Performance of Deeply Analyzing Application Switch

Satoshi ITO[†a)], ***non-Member***, **Akihiro NAKAO**[††], **Masato OGUCHI**[†††] ***Member***, ***and*** **Saneyasu YAMAGUCHI**[†], ***Member***

1. Introduction

Kanaya et al. have proposed a method of caching KVS(Key-Value-Store) data in application switches to improve the performance of Cassandra, which is a database management system based on a key-value store, running on a network. In the method, a developer implements a function for analyzing Ethernet frames in an application switch. The switch analyzes the frames, extracts query and response data, and caches the data in the switch to improve the response performance.

However, this existing method does not analyze lexically and syntactically frames. It extracts information from the data at fixed addresses in the frame payload. Therefore, variable-length table names, key names, and values cannot be analyzed. This means that a user must use columns and column families with specified lengths. This issue can be solved by implementing lexical and syntax analyzers in an application switch. This paper evaluates the overheads of these analyses on KVS performance and shows that our implementation achieves the analysis with a very small performance decline.

2. Analysis of Overhead

We have implemented a scanner and parser in the application switch that can analyze Cassandra queries. This enables the extraction of table names and other information from queries composed of variable-length fields. We evaluated the KVS performance of three methods, which are the normal method, the existing method [1], and the existing method with a scanner and parser. The normal method only forwards frames based on the destination MAC address at the switch. The key was selected according to the Zipf's law. Fig. 1 shows the average turnaround times of 1024 queries of the methods. In this experiment, the cache hit ratio was 37%. The "existing method + analysis" shows the results in the case of cache miss including analytical overhead. Comparing the results of "existing method (cache miss)" and "existing method + analysis," we can see that the increase in the turnaround time, which is the size of the performance decline, caused by these analyses was not large. On the other hand, the size of the decrease in turnaround time at the cache hit of the existing method is large compared to the increase caused by the analyses. Therefore, we expect that an application switch with the analyses effectively improves the KVS performance.

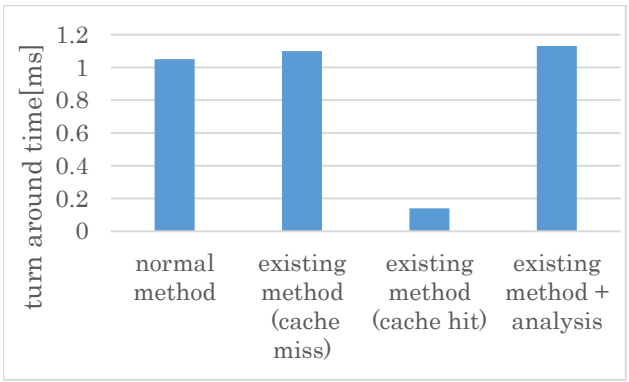

Fig. 1 KVS turnaround time

3. Conclusion

In this paper, we introduced a method for improving KVS performance by implementing a caching function in an application switch and its issue that it supported only fixed-length fields. We argued that this issue could be solved by implementing a scanner and parser, and that these analyses caused processing overhead. We implemented an application switch including these analyses and showed that the overhead and the size of performance decline were not large. This implies that an application switch can support variable-length fields with a small overhead and it is a promising way for improving the KVS performance.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 21K11854 and 21K11874.

References

[1] T. Kanaya, A. Nakao, S. Yamamoto, M. Oguchi, S. Yamaguchi, "Intelligent Application Switch and Key-Value Store Accelerated by Dynamic Caching," In Proceeding of 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC 2020), DBDM 2020: The 5th IEEE International Workshop on Distributed Big Data Management, pp. 1318-1323, Jul. 2020.

[†]The author is with Kogakuin University
[††]The author is with the University of Tokyo
[†††]The author is with Ochanomizu University
[a)] E-mail: cm22009@ns.kogakuin.ac.jp