in an application. The domain model consists of a network of components used in business logic, and each component comprises a mixture of data and processes [4].

As the business logic is modified or changed, the complexity and maintenance costs of the domain model continue to increase. First, the domain model is different from the originally designed direction or intention. For example, for a complex user experience, data structures that are not related to business logic may be involved, or the process may become too large to meet the rapidly changing requirements [5].

The most fundamental problem with applications based on the Create, Read, Update, Delete (CRUD) design is that changes and queries occur in one place. A CRUD-based database may not guarantee consistency of data between the time the data are modified or added and the time the data are read and displayed on the screen.

The Relational Database Management System (RDBMS) provides transactions for consistent data. A transaction is a unit that processes a query in a minimum unit, performs a rollback that is executed from the beginning if the execution is interrupted in the middle, and executes a commit when the execution completes without errors. A transaction is a logical unit of work in which all or nothing is executed when the query is executed [6]. Transactions have four characteristics: Atomicity, Consistency, Isolation, and Durability (ACID) First, atomicity is all or nothing, regardless as to whether the transaction is reflected in the database. Second, consistency means that, even if a database changes during the course of a transaction, the transaction does not proceed to the updated database, but proceeds to the referenced database for the first transaction. Third, independence means that two or more transactions cannot interfere with the operation of another transaction if they are running concurrently, and another transaction cannot reference the result of a particular transaction until the first transaction is completed. Fourth, persistence means that if a transaction is successfully completed, the results must be reflected permanently [6]. Transaction ACID is an important advantage and feature of RDBMS because it guarantees data consistency.

In recent times, the amount of data generated in small devices such as those connected to the IoT and V2X networks, including mobile devices, has increased rapidly, and this has led to increasing interest in big data [7]. Big data not only refers to the size of the data, but also to data of various formats, including formal data and unstructured data. To extract valuable information from big data and analyze the data, various methods have been proposed to store the data [8]. However, existing RDBMS cannot accommodate ACID-based transactions.

Because data is frequently inserted or modified, RDBMS uses a lock to guarantee transactions consistency, which slows down the reading of data [9].

Although different opinions exist about NoSQL [10] based on opinions, this study defines a solution based on Not Only SQL [11]. NoSQL does not provide the ACID properties [6] provided by RDBMS, it has consistency, availability, and partition tolerance characteristics based on CAP theory [12]. Because NoSQL does not guarantee the same characteristics as ACID, it is necessary to choose the NoSQL database specifically for your intended purpose and choose a database that fits your reason for storing data, such as the requirements of consistency and validity.

The CQRS architecture pattern, which provides a way to select a database that meets its purpose and that processes the data in an efficient manner using that database, provides a transaction for data consistency in the CRUD-based repository of RDBMS, and recognizes the problem that arises. CQRS separates commands and queries from each other. It is essential to separate responsibility from processing commands and responsibility from processing queries [3]. CQRS has been developed in command-query separation (CQS) Because CQS cannot avoid side effects when using an imperative language, we propose a method to separate a procedure with side effects and a pure function without side effects to minimize the effect of side effects. The basic concept of the CQS principle is to divide the methods of an object into commands and queries according to whether side effects occur [13].

Young introduced CQRS as a very simple pattern [14]; thus, CQRS is an intuitive pattern. The main difference between CQS and CQRS is that CQS is separated from the object, system, or component level if the command and query are separated based on the operator [15]. Because CQRS can separate queries and commands at the system level, it is ideal for environments in which large amounts of data need to be modified and the information must be processed, such as V2X environments. This prompted us to develop a system configuration that associates the CQRS pattern with the NoSQL database.

2.2. NoSQL system with CQRS pattern

Data generated on various devices such as those participating in a V2X environment do not have a standard format defined. In addition, because these data are generated by many devices, transaction-based RDBMS may present a bottleneck when attempting to store a large amount of data and these databases are not scalable.

NoSQL provides a flexible data model, which is specialized for storage and retrieval, and is easy to extend, making it ideal for distributed environments such as V2X environments or for storing large amounts of data. However, relational databases are still in widespread use despite the emergence of databases that provide a variety of purposes and functions, such as NoSQL. Relational databases are still used in services where data accuracy is essential, such as enterprise ERP, MIS, and CRM. This is because of the convenience of SQL, but the CAP theory, which is a feature of NoSQL, cannot replace ACID to replace RDBMS [12].

However, because NoSQL does not provide a database that satisfies all the characteristics of ACID, it is necessary to store data in a combination of its specialized functions. Because CQRS can be separated at the system level unlike CQS, CQRS patterns can be easily applied according to the characteristics of NoSQL. Based on the CAP theory of NoSQL, the system can be configured by reflecting the characteristics required for inquiries and commands.

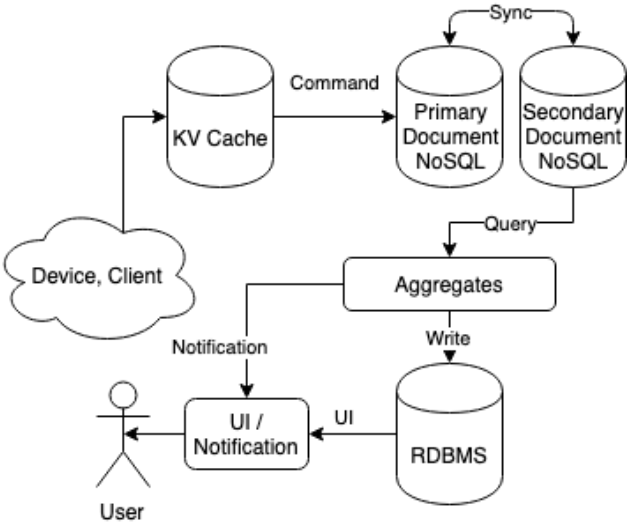

Figure 1. NoSQL system with CQRS

[Figure 1] presents a graphic representation of the NoSQL system with CQRS, showing the components and data flow of the data processing system in which the CQRS pattern is applied to the NoSQL system. Information on the device, client, etc., are temporarily stored in NoSQL based on the KV Cache. By storing data in a general-purpose distributed cache based on a KV cache, a large amount of data can be stored at high speed, enabling quick processing and a rapid response to a device or a client in the process of transmitting the data. In addition, when using KV Cache, it is advantageous to perform multiplexing when the amount of data increases or surges. The approach enables fast storage, allows easy expansion, and permits rapid responses to transmissions.

Document NoSQL is used to persist data temporarily stored in the KV Cache. The primary storage is used to read the necessary data from the KV Cache, to store the data, to modify or delete specific data, and to use the secondary storage as read-only. It is necessary to synchronize the repositories to separate them for primary and secondary storage. In the case of NoSQL, most databases support self-replication, considering scalability.

Aggregates handle the necessary data in a read-only store. The most decisive reason for separating write and read is that the relationship between the data in the aggregate needs to be changed at any time. If the standard of collected data such as those acquired in the V2X environment is not constant, and the same equipment is not constantly connected to the network, the method that is used to process the data or the necessary rules may be changed at any time. Therefore, read-only storage is required to recalculate data at any time, and data processing steps such as Aggregates [Figure 1] are required to apply data processing rules in the repository. In addition, write and read operations should be separated to prevent them from interfering with data collection because of the throughput that occurs while the data is being processed.

Prior to presentation of the processed information to the user (shown in [Figure 2]), it is necessary to convert the data into standardized data before storage in the RDBMS. Car Dashboard or Car Information is used to output the necessary data after storing the data in the RDBMS based on the data processed by Aggregates. Notifications, on the other hand, send notifications directly to users in Aggregates only when immediate notification is needed during data collection and processing. When the UI provides the necessary information to the user, if the specification of the data is not constant as in NoSQL, it is costly to process the UI, and

because it is not efficient to express the data, correct information cannot be provided. A user who operates a database would prefer to use an efficient database, such as RDBMS, for regular data.

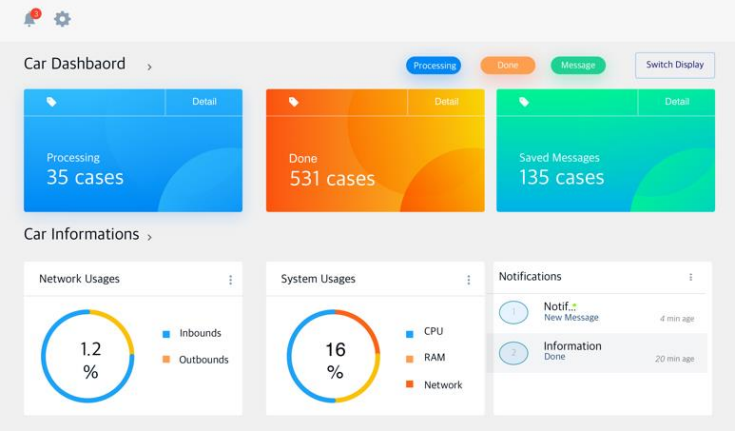

Figure 2. User interface that displays processed data to the user

The V2X data processing system presented in this paper was tested using Azure operated by Microsoft. All RDBMS and NoSQL used in the experiment used the managed service provided by Azure. We tested the data processing system by creating a generator that could randomly generate both regular and unstructured data in the JSON standard.

When the test was initially conducted using 10,000 data items, excessive traffic was generated in Azure. The test was therefore mistaken for a DDoS attack and could not be completed. We then tested 10,000 locales in the Windows server using the message queue and Docker software. No significant difference in performance between RDBMS and NoSQL could be detected when the test size was approximately 10,000.

The number of messages was increased to approximately 100,000, and the server was started on Ubuntu. As the amount of transferred data items exceeded 100,000, a bottleneck occurred at the network interface card (NIC). Thus, the buffer size of the TCP socket in the Linux kernel configuration and the ARP response method were adjusted according to the NIC latency. Not only the message size but also unsigned data were included based on the JSON standard. For the tests involving unstructured data, the UI functioned smoothly using the NoSQL database based on the CQRS pattern compared to using a single RDBMS. In the case of the RDBMS, the message processing speed was not constant because of the aforementioned bottleneck in the message processing speed. However, the proposed system succeeded in maintaining message processing speed at a constant level.

3. Conclusion

We propose a data processing system suitable for the V2X environment using a CQRS pattern and the NoSQL-based database. In order to increase the efficiency of the processing system when selectively using various databases, the CQRS pattern was applied to separate roles related to commands and queries. The CQRS pattern can be efficiently applied to select the required database according to the size, type, and scale of the data. Because the query and the command are separated, more resources can be allocated where required by considering the

system utilization, and this is advantageous for scaling as compared to a single RDBMS or NoSQL structure.

The system proposed in this paper uses the NoSQL database to process data generated from various devices and to easily expand the data for large-scale data processing. In addition, RDBMS is applied where standardized data such as user UI is needed. Assuming a V2X environment, more intense tests should be reflected. However, the higher the number of messages, the more likely it is to be mistaken for DDoS attacks in cloud systems, and the bottleneck was caused by the Network Interface Card (NIC) rather than the expected RDBMS. There is a need for conducting quantitative testing in environments where various devices such as the V2X are connected.

In the future work, quantitative test methods should be studied to compare the efficiency of real-time data processing in KV Cache as well as Aggregates and how these data can be processed in real time.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07042967)

References

[1] D. G. Choe and J.H. Jung, “Vehicular networking and applications for connected cars," Communications of the Korean Institute of Information Scientists and Engineers, vol.37, no.1, pp.45-53, **(2019)** DOI: 10.5626/KTCP.2019.25.10.500

[2] N. Cheng, et al, “Big data driven vehicular networks,” IEEE Network, vol.32, no.6, pp.160-167, **(2018)** DOI: 10.1109/mnet.2018.1700460

[3] M. Fowler, “CQRS,” URL: https://martinfowler.com/bliki/CQRS.html, Checked on 2019-08-01, **(2011)**

[4] M. Fowler, “Patterns of enterprise application architecture,” Addison-Wesley Longman Publishing Co., Inc., **(2002)**

[5] E. Evans, “Domain-driven design: tackling complexity in the heart of software,” Addison-Wesley Professional, **(2004)**

[6] R. Elmasri, “Fundamentals of database systems,” Pearson Education India, **(2008)**

[7] J. Manyika, “Big data: The next frontier for innovation, competition, and productivity,” https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation, Checked on 2019-08-01, **(2011)**

[8] R. Zafar, et al, “Big data: the NoSQL and RDBMS review” Proceedings of International Conference on Information and Communication Technology (ICICTM), IEEE, pp.120-126, **(2016)** DOI: 10.1109/icictm.2016.7890788

[9] V. N. Gudivada, D. Rao, and V. V. Raghavan, "NoSQL systems for big data management," 2014 IEEE World congress on services, IEEE, pp.190-197, **(2014)** DOI: 10.1109/services.2014.42

[10] J. Han, et al, “Survey on NoSQL database,” Proceedings of the 6th international conference on pervasive computing and applications, IEEE, pp.363-366, **(2011)** DOI: 10.1109/icpca.2011.6106531

[11] S. George, “NoSQL–NOT ONLY SQL,” International Journal of Enterprise Computing and Business Systems, vol.2, no.2, **(2013)**

[12] C. Kong, et al, “Acid encountering the cap theorem: Two bank case studies,” Proceedings of the 12th Web Information System and Application Conference (WISA), IEEE, pp.235-240, **(2015)** DOI: 10.1109/wisa.2015.63

[13] B. Meyer, “Object-Oriented Software Construction,” Prentice Hall, **(1997)**

[14] Y. Greg, "CQRS Documents," https://cqrs.files.wordpress.com/2010/11/cqrs_docume nts.pdf, Checked on 2019-08-01, **(2010)**

[15] D. Esposito, "Cutting edge - CQRS and message-based applications," https://msd n.microsoft.com/magazine/mt238399, Checked on 2019-08-01, **(2011)**

This page is empty by intention.