RocksFS: Reducing Log Harm on LSM-Tree based Key-Value Store

Haitao Wang, Zhanhuai Li, Xiao Zhang, Zhen Chen, Xiaonan Zhao
School of Computer Science, Northwestern Polytechnical University, Xi'an Shaanxi China

Abstract

Nowadays, persistent key-value (KV) stores play a critical role in a variety of modern data-intensive applications, such as web indexing, e-commerce, and cloud data storage, etc. KV stores serve as the foundation for a growing group of important applications by enabling efficient insertions, point lookups, and range queries. RocksDB[1] is a typical and popular open source KV store which features highly flexible configuration settings that can be tuned to run on a variety of production environments, including pure memory, Flash, hard disks or HDFS. Owing to the advantages of RocksDB, it is widely used in Facebook and other distributed storage systems, such as Ceph[2].

However, RocksDB also has some performance issues. On one hand, RocksDB needs to leverage write-ahead-log (WAL) files to guarantee the atomicity of write and enable recovery in case of a crash, leading to additional log overhead. On the other hand, RocksDB uses local filesystem (e.g., Ext4) to store data, which can harm its performance because of unnecessary operations of local filesystem. The update of WAL files will introduce a huge filesystem overhead, particularly for small insert-intensive workload, which severely degrades the performance of RocksDB.

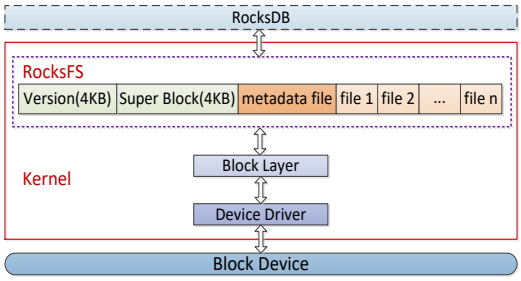

Figure 1: Architecture of RocksFS.

In order to address these issues, we present RocksFS-a simplified filesystem optimized for WAL writes to reduce the log overhead. The architecture of RocksFS is illustrated in Figure 1. The basic idea behind RocksFS is preallocating space for WAL files to lower the update frequency of metadata (i.e., WAL file size). RocksFS leverages a new WAL format and preallocates space for WAL file. In particular, it encapsulate every WAL write to an encoded transaction which has a head to indicate that transaction begins here, a length to record its data size, and a tail to indicate the end of it (see Figure 2). When a WAL file is created, RocksFS preallocates a 4MB disk space for it. Every WAL write will directly apply to the free space orderly until the residual space is not big enough for current transaction, in which case a new 4MB space is allocated for the WAL file. Through this kind of new format, the metadata update frequency of WAL files can be reduced vastly especially for small insert- intensive workload, since the metadata only need to be updated when the WAL file size changes.

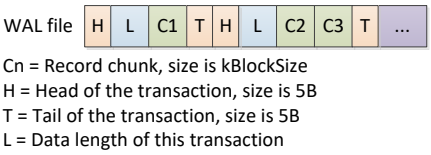

Figure 1: New WAL file format in RocksFS

Besides, RocksFS only supports operations that are necessary to RocksDB to lower filesystem overhead. We compare RocksFS with Ext4 and BlueFS in the environment of RocksDB on a HDD, the write performance of storage system based on RocksFS are at most 50% and 390% higher than it on BlueFS and Ext4, respectively.

In conclusion, RocksFS can drastically improve the small write performance of RocksDB. The data layout and I/O patterns of RocksFS are highly optimized for RocksDB running on HDD device. We hope that these optimization techniques used to reduce the log overhead in RocksFS will inspire the future generation of high-performance key-value stores, especially for LSM-tree based key-value stores. In future works, we will research the performance of RocksFS in SSD, and make effort to improve the implementation of RocksFS.

Keywords: key-value store, RocksDB, log overhead, simplified filesystem, write performance.

References

[1] Facebook, "Rocksdb," https://github.com/facebook/rocksdb, 2013, [Online; accessed 1-Dec-2016].
[2] S. Weil, "Ceph," http://ceph.com/, 2007, [Online; accessed 17-Nov-2016].