MV-FTL: An FTL That Provides Page-Level Multi-Version Management

Doogie Lee, Mincheol Shin, Wongi Choi, Hongchan Roh, and Sanghyun Park, *Member, IEEE*

Abstract—In this paper, we propose MV-FTL, a multi-version flash transition layer (FTL) that provides page-level multi-version management. By extending a unique characteristic of solid-state drives (SSDs), the out-of-place (OoP) update to multi-version management, MV-FTL can both guarantee atomic page updates from each transaction and provide concurrency without requiring redundant log data writes as well. For evaluation, we first modified SQLite, a lightweight database management system (DBMS), to cooperate with MV-FTL. Owing to the architectural simplicity of SQLite, we clearly show that MV-FTL improves both the performance and the concurrency aspects of the system. In addition, to prove the effectiveness in a full-fledged enterprise-level DBMS, we modified MyRocks, a MySQL variant by Facebook, to use our new Patch Compaction algorithm, which deeply relies on MV-FTL. The TPC-C and LinkBench benchmark tests demonstrated that MV-FTL reduces the overall amount of writes, implying that MV-FTL can be effective in such DBMSs.

Index Terms—MVCC, FTL, flash translation layer, SSD, concurrency control

1 Introduction

Solid-state drives (SSDs) are being increasingly adopted in the DBMS research literature [1], [2], [3], [4], [5], [6]. In particular, there exists a unique and interesting characteristic of SSDs called the out-of-place (OoP) update. Because of the inability to overwrite in NAND-flash-memory, SSDs' main storage media, SSDs process updates in an append-only manner [7]; instead of overwriting existing data, SSDs redirect the updated data to an empty page and move the logical-to-physical (L2P) mappings onto them. When there is no free space to append new pages, an SSD triggers a garbage-collection mechanism that reclaims the space occupied by old and unnecessary page versions.

The way an SSD handles an update is quite familiar in the DBMS literature—in practice, the approach can easily be found on many DBMSs that implement multi-version concurrency control (MVCC)[8], [9], [10]. In MVCC, a DBMS handles updates in an append-only manner; that is, instead of replacing the old version with the new one, DBMS simply appends the new one, keeping old ones intact. Consequently, a DBMS stacks multiple versions of each record and serves concurrent accesses to a record in more flexible ways by using those multiple versions. Moreover, a DBMS also has garbage-collection mechanisms to reclaim the space occupied by expired versions.

However, being unaware of the SSD, a DBMS implements MVCC in its own way, without utilizing the OoP update characteristic of SSDs. In other words, a DBMS implements the append-only updates by itself, without knowing an SSD already handles updates in an append-only manner. Moreover, a DBMS executes garbage-collection operations to secure free space, which may overlap with SSD garbage collection.

From this observation, we propose MV-FTL, a multi-version flash translation layer (FTL), which is the core software layer of an SSD. The proposed solution explicitly manages multiple page versions and serves the multiple versions to a DBMS. More specifically, MV-FTL gathers the updated pages from each transaction into a new data structure called *diffL2P*, a logical data structure that represents a *version*. MV-FTL serves these diffL2Ps to transactions in an atomic fashion; that is, a transaction can see either all the pages in a diffL2P, or none of them. In addition to this atomic update, MV-FTL manages the commit orders between diffL2Ps, enabling the snapshot isolation [11] between updates from transactions.

Consequently, MV-FTL relieves the burdens of a DBMS to implement MVCC.[1] By cleverly exploiting the OoP update characteristic, MV-FTL provides the host the ability to manage multiple page versions without explicitly writing any additional log data. In addition, MV-FTL garbage-collects unnecessary page versions gracefully by extending the SSD's garbage-collection scheme, which will significantly reduce the MVCC garbage-collection overhead.

- *D. Lee is with Department of Computer Science, Yonsei University, Seoul 03722, Korea. E-mail: edoogie@gmail.com.*
- *M. Shin, W. Choi, and S. Park are with the Department of Computer Science, Yonsei University, Seoul 03722, Korea. E-mail: {smanioso, cwk1412, sanghyun}@yonsei.ac.kr.*
- *H. Roh is with SK Telecom, Seoul 04539, Korea. E-mail: fallsmal@gmail.com.*

Manuscript received 5 Sept. 2016; revised 4 Aug. 2017; accepted 14 Sept. 2017. Date of publication 27 Sept. 2017; date of current version 5 Dec. 2017. (Corresponding author: Hongchan Roh and Sanghyun Park.)
Recommended for acceptance by C. Ordonez.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2757016

1. MV-FTL will not fully replace MVCC, because MV-FTL can manage versions only in NAND page units, which is too coarse-grained to be used in enterprise-level DBMSs, where versions are managed in tuple units. Nevertheless, MV-FTL can still replace the MVCC of MyRocks or MongoRocks, which will be discussed later.