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.

To show the performance and concurrency implications of MV-FTL, we first modified SQLite [12], one of the most popular DBMSs for mobile applications, and conducted several experiments using well-known benchmarks. In addition, to demonstrate the usefulness of MV-FTL in more sophisticated DBMSs, we modified MyRocks [13], a MySQL variant that is widely adopted by Facebook. Then, by running several benchmark tests on an MV-FTL emulator, we confirmed the potential benefits of MV-FTL on MyRocks.

In summary, the contributions of this paper are as follows:

- We propose a novel MV-FTL that provides page-level multi-version management to the host. MV-FTL

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.

Manuscript received September 5, 2016.
Corresponding author: Sanghyun Park (email: sanghyun@yonsei.ac.kr).