Query-driven compaction in LSM-trees

Shubham Kaushik
Boston University
USA

****
Boston University
USA

****
Boston University
USA

ABSTRACT

In modern data systems, particularly for write-intensive workloads, log-structured merge (LSM)[1, 2] trees have become the most widely used technique. The idea of out-of-place updates, which logically invalidate keys that may exists on multiple levels, helps LSM-trees excel for write heavy workloads. However, current LSM-tree designs do not capitalize on the sort-merge operations performed for the range queries, which leads to carrying out almost the same amount of work for each range query, even when they are same or overlapping. This inefficiency can result in the wastage of CPU cycles and unnecessary I/O operations in worst-case scenarios due to presence of logically invalid keys.

In this paper, we present a query-driven compaction strategy that removes the invalid (logically deleted) keys from the LSM-tree and flushes back the valid keys filtered by sort-merge performed during a range query. We conduct performance experiments with the help of a custom implementation in one of the popular LSM-based data stores, RocksDB.

ACM Reference Format:
Shubham Kaushik, ****, and ****. 2023. Query-driven compaction in LSM-trees. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Write heavy workloads with LSM. The rising research in the field of robotics and IoT devices, combined with the integration of artificial intelligence and machine learning, has resulted in the generation of vast amounts of data. This data often requires real-time processing and exhibits a write-heavy nature. The data stores like RocksDB and LevelDB have been designed to efficiently handle such workloads. These data stores rely on the technique of **log-structured merge (LSM)** trees, an efficient data structure tailored for managing write-heavy workloads. The fundamental concept underlying LSM trees is of out-of-place updates, which logically invalidate keys instead of performing in-place updates.

Compaction. The LSM trees are composed of multiple levels, each of which is a sorted run of key-value pairs. The compactions are performed to merge the sorted runs from the lower levels into the higher levels. The process is triggered when the size of a level exceeds a certain threshold. It helps in removing the stale data from LSM and making room for new data in lower levels.

Conference'17, July 2017, Washington, DC, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Range queries. The LSM design is optimized for write-heavy workloads, but it also supports range queries. The range queries are performed by merging the sorted runs from multiple levels and filtering out the keys that have been logically invalided. This process is also called **sort-merge**, which is similar to the compaction.

Problem. When executing a range query, the sort-merge operation is performed on sorted runs from multiple levels, leading to the retrieval of both valid and invalid keys from higher levels. Consequently, more data is read than actually required. This situation is acceptable when performing a single range query for a specific range. However, when the same query or an overlapping one is executed repeatedly, a nearly identical amount of work is duplicated unnecessarily. Furthermore, when a compaction is triggered within that specific range, some of the invalid keys that were previously read and sorted during the range query are revisited. This results in redundant CPU cycles and I/O operations, where the same data bytes are read repetitively until reaching the last level.

1.1 Motivation

The repetition of work involving invalid keys can be effectively mitigated by redirecting valid keys, filtered through the sort-merge operation during a range query, back to the lower levels. This approach can be termed as **query-driven compaction**. By minimizing the presence of invalid keys within the LSM tree, the compaction process gains efficiency, subsequently leading to less number of I/O operations and a more optimal utilization of CPU cycles.

1.2 Problem Statement

...

1.3 Contributions

...

2 BACKGROUND

3 PROBLEM

4 SOLUTION NAME

5 EVALUATION

6 RELATED WORK

7 CONCLUSION

REFERENCES

[1] N. Dayan, M. Athanassoulis, and S. Idreos. Monkey: Optimal navigable key-value store. In *SIGMOD Conference*, pages 79–94, New York, NY, USA, 2017. Association for Computing Machinery.

[2] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.