Database software for non-volatile byte-addressable memory

Goetz Graefe, Harumi Kuno

HP Laboratories
HPL-2011-37

Keyword(s):

database, NVRAM, byte-addressable, memory, concurrency, recovery, 8-trees

Abstract:

Most transactional software, e.g., in database systems, is written for a 2-level memory hierarchy with volatile RAM and persistent disk storage. For example, the standard "write-ahead logging" technique relies on an in-memory buffer pool to hold back dirty data pages until the relevant log records have been written to stable storage. It is well-known that a buffer pool enables fast access via locality of reference. In addition, in transactional systems, a buffer pool seems an essential component of write-ahead logging, which ensures the consistency of persistent data even in the face of recovery from media failure. For a transactional storage system using non-volatile, byte-addressable memory, which does not suffer from the slow access times of disk, a buffer pool seems unnecessary. However, removing the buffer pool seems to complicate transactional updates. We introduce a design and supporting techniques that simplify the implementation of transactions using NVRAM as persistent storage. One new technique, "log-shipping to self," avoids use of a buffer pool, supports existing write-ahead logging protocols, and is simpler than traditional implementations of write-ahead logging. We also present a second innovation - a software frame-work for automatically detecting and repairing individual pages. This feature is particularly relevant in the context of NVRAM, since some NVRAM technologies may incur single-page failures. Traditional database techniques know only offline (or snapshot) consistency checks; online and incremental verification of data structures does not verify all invariants. Traditional database techniques also know only media recovery, i.e., recovery of an entire disk rather than of individual pages. Repair of individual pages today is more art than science and engineering. Techniques for complete online consistency checks as well as principled automatic repair are also introduced here.

External Posting Date: March 21, 2011 [Fulltext] Approved for External Publication
Internal Posting Date: March 21, 2011 [Fulltext]
To be published and presented at Non-Volatile Memories Workshop 2011, March 6-8, 2011.

Database software for non-volatile byte-addressable memory[1]

Goetz Graefe, Harumi Kuno

Hewlett-Packard Laboratories

Abstract: Most transactional software, e.g., in database systems, is written for a 2-level memory hierarchy with volatile RAM and persistent disk storage. For example, the standard "write-ahead logging" technique relies on an in-memory buffer pool to hold back dirty data pages until the relevant log records have been written to stable storage. It is well-known that a buffer pool enables fast access via locality of reference. In addition, in transactional systems, a buffer pool seems an essential component of write-ahead logging, which ensures the consistency of persistent data even in the face of recovery from media failure.

For a transactional storage system using non-volatile, byte-addressable memory, which does not suffer from the slow access times of disk, a buffer pool seems unnecessary. However, removing the buffer pool seems to complicate transactional updates.

We introduce a design and supporting techniques that simplify the implementation of transactions using NVRAM as persistent storage. One new technique, "log-shipping to self," avoids use of a buffer pool, supports existing write-ahead logging protocols, and is simpler than traditional implementations of write-ahead logging.

We also present a second innovation – a software framework for automatically detecting and repairing individual pages. This feature is particularly relevant in the context of NVRAM, since some NVRAM technologies may incur single-page failures. Traditional database techniques know only offline (or snapshot) consistency checks; online and incremental verification of data structures does not verify all invariants. Traditional database techniques also know only media recovery, i.e., recovery of an entire disk rather than of individual pages. Repair of individual pages today is more art than science and engineering. Techniques for complete online consistency checks as well as principled automatic repair are also introduced here.

1 Introduction

A relational database software system can be split into multiple pieces. For example, the relational layer provides catalogs and interactive data definition, parsing and validation including security, and query processing including compile-time optimization and run-time execution. The storage layer provides access methods (e.g., B-tree indexes) including search and updates, transactions including concurrency control and recovery, and utilities including defragmentation, consistency checks, backup and restore. The present paper concerns only some aspects of the storage layer, even if the relational layer also could be optimized for entire databases fitting into byte-addressable memory.

Even if hardware and low-level system software provide some of the "enterprise abilities" (reliability, scalability, manageability, etc.), each database system must verify its data structures and repair or recover them as appropriate. For example, if database replication fails to propagate all aspects of a node split in a B-tree, some keys or pointers may be incorrect, which in turn may lead to incorrect query results as well as subsequent incorrect updates. All commercial database packages include verification utilities, and all vendors recommend their regular use.

Current software for disk-based databases represents a balance between functionality, execution efficiency, and software complexity (development, testing, maintenance, enhancement). Introduction of a new type of storage medium upsets that existing balance. New features or optimizations for reliability or performance might therefore render data management software more complex. Our goal is to provide carefully chosen innovations that make the overall software system less complex, more principled, and, ideally, also more efficient.

For example, we propose to continue dividing a database into pages, even in byte-addressable NVRAM. There are several good reasons for doing so. During insertion, update, or deletion of a record, only a small part of the overall data structure is subject to change. This has immediate beneficial effects on complexity, concurrency control and recovery. Pages provide containment for faults and repair, and there also are subsequent benefits, e.g., for efficient incremental backup to page-based storage such as a traditional disk. Page sizes require new optimization, however, since heuristics such as for optimizing disk-based B-trees do not apply.

2 Related techniques

In Gray's early form of write-ahead logging in databases [G 78], each type of database update must be implemented in three forms: as "do" action during normal processing including creation of log records, as "redo" action invoked during media or system recovery, and as "undo" action invoked during transaction rollback or system recovery. Since a failure (system crash) is at least as likely during recovery as during forward processing, and since recovery actions are not logged, recovery actions must be idempotent, i.e., they may be applied repeatedly with the same outcome.

ARIES logs recovery actions and tags each data page with the address of the most recent log record already applied, called the log sequence number (Page LSN) [MHL 92]. "Exactly once" application of log records can be ensured, idempotent recovery actions are not required, and thus "undo" can be a logical compensation rather than a strict physical reversal of the original action.

When a database writes multiple disk sectors as a single large page, partial writes are possible. Multiple research and development efforts have focused on verification of individual database pages and of complete databases, e.g., the structure of B-trees and the consistency of tables, indexes, and views [GS 09, M 95]. Our work builds upon those prior efforts and designs.

3 Transactional updates

Write-ahead logging prohibits modifications in the persistent database prior to successful logging. In traditional database architecture, a buffer pool holds intermediate states. Dirty pages remain in the buffer pool until relevant log records are on stable storage.

In byte-addressable persistent memory, a buffer pool with copies of stored pages is neither required nor desired. Nonetheless, it is imperative that uncommitted updates can be rolled back if a

[1] *Presented at the 2nd Annual Non-Volatile Memories Workshop (NVMW) 2011. UC San Diego, March 6-8, 2011.*

transaction fails. A naïve approach to the problem therefore requires additional recovery logic, code, complexity, and test cases.

In our technique, a log record is formed first and logged immediately in persistent storage (NVRAM), whereupon it can be applied to the database. This adheres to write-ahead logging – the log record is saved before any in-place database update. The traditional "do" action is reduced to (or replaced by) creation of a log record. Thus, the traditional "do" action of "do, redo, undo" is obsolete; only "redo" and "undo" remain for database updates.

The actual database modification is driven by log records, not by the code executing the user transaction. If desired, the threads that apply log records may be separate from those that execute user transactions and create log records, in particular on many-core processors. On the other hand, if there is a delay, a new data structure must keep track of pending updates to each data page, such that subsequent retrievals from such pages can produce the most up-to-date data. In order to avoid searching something like a buffer pool, this data structure could be anchored in the data page itself. A traditional lock manager cannot serve this purpose because locks disappear when an update transaction commits.

The new technique promises to simplify transaction implementation and it may be more suitable to many-core processors than today's standard techniques. Due to the similarity of database replication by log shipping, we call the new sequence of update actions "log shipping to self." A new data structure is needed to ensure timely application of log records in order to prevent queries retrieving incorrect results.

4 Fault detection and recovery

Database pages may become corrupt for a number of reasons [M 95], from hardware faults to software problems in update logic, recovery, or replication. Offline consistency checks require a quiescent database, which is inconvenient at best, or a snapshot, which usually is out-of-date by the time the consistency check is complete. Traditional online consistency checks usually are incomplete. For example, they may ensure integrity constraints within a page but not across pages. In a B-tree index, space allocation for variable-size records and sort order of key values may be verified within a page but not non-overlap of key ranges in neighboring leaf pages. Fortunately, with minor modifications to the traditional B-tree page format, online verification can be complete, including all relationships to neighboring pages [GS 09]. Each root-to-leaf traversal can verify all integrity constraints of all pages it touches. Depending on the reliability of various technologies and successive generations of non-volatile storage memory, such online verification may become a crucial defensive technique.

Detection of faults has little value, however, if detected faults cannot be repaired. Traditional techniques enable transaction recovery (rollback or durability of a single transaction), system recovery (clean-up after a process or system crash), and media recovery (re-creation of a failed disk or disk array). What is needed, however, is a fourth form of database failure and recovery – repair of one or more individual database pages.

Current single-page "repair" mechanisms seem more "black art" than solid science and engineering. They use heuristics such as "most likely change" or "minimal edit distance" rather than a principled approach. If the heuristics fail or their change seems too radical, database administrators today are asked to erase and recreate information, e.g., drop secondary indexes and rebuilt them from the primary index, which are deemed reliable and accurate. Efficient query and update processing can resume when the secondary indexes are complete, i.e., after considerable delay.

Our proposed technique introduces single-page backups and online single-page recovery using the transaction log. The backup pages may be in the database or in the recovery log. In the database, they may be the result of a page movement during defragmentation, page relocation in a write-optimized (log-structured) operation [G 04], or an explicit backup operation. In the recovery log, backup pages may be the result of formatting after allocation or of explicitly logging the current page contents. In addition to page backups, our proposed technique requires that log records pertaining to the same data page be linked for efficient scan within the recovery log. Many database systems already link log records in this way. Ordinarily, the anchor of this linked list is in the data page, i.e., the Page LSN. However, if a data page is corrupt in non-volatile memory, all of its contents, including its Page LSN, are suspect. Therefore, a second copy of the Page LSN is required.

In systems without buffer pool, the queue of waiting log records ("log shipping to self") can serve the purpose. If page locks are employed, this queue can be integrated with the lock manager. When an update transaction commits without applying its log records, a "no lock" entry must remain pointing to the pending log records such that subsequent retrievals from that data page can first apply those log records. Once a data page is up-to-date, its parent page (e.g., in a B-tree index) can retain the second copy of the Page LSN. This is equivalent to tracking the current location of pages in write-optimized (log-structured) B-trees.

It is not required to keep a pointer to the most recent backup page if page backups are logged. In that case, traversing the chain of log records into the past will eventually find a backup page.

5 Summary

In summary, software that supports transactional retrievals and updates requires modifications when NVRAM replaces RAM and perhaps even disks. These modifications need not increase complexity; in fact, new techniques may improve both complexity and performance. For updates, we reduced the traditional "do, redo, undo" paradigm of transactional updates to "redo and undo" only. For repair of individual pages, we described data structures for complete online verification of B-tree (database index) structures and we introduced techniques for a principled repair based on page backups and the recovery log.

6 References

[G 78] Jim Gray: Notes on data base operating systems. Advanced course: operating systems 1978: 393-481.

G 04] Goetz Graefe: Write-optimized B-trees. VLDB 2004: 672-683.

[GS 09] Goetz Graefe, R. Stonecipher: Efficient verification of b-tree integrity. BTW 2009: 27-46.

[M 95] C. Mohan: Disk read-write optimizations and data integrity in transaction systems using write-ahead logging. ICDE 1995: 324-331.

[MHL 92] C. Mohan, Donald J. Haderle, Bruce G. Lindsay, Hamid Pirahesh, Peter M. Schwarz: ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM TODS 17(1): 94-162 (1992)