This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3004518, IEEE Transactions on Knowledge and Data Engineering

Embedded Transaction Support inside SSD with Small-Capacity Non-volatile Disk Cache

Yanjie Tan, Huailiang Tan*, Peng Zhu, Youyou Lu, and Zaihong He

Abstract—Flash-based Solid State Drives (SSDs) have proved to be ideal devices that support embedded transaction protocols inside SSDs. Existing embedded transaction protocols in SSDs effectively improve transaction throughput, but still incur high transaction overhead and long recovery time. While it is reasonable to provide a small-capacity non-volatile (NVM-based) disk cache in the SSDs, in this paper, we propose a new embedded transaction protocol called Non-volatile Cache Transaction (NVCTX). NVCTX reduces transaction overhead and provides fast recovery by leveraging the small-capacity NVM-based disk cache from two aspects. First, we store transactional metadata, which is of small amount but is frequently accessed, in the NVM-based disk cache rather than in the flash memory. Second, we introduce two techniques, i.e., a dynamic allocation algorithm and a hybrid storing method, to improve the performance when the capacity of the NVM-based disk cache is very limited. We have implemented NVCTX on a real hardware board called Cosmos+ FPGA platform, and modified ext4 file system and NVMe (Non-Volatile Memory express) driver to be compatible with the transactional interfaces provided by NVCTX. For comparison, we also implement SCC, BPCC, WAL, and X-FTL protocols in the firmware of Cosmos+ FPGA platform. Evaluations using DBMS (Database Management System) and file system workloads show that, compared to four typical transaction protocols (SCC, BPCC, WAL, and X-FTL), NVCTX improves transaction throughput by up to 136.5%, 9.4%, 131.6% and 29.9%, reduces write traffic to flash memory by up to 42.8%, 4.1%, 62.4%, 31.2%, lowers garbage collection overhead by up to 93.2%, 63%, 66.5%, 22.1%, and shortens recovery time to 1/2574, 1/2559, 1/95 and 1/2 respectively compared with SCC, BPCC, WAL, and X-FTL.

Index Terms—Embedded Transaction Protocol, NVM-based Disk Cache, Small-capacity, Solid State Drive, Throughput, Transaction Recovery.

1 Introduction

For decades, transaction [11] has been widely utilized in database management systems (DBMSs), file systems, and high-level applications, which belong to the software layer. Transaction recovery, as a vital part of the transaction mechanism, ensures atomicity and durability of data updates to the system. Write ahead log (WAL) [23] and shadow paging [10] are two common transaction recovery methods. WAL is used in file system (e.g., ext4 [22]) and DBMS (e.g., PostgreSQL [31]) to ensure system consistency by the means of logging, which requires writing data to a log area before the data is written back to the storage area. Checkpoint operation is triggered when free pages of the log area are less than the threshold value. WAL could undo and redo transaction operations by utilizing the data of the log area after a power failure to ensure faster recovery. Unfortunately, this frequently triggered check point greatly degrades system performance. Besides, in order to record the log records, the data needs to be written twice to the storage device, which results in a write amplification. In comparison, shadow paging uses a strategy called out-of-place updating to handle data updates. It allocates a shadow page that is different from the original page which needs to be updated. But shadow paging induces random writes which lower I/O performance since the data is required to be written to its new location followed by the pointer update. In addition, DBMS and file systems [26], [37], [28] implement transaction respectively, which increases redundancy and the ratio of data error in overall system. To solve the problem, some researchers try to use WAL and log [12], [21] to implement transactional protocols within the hard disk. But this leads to one side effect, i.e., the linear address space of hard disks becomes more fragmented, which weakens the I/O performance of storage system. In the traditional disk controller, read delay caused by the debris problems has become a major obstacle to the realization of the transaction mechanism [29].

Fortunately, the emergence of flash memory can reduce these performance barriers and better support transaction [25] in the device than in the software layer. Flash memory with its fast random read can amortize the read delay caused by the debris [29], flash out-of-place mode [29], [40], [4], [20] can save old and new versions data simultaneously. At the same time, byte-addressable non-volatile memory (NVM) is also being introduced into the SSD in the form of disk cache. Due to the difference between the SSD and the traditional disk itself, its internal details need to be shielded by FTL (Flash Translation Layer [13]), so that it can provide a unified disk interface, which makes flash memory provide better transaction support than the software layer. In recent years, the researchers have proposed some internal embedded transaction commit protocols based on SSD, such as WAL in SSD, TxFlash [29], Flag Commit [27], KAML [14] etc. The implementation of WAL in SSD can effectively solve the problem of debris, but write amplification is still inevitable, which will shorten the SSD lifetime because of the limited erasure number. A new ring commit protocol derived from shadow paging is proposed in TxFlash which is divided into Simple Cycle Commit (SCC) and Back Pointer Cyclic Commit (BPCC) protocol through recording the pointer out-of-band (OOB). All the pages of the same transaction are linked into a ring structure. When the system recovers, they verify the transaction submission by the state of the loop, which can effectively reduce the system failure and lower redundancy, and don't require battery backed memory. Flag Commit exploits the partial page programming characteristic of Single-Level Cell (SLC) to keep track of the transaction status, which decreases the cost of TxFlash garbage collection and improves the performance. KAML proposes a key-addressable multi-log approach which is finer-grained to support atomic transactions. It also designs a caching layer to utilize host DRAM (Dynamic Random Access Memory) to afford more transactional features and enhance performance. However, it does not consider the transaction recovery situation.

* *Yanjie Tan, Huailiang Tan, Peng Zhu, and Zaihong He are with College of Information Science and Engineering, Hunan University, Changsha, 410082, China. E-mail: tanyanjiecws@hnu.edu.cn, tanhuailiang@hnu.edu.cn, zhupeng1011@163.com, hezaihong@hnu.edu.cn.*

* *Youyou Lu is with Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China. E-mail: luyy09@mails.tsinghua.edu.cn.*

* *Huailiang Tan is the corresponding author. E-mail: tanhuailiang@hnu.edu.cn.*

With the explosive growth of data, the amount of data has become a major obstacle to rapid recovery of transaction failures in flash memory data storage systems [41], [9]. Under this situation, it is necessary that the transaction protocol should guarantee that the recovery time could be very short or even zero [23], [5]. Lahiri et al. [17] pointed out that it is important to have a low recovery time from any failures which do occur so that it can minimize the unplanned downtime. Son et al. [32] also indicated that the reduction of recovery time is critical because user transactions cannot be dealt with by database systems during the recovery process. Especially for replicated database systems [1], [2], with the recovery time raising, the utilization of all nodes will be lower. But most embedded transaction protocols have high recovery time after a power failure. Both TxFlash and Flag Commit need to scan the entire SSD mapping information to play back the transactions during system recovery. Especially in distributed storage, the recovery time grows linearly with SSD capacity so that it is difficult to meet the high reliability requirements. For example, when the SSD capacity is 1TB (Trillionbyte), TxFlash only take few seconds to recovery. However, if the capacity grows to 1EB (Exabyte), it perhaps needs a dozen days to scan the entire SSD to reconstruct the mapping table. So, a new embedded transaction protocol which can effectively shorten the recovery time to satisfy the modern distributed storage system is urgently required.

In the case of high concurrency and high abort rate (such as Taobao and other business platform, smart grid system, some security monitoring systems), the performance of transactions is greatly affected by the abort rate. For example, in YCSB load, which uses OCC (optimistic concurrency control) to ensure data serialization, with 10 writes per transaction, the transaction abort rate is up to 98% [36]. Actually, there is always one or more hot spots that need to handle high conflict accesses in numerous transactional applications [36]. Such a high abort rate can completely suffocate the system performance. In the SSD embedded transaction commit protocol, TxFlash needs to maintain commit cyclic link list for system data consistency. In order to form the list, SCC and BPCC need to cache the shadow page until the next transaction shadow page arrives. Especially when the abort ratio is high, SCC needs to frequently erase uncommitted pages before writing new-version, and BPCC requires a complicated garbage collection mechanism that reclaims straddle responsibility set (SRS) of a page before the page is reclaimed [29], which leads to lower system performance and higher write delay. Flag Commit reduces the cost of the pointer by covering the way of writing the page metadata based on the defects of the TxFlash. However, for covering the page data, it requires the flash page must be able to be partially programmed which is only available for SLC flash chips, so it cannot be applied to Multi-Level Cell (MLC) flash chips.

In this paper, we propose a new embedded transaction protocol inside SSDs, namely Non-volatile Cache Transaction (NVCTX), effectively exploiting the small-capacity NVM-based disk cache for decreasing transaction overhead, achieving fast recovery and enhancing transaction performance. NVCTX caches running transactions and a part of mapping table in the NVM-based disk cache. It is easy to identify whether a page is valid by running transactions' status and persisted mapping table, which only needs to distinguish recently written pages. Thus, NVCTX significantly lessens write amplification by avoiding writing the log area frequently. By checking the status, the garbage collection mechanism in NVCTX only needs to reclaim the invalid pages rather than immediately erasing the relevant pages, when the transaction is interrupted. So NVCTX does not have complicated garbage collection mechanism and achieves a stable performance, even in high abort rate situations. Meanwhile, NVCTX needs not to scan the whole SSD for distinguishing between aborted pages and committed pages, so the recovery time is greatly shortened. Finally, because NVCTX does not use partial page programming, it not only is suitable for SLC flash memory, but also can be used in the MLC flash memory which is easily available in the market.

Our main contributions are summarized as follows.

(1) We propose an embedded transaction protocol design inside SSD, called NVCTX, which takes full advantage of the small-capacity NVM-based disk cache to efficiently keep the transactional metadata persistent rather than flushing to flash memory frequently. It can effectively decrease the write amplification and prolong the lifetime of SSD.

(2) We extend the FTL interface in flash memory to support the atomicity and consistency of transaction. For accelerating transaction recovery, we construct the NVM Mapping Table and In-Process Table from NVM-Cache (non-volatile cache) to store the transactional metadata, which is of small amount but is frequently accessed. We also introduce a dynamic allocation algorithm and a hybrid storing method, to improve the cache efficiency of the NVM-based disk cache with small capacity.

(3) We guarantee NVCTX still maintains superior performance with high abort rates, by tracking and judging the state of running transactions.

(4) We implement NVCTX on a real hardware board called Cosmos+ FPGA platform, and modified ext4 file system and NVMe (Non-Volatile Memory express) driver to be compatible with the transactional interfaces provided by NVCTX. We also implement SCC, BPCC, WAL, and X-FTL protocols in the firmware of Cosmos+ FPGA platform for comparison. Our experimental results demonstrate that NVCTX has longer lifetime, shorter recovery time and better performance than previous embedded transaction protocols.

The rest of this paper is organized as follows. Section 2 introduces related work. Section 3 illustrates detailed design of NVCTX components and describes the embedded transaction protocol. Section 4 presents implementation details. We present evaluation results in Section 5. Finally, in Section 6, we summarize our results and provide a number of conclusions.

2 Related Work

In this section, we review related research in transaction support in flash memory with small capacity non-volatile cache.

Flash-based NVM. Some works have leveraged non-volatile memory to ensure system consistency and speed up system recovery. WSP [26] reduced recovery time from back end storage, and eliminates runtime overheads by using flush on fail technique. It integrated DRAM, ultra-capacitors, and NAND flash into a single module NVDIMM. The host system only flushes the transient state to NVDIMM over memory bus and signals the start of a save operation. NAND flash is served as the back end device and stores persisted data when a power failure occurs. Thus, DRAM content and transient state are all preserved to NAND flash using ultra-capacitor power when the host loses power. WSP provided the illusion of a non-volatile memory to the host system, which makes sure DRAM content and transient state will not be lost, and only focused on the consistency of file system, which has no transaction mechanism. FTL2 [37] solved the endurance problem and performance degradation caused by partial page updates in flash memory. It designed the Content Cache to cache partial page updates, which reduces write amplification and improves I/O performance. In order to make sure that data cached in the Content Cache is not lost, FTL2 needs to make sure the Content Cache be a non-volatile memory unit. So FTL2 is proposed to implement in battery-backed DRAM or byte-addressable memories

such as memristor and phase change memory (PCM). FTL2 writes partial page to NVM to reduce write amplification, but it needs read data from flash memory to get partial data. These works all need to use high-capacity non-volatile memory as data cache or persistent backup, our NVCTX manages non-volatile memory with small capacity as transactional metadata storage to support embedded transaction commit protocol.

Flash Transaction. Flash memory has the characteristics of out-place-update and fast random read, which is particularly ideal for supporting embedded transaction protocols. Atomic-Write [28] leveraged the no-overwrite property and log structured FTL to support embedded transaction in flash-based SSD, and tracked the state of a transaction using the log. Atomic-Write set first page flag of a transaction to 0, and sequentially appended with 0 of the same transaction. It revised the flag of last page to 1 when transaction commits. Atomic-Write cannot perform transaction concurrence because of strict isolation. NVCTX does not have strict isolation constraint, and can support transaction concurrency. LightTx [19] provided a zone-based transaction state tracking scheme to track recent updates, and reduced the cost of transaction state tracking by retiring dead transactions periodically. Meanwhile, LightTx also supported arbitrary transaction concurrency. But it needs to flush frequently mapping pages to flash memory for small transactions, which takes high overhead to track long transaction status. NVCTX considers long transaction workload carefully. Mobius [30] provided different type of transactional primitives to support static and dynamic transactions. It wrote mapping information and transaction information as atom inode to Atom Log Area (ALA) before starting the transaction for static transaction. Nevertheless, for a dynamic transaction, atom inode with mapping information and transaction information is written to ALA when the dynamic transaction is committed. For small transactions, Mobius needs to frequently write atom inode to ALA, thus write latency is increased. However, these embedded transaction protocols ignore non-volatile cache, which could make transaction design more effective and simpler. MixSL [6] combined shadow paging and logging technique by tracking transaction update operations using shadow paging and trailing transactions' status utilizing logging. When a transaction commits, MixSL inserts transaction identifier and transaction status to log buffer, and then flushes the log buffer page to fixed flash area. Thus MixSL frequently flushes log buffer page to flash memory for small transaction, and leads to increasing write latency. TxCache [20] is also an embedded transactional protocol inside SSD, which persisted new-version data in non-volatile disk cache using shadow page way. However, TxCache needs high-capacity non-volatile disk cache. For example, for 1TB SSD, TxCache needs at least 32MB NVM-based disk cache. That will lead to larger volume, high energy consumption, and high cost. NVCTX only leverages very little capacity (i.e., 1MB) of NVM-based disk cache to ensure system consistency. X-FTL [15] supports atomic propagation of pages to SSD device for SQLite databases, and is implemented on Jasmine OpenSSD platform based on the Barefoot controller. But the X-L2P table in X-FTL have to be written to a new location in flash with every transaction committing, which causes high performance overhead and write amplification. Unlike previous researches which implement a transactional feature for SSDs by extending the host interface, Jin-Young Choi et al. [3] presents an FTL framework called HIL for constructing FTLs to provide a conventional block device interface, and proves the correctness of HIL for the crash recovery. But the HIL framework just focuses the crash recovery of FTLs.

3 Design

In this section, we describe the NVCTX design, which leverages the small-capacity NVM-based disk cache to efficiently support embedded transaction protocol inside SSDs.

Design Analysis. The implementation of transaction mechanism is divided into two parts: concurrency control and downtime recovery. Database systems and file systems are implemented separately for concurrency control and downtime recovery. In database systems, the concurrency control is done by locking operation, and the isolation level of the transaction can be set according to the requirements. Taking into account the out-of-place feature of SSD, only the write request may update the mapping relations in the FTL, and the read request of upper layer will not write new data, so the cost of reading request is not compared and read concurrency are implemented in the upper layer, just like TxFlash did [29]. In this paper, we make full use of the characteristics of out-of-place for flash and propose an optimized strategy for the data management of page writing request, the detail will be discussed in Section 3.4. For ensuring the downtime recovery of system effectively, NVCTX must guarantee that the page write request is an atomic operation. Thus, the transaction writing request is divided into multiple page writing requests. If the transaction is committed, the mapping relationship in In-Process mapping is flushed to the NVM Mapping Table. Otherwise, once the transaction is aborted, its cache page data in In-Process Table is invalidated until the garbage collection mechanism reclaims it.

Load Analysis. To take full advantage of the small-capacity NVM-based disk cache, we collect the transactional traces by using fileserver, varmail and webproxy workloads from file benchmark [7] to collect file system traces and using the TPC-C benchmark DBT2 [38], [18] on PostgreSQL to collect database workload traces. Then we observe the distribution of transaction size (the number of I/O requests per transaction), and analyze the distribution result after testing these four real workloads (fileserver, varmail, webproxy, and database workloads TPC-C), as shown in Fig. 1. X-axis represents transaction size, and Y-axis denotes the cumulative distribution of the number of requests in each transaction. From the figure, we can obtain two observations. First, to a certain workload, all transactions are either short (the transaction size is less than 500) or long (the transaction size is greater than or equal to 500). For example, all transactions are short for varmail and TPC-C workloads, their transactions contain the number of requests from 0 to 300. On the contrary, for webproxy workload, 95% transaction size is between 800 and 2500, 80% transaction size is between 5000 and 7000 for fileserver workload. Second, the change of transaction size is smooth and does not have distinct fluctuation. Therefore, two conclusions are summarized as follows. First, because the number of requests of running transactions does not tremendously change, the metadata size of running transactions also does not sharply vary. Second, to reduce write latency and improve system performance, non-persistent mapping pages are not frequently flushed to flash memory.

With the two observations, NVCTX could be efficient by only caching running transactions and a part of mapping table in the NVM-based disk cache. To effectively leverage the small-capacity NVM-based disk cache, a dynamic allocation algorithm is designed. To support workloads with long transactions (e.g., fileserver), we design a hybrid storing method in NVCTX to improve system performance and prolong SSD lifetime.

3.1 NVCTX Overview

While emerging NVM technologies have gained significant development in recent years [8], a part of volatile DRAM used as the disk cache is provided with energy storing capacitors (similar with NVM) to prevent data loss in SSDs. In the case of a power failure, the NVM-based disk cache offers an opportunity to complete commands and guarantees fixed size data in the temporary buffer to be committed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3004518, IEEE Transactions on Knowledge and Data Engineering

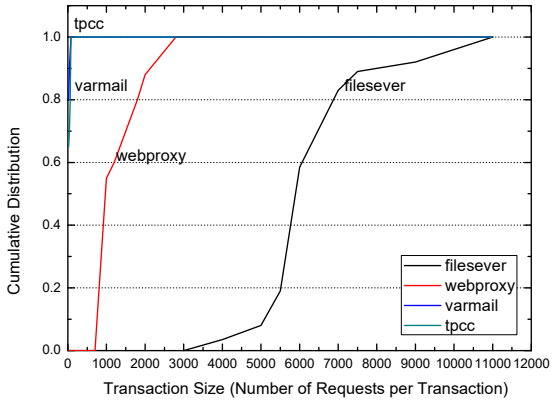


Fig. 1. Transaction size distribution.

TABLE 1
NVCTX interface

Operation	Description
WRITE(LBA, Len)	Normal write data to flash in the LBA(logic block address).
READ(LBA, Len)	Read data from LBA.
TWRITE(TXID, LBA, Len)	Write data to the transaction TXID in the LBA.
BEGIN(TXID)	Check the availability of TXID and start transaction whose identifier is TXID.
COMMIT(TXID)	Commit the transaction whose identifier is TXID.
ABORT(TXID)	Abort the transaction whose identifier is TXID.

to the non-volatile flash media. The constraint of NVM-based disk cache is the limited capacity that energy storing capacitors inside SSDs can only provide limited supplemental power for the buffer to be flushed. Thus, using a tiny super-capacitor to supply power for a smaller part of DRAM and make it non-volatile is a better cost-benefit way. This is also help to reduce the power consumption and enable further size reduction of SSDs. For this reason, NVCTX is designed by exploiting as small capacity non-volatile memory as possible to ensure cost-effective.

NVCTX utilizes the no-overwrite property of flash memory for maintaining new-version and old-version data at the same time. It tracks the running transactions' status in the event of power failure by using NVM-based disk cache, and only stores the mapping metadata of running transactions' requests and the status of activate transactions due to the very limited capacity. During system recovery, NVCTX can redo or undo a transaction according to transaction's status. To cut down the time of reconstructing FTL mapping table, NVCTX flushes the mapping pages which will be replaced to flash memory. So NVCTX does not need to scan physical pages of whole flash memory during recovery.

Architecture. As TxFlash [29] demonstrates, SSDs are particularly ideal for supporting transactions because of its copy-on-write (Cow) nature of SSDs, fast random reads, high concurrency, and new interface specifications. To support embedded transaction protocol inside SSD, the device interface and FTL in the SSD are extended. The NVCTX architecture is shown in Fig. 2. In the figure, in addition to the common modules (i.e., FTL Mapping Table, Garbage Collection, Wear Leveling, Free Block List, and Dirty Page List), NVCTX introduces four components (In-Process Table, NVM Mapping Table, Commit Logic, and Recovery Logic). The Commit Logic module extracts the transaction information from the extended transactional interface (as shown in Table 1) and tracks in-progress transactions using the In-Process Table. The In-Process Table records in-processing transaction identifier (TxId), transaction status, and the mappings of transactions' requests. The NVM Mapping Table stores the mappings of latest committed transactions and non-persistent mapping pages. The Recovery Logic distinguishes the committed transactions from the uncommitted transactions during system recovery, then redoes the committed transaction requests or undoes the uncommitted transaction requests.

Core Components. To complete garbage collection and recovery, some data structures are stored in the memory. As shown in Fig. 3, there are three types of primary data (FTL Mapping Table, NVM Mapping Table, and In-Process Table) in the memory. Because the small-capacity NVM-based disk cache can only store limited data, NVM Mapping Table and In-Process Table are treated as persisted data which will be provided to host. Fig. 4 shows the bi-direction linked list structure of In-Process Table, which contains one or more transaction lists, and every transaction owns one transaction list. Each transaction list links the page mappings of all requests which belong to the transaction. For example, in Fig. 3, the transaction T7 has three request mappings which are linked in the same transaction list. When a write request arrives, system allocates a flash page for the write request and inserts the mapping of logical page number (LPN) to physical page number (PPN) to In-Process Table. After a transaction commits, NVCTX immediately updates the mappings of In-Process Table to NVM Mapping Table.

In Fig. 3, the sudden power-off (SPO) area stores activate transactions' information. If power fails, system flushes NVM Mapping Table entries to the Mapping Table area of flash memory, and flushes sequentially In-Process Table data to the SPO Area.

Traditional FTL Mapping Table maintains all logical-to-physical entries of flash memory before extending transactional interface. In NVCTX, the FTL Mapping Table is divided into two parts for fast system recovery. One part is stored in NVM-based disk cache, namely NVM Mapping Table; the other is stored in volatile cache, still called FTL Mapping Table. Fig. 5 shows the mapping relation of LPN to PPN.

Interface and Operation. FTL provides transaction interface to high-level application such as DBMSs and file systems. The embedded transaction operations are hidden in the FTL layer. In NVCTX, BEGIN, TWRITE, COMMIT and ABORT command interfaces are extended to FTL for supporting the transaction mechanism. BEGIN command is used to check whether a transaction identifier (TxId) is valid. If TxId is valid, system will insert TxId to the In-Process Table and initialize the transaction' status. Otherwise, an error message will be informed to system, and a new TxId should be reallocated. Because READ command does not update system data, NVCTX does not need to change original READ interface. WRITE command processes normal write requests which are not attributed to transactions. To support transactional write requests, TWRITE command extends a TxId parameter on the basis of traditional write interface. Its execution process demands system to allocate a flash page and write data to flash memory, then to insert logical-to-physical mapping' metadata of the write request to In-Process Table. When a transaction commits, COMMIT command updates the page mappings' metadata of the transaction to NVM Mapping Table. When a transaction aborts, ABORT interface is called. ABORT command interface processes transactional aborted requests and rolls back to consistent system status. When high-level applications call ABORT command, NVCTX only deletes the transaction's metadata, and does not update NVM Mapping Table and FTL Mapping Table, so original mappings will not be changed.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3004518, IEEE Transactions on Knowledge and Data Engineering

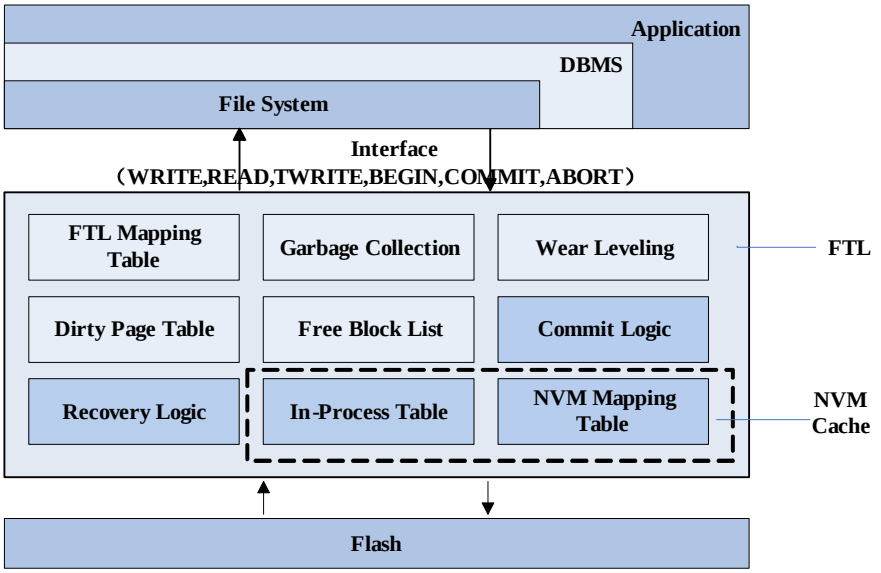


Fig. 2. NVCTX architecture.

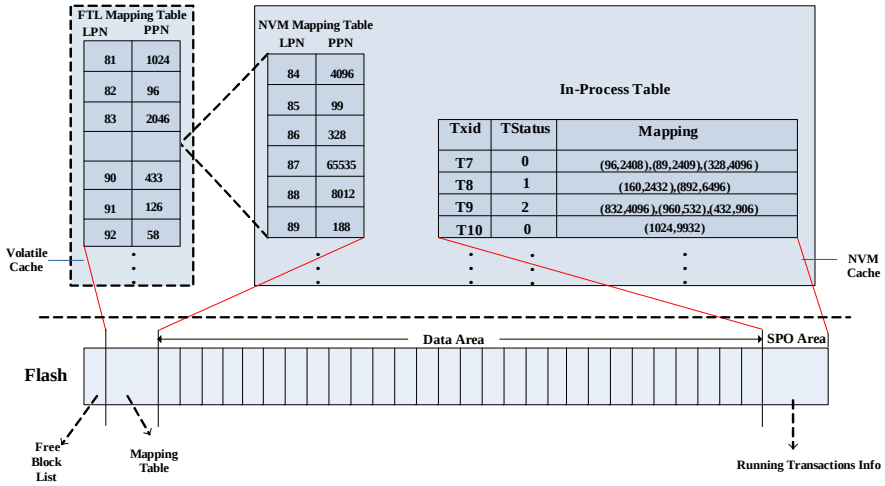


Fig. 3. NVCTX Core Components.

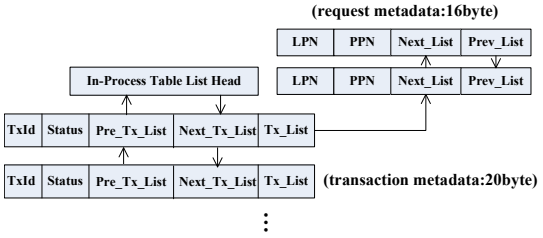


Fig. 4. In-Process Table structure.

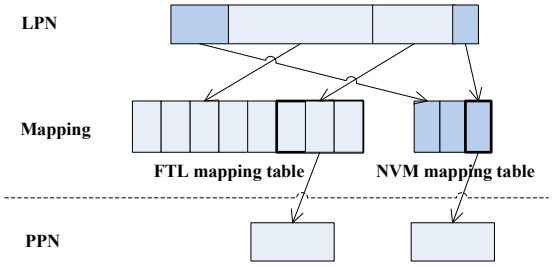


Fig. 5. NVCTX mapping.

3.2 NVCTX Commit Protocol

An embedded transaction protocol maintains data versions, and tracks transaction's status for rolling back the system to consistent status. In this paper, we address these issues by utilizing the non-volatile property of NVM-based disk cache.

Update Flow. To understand the NVCTX commit protocol, the update flow of NVCTX is shown in Fig. 6. First, NVCTX executes TWRITE command to allocate flash memory pages for writing requests. Second, the mappings' metadata of these write requests are kept in the In-Process Table in NVM-based disk cache. Third, after all the write requests of a transaction are written, a commit of the transaction will be executed. When the transaction commits, the mappings' metadata are updated to NVM Mapping Table from In-Process Table. Fourth, if a mappings' metadata are not located in the NVM Mapping Table, NVCTX will flush a mapping page chosen from NVM Mapping Table to flash memory. Fifth, the chosen mapping page of NVM Mapping Table is replaced by a mapping page from FTL Mapping Table. For example, the update flow for T8 in Fig. 3 including two page requests is as follows. (1) A flash page with a PPN value 2432 is allocated for first page request (LPN is 160), and the mapping (160, 2432) of LPN to PPN is inserted to In-

Process Table. For the second page request (LPN is 892), the mapping (892, 6496) is inserted to In-Process Table after acquiring another flash page (PPN is 6496). (2) When T8's commit command reaches, NVCTX updates the mapping of first page request to NVM Mapping Table. If LPN 160 is not in NVM Mapping Table, NVCTX looks up a mapping page which will be replaced out NVM Mapping Table and writes it to flash memory, and swaps it with a mapping page coming from FTL Mapping Table (containing LPN 160). Otherwise (i.e., LPN 160 is in NVM Mapping Table), NVCTX updates immediately the mapping to NVM Mapping Table. The second LPN (892) is similar to that of the first LPN. (3) If both mappings are successfully updated to NVM Mapping Table, T8 is successfully committed.

Versioning. Transaction atomicity requires the system to update all or none requests of a transaction to flash memory. New-version and old-version data are kept in the flash memory in the case of a power failure. In shadow paging (e.g., TxFlash), new-version data is located to free page, and the corresponding mapping metadata is updated. In WAL (e.g., ext4), data is appended to the log area, and is updated to data area when the system executes checkpoint operation.

Similar to shadow paging, NVCTX leverages no-overwrite property of flash memory. It updates data to free pages, and keeps these mapping entries for operative transactions in In-Process Table rather than NVM Mapping Table and FTL Mapping Table. Thus, NVCTX maintains new-version data and old-version data in the flash memory, and both versions are accessible. If a transaction commits, NVCTX updates associated mappings' metadata with the transaction's pages to NVM Mapping Table, and old-version pages are set to invalid and can be reclaimed by garbage collection mechanism.

Clustering. Clustering tracks the status of each transaction as well as its pages. When the system restarts after a power failure, clustering is used to identify all pages of each transaction to execute redo and undo operations for committed or aborted transactions. In WAL, clustering pages of each transaction are sequentially appended to the log area. And these clustering pages are copied to the data area when the system executes the checkpoint operation. Thus, WAL equals to write twice to persistent device, and leads to write amplification and shortens SSD lifetime. TxFlash clusters all pages of each transaction for identifying transaction status by using a cycle link list. However, garbage collection breaks the link list. In order to maintain the link list of each transaction, TxFlash needs to pay high overhead. NVCTX tracks active transactions instead of all transactions. So NVCTX only pays attention to the clustering of running transactions, which reduces clustering overhead enormously.

In NVCTX, the transaction's status recorded in In-Process Table is set to 0 when a transaction starts. It is set to 1 once transaction commits or set to 2 when transaction aborts. If a transaction commits, the page mappings' metadata of the transaction in In-Process Table is updated to NVM Mapping Table. Then NVCTX deletes the mappings' metadata in In-Process Table. Therefore, the mappings in both NVM Mapping Table and FTL Mapping Table belong to committed transactions. NVM Mapping Table and In-Process Table are flushed to flash memory when a power failure occurs. If NVM mapping pages need to be replaced, they will be written back to flash memory for persistent mapping relation. NVCTX writes mapping entries of a flash page size back to flash memory every time. When system restarts after a power failure, NVCTX can execute redo/undo operations only utilizing the data of SPO area.

Because the capacity of NVM-based disk cache is very limited, we propose a dynamic allocation algorithm and a hybrid storing method to take full advantage of NVM-based disk cache and improve the overall system performance. Dynamic allocation algorithm and hybrid storing method are designed as follows.

3.2.1 Dynamic Allocation

How to efficiently allocate and dynamically adjust small-capacity cache between In-Process Table and NVM Mapping Table is a vital issue. The size of In-Process Table is concerned with the number of active transactions' requests, and cannot be determined in advance. We denote the size of NVM-based disk cache by $CACHE_{total}$, and $CACHE_{ipt}$ means the cache size for In-Process Table. Then the cache size for NVM Mapping Table (i.e., $CACHE_{nmt}$) can be expressed as $CACHE_{total} - CACHE_{ipt}$. If the mapping's metadata of a committed request does not hit in NVM Mapping Table, NVCTX writes a replaced NVM mapping page to flash memory. To improve hit ratio, an available $CACHE_{ipt}$ needs to be held.

According to the observations and analysis of Section 2, the size of In-Process Table does not sharply change for a processing workload. Therefore, we design a dynamic and adaptive allocation algorithm based on the transaction size of a workload. According to the structure of Fig. 4, we can compute the size of In-Process Table when a transaction commits or aborts. Algorithm 1 shows the procedure of dynamic allocation. The lower bound factor α and the upper bound factor β are the regulating coefficients of $CACHE_{ipt}$. If the size of transactional metadata approaches $\beta \times CACHE_{ipt}$, $CACHE_{ipt}$ needs to increase; while the size of transactional metadata is less than $(1-\alpha)\times CACHE_{ipt}$, $CACHE_{ipt}$ will decrease. The tune factor γ is the coefficient of In-Process Table size turning up or down. The three statistics parameters provide the basis for the algorithm, and can be acquired by experiment and statistical analysis.

The algorithm's main steps are as follows. When a transaction commits or aborts, NVCTX computes the size of all active transactional metadata (denotes by *tot_mem*), and compares with the cache size of In-Process Table. If *tot_mem* is greater than $\beta\times CACHE_{ipt}$, NVCTX perceives that transactional metadata requires more space, and increases the cache size of In-Process Table by the minimum between $\alpha \times CACHE_{total}$ and $\gamma \times CACHE_{ipt}$. Otherwise when *tot_mem* is less than $(1-\alpha)\times CACHE_{ipt}$, NVCTX will reduce the cache size of In-Process Table by the maximum between $(1-\beta)\times CACHE_{total}$ and $1/\gamma \times CACHE_{ipt}$.

Algorithm 1 Dynamic Allocation Algorithm

1) Input: requests of transactions.
2) Output: the capacity of In-Process Table of current transactions.
3) Initialization: $CACHE_{ipt} \leftarrow \alpha\times CACHE_{total}$, $tot\_mem \leftarrow$ 0, initialize In-Process Table list head;
4) **if** a transaction commits or aborts
5) $tot\_mem \leftarrow 0$;
6) **for** each transaction in the In-Process Table
7) $tot\_mem \leftarrow tot\_mem + (20 + req\_num \times 16)$; //20 and 16 are transaction metadata size and request metadata size, coming from Fig. 4;
8) **if** $tot\_mem > \beta \times CACHE_{ipt}$ //β is the upper bound factor
9) $CACHE_{ipt} \leftarrow min(\alpha \times CACHE_{total}, \gamma \times CACHE_{ipt})$; //$\alpha$ is the lower bound factor, and γ is the tune factor, $CACHE_{ipt}$ increases;
10) **else if** $tot\_mem < (1-\alpha)\times CACHE_{ipt}$
11) $CACHE_{ipt} \leftarrow max((1-\beta)\times CACHE_{total}, 1/\gamma \times CACHE_{ipt})$;
12) **end if**
13) **end for**
14) **end if**
15) **End**

3.2.2 Hybrid Storing

Due to the very limited capacity of NVM-based disk cache, we adopt a dynamic allocation algorithm to improve system performance. In

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3004518, IEEE Transactions on Knowledge and Data Engineering

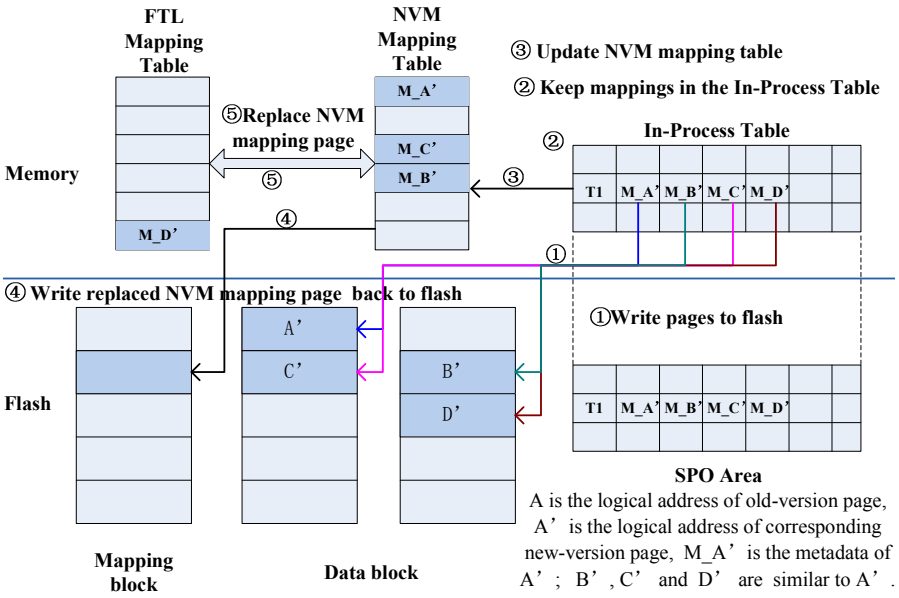


Fig. 6. Update flow of NVCTX.

Algorithm 1, to avoid flushing excessive mapping pages to the flash memory, the lower bound factor (α) is set to ensure the minimal size of NVM Mapping Table. However, some workloads (such as file-server) containing thousands upon thousands requests in a transaction require more NVM-based disk cache to store operative transactional metadata. Under this kind of situation, the maximal In-Process Table allocated from NVM-based disk cache may still be insufficient to store active transactional metadata. Therefore, we design a hybrid storing method in NVCTX by combining the NVM-based disk cache for storing a part of transactional metadata with the flash memory for storing the other part. The reason is that we choose to flush partial transactional metadata to the flash memory rather than NVM mapping pages, it can be explained as follows. First, if a transaction has committed or aborted, its metadata in flash memory will be invalid, and be reclaimed by garbage collection mechanism, so the overhead of tracking persistent transactional metadata can completely be eliminated. Second, reducing the size of NVM Mapping Table will decrease mapping hit ratio, lead to higher frequency of replacing a NVM mapping page to flash memory, prolong write latency, and weaken overall performance. Based on the above two reasons, if the capacity of NVM-based disk cache is severely insufficient, we choose to write partial transactional metadata to flash memory. How NVCTX processes transactional metadata by the hybrid storing method is introduced in the following section.

When the capacity of In-Process Table acquired from NVM-based disk cache is insufficient to store all metadata (mappings, not contain transaction status and identifier) of active transactions, NVCTX writes the transactional metadata to the volatile memory and appends metadata pages to the SPO area to make sure system consistency and achieve fast recovery. Nevertheless, in order to avoid reading metadata page of SPO area when a read request arrives, NVCTX maintains a map of SPO area in the volatile memory. Fig. 7 shows the page layout in SPO area when the system restarts after a power failure. The former part is normal metadata pages which are from volatile memory, and the latter part is power protected metadata pages which are flushed to the SPO area when the system crashed. The page metadata stored inside Out-of-Band area (OOB) is extended to contain three new fields (PPF (power protected flag), TxId and LPN). PPF denotes a flag used to decide whether the page is

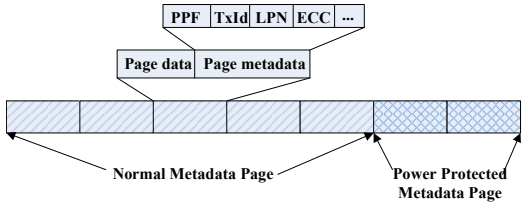


Fig. 7. Page layout in SPO area.

written SPO area after a power failure occurs, TxId means transaction identifier, LPN represents logical page number. NVCTX executes the following steps.

1) If the size of transactional metadata in volatile memory reaches a flash page, and the transaction has not committed, NVCTX sequentially appends the page to the SPO area. PPF is unset, and TxId field is set to the transaction identifier. If power failure occurs, PPF will be set, In-Process Table is written to SPO area, NVM Mapping Table is written to mapping table area in flash memory.

2) When a transaction commits, its last metadata is written to SPO area, and then the transactional metadata in NVM-based disk cache and volatile memory will be updated to NVM Mapping Table. Otherwise, if a transaction aborts, NVCTX frees all metadata of the transaction in NVM-based disk cache and volatile memory.

With the above two steps, NVCTX still can reserve appropriate capacity of NVM Mapping Table by borrowing partial volatile memory to expand In-Process Table, and ensures high hit ratio. Fast recovery can be completed by scanning SPO area in which transactional metadata is contained.

3.3 Crash Recovery and Correctness Proof

Almost all recent researches need to scan whole SSD to execute redo or undo operations, and reconstruct the mapping table in the memory during crash recovery, such as SCC and BPCC, which incurs high recovery cost. The crash recovery time is proportional to the number of flash pages. X-FTL just needs to read the X-L2P table from the flash chips and reflect the committed status entries to the L2P table (traditional page mapping table). But the X-L2P table in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3004518, IEEE Transactions on Knowledge and Data Engineering

X-FTL have to be written to a new location in flash with every transaction committing, which causes high performance overhead and write amplification. However, NVCTX only scans SPO area for executing redo or undo operations. In-Process Table contains status of all active transactions, so the transactions' status can be accurately read from In-Process Table flushed to SPO area. Normal metadata pages are in front of power protected metadata pages in SPO area. NVCTX can distinguish between normal metadata pages and power protected pages by the PPF field in page metadata. During restart after system crashes, power protected pages are read to NVM-based disk cache, while normal metadata pages are read to volatile memory. Fig. 8 shows the flow of crash recovery. The following detailed steps are demonstrated as a sketch of proof that the crash recovery in NVCTX correctly executes all redo or undo operations to recover the FTL mapping table.

1) Because power protected metadata pages are flushed to the bottom of SPO area, NVCTX first scans inversely SPO area and reads a flash page, and decides whether the page is power protected metadata page by its PPF field of page metadata.

2) If the page is power protected metadata page, NVCTX adds the metadata page to In-Process Table in NVM-based disk cache, and scans next page in SPO area. Otherwise, NVCTX acquires TxId field from the page metadata, and lookups TxId in In-Process Table.

3) If the TxId is not located in In-Process Table, which indicates that the metadata page is terminate page, and those metadata pages used to recover have been read from SPO area, NVCTX stops to scan flash page in SPO area, and frees memory space of the metadata page, the committed transactional metadata in In-Process Table will be updated to NVM Mapping Table. Otherwise, the page is normal metadata pages, for a committed transaction, NVCTX updates mappings metadata to NVM Mapping Table and frees transactional metadata, while NVCTX does not update mappings metadata for an uncommitted transaction, only frees corresponding transactional metadata space.

With the above three steps, transactions are identified as either committed or not-committed. And the committed transactions are accessible by FTL Mapping Table or NVM Mapping Table, while the not-committed ones are later reclaimed by garbage collection. Thus, the FTL mapping information are recovered correctly.

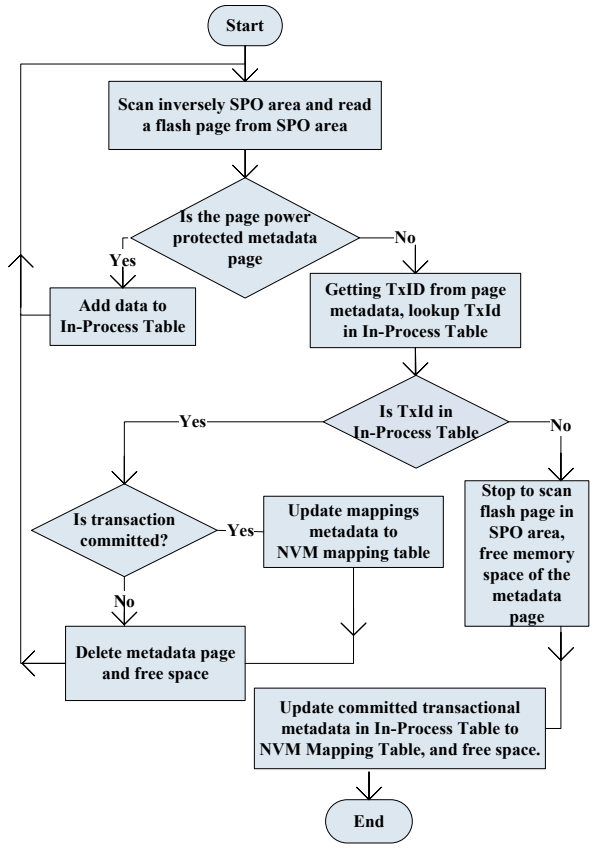


Fig. 8. Crash recovery flow for NVCTX.

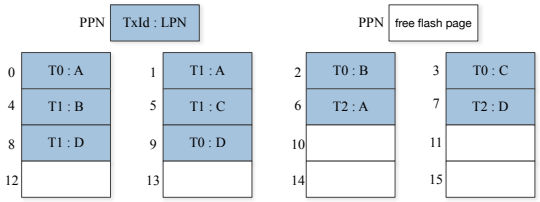


Fig. 9. Write concurrency.

3.4 Concurrency Discussion

In this subsection, we discuss concurrency control under the situation that different versions of a single page may be accessed by different transactions. Because of the out-of-place mode in SSD, new-version pages are written to free flash pages instead of overwrite old-version data, so new-version pages will not affect old-version data. If multiple transactions update the same logical pages at the same time, NVCTX can prevent a transaction being interfered from other transactions, i.e., NVCTX supports high write concurrency inside SSD. We take Fig. 9 as an example to demonstrate that how the NVCTX protocol supports it. In Fig. 9, the number to the left of the rectangle (i.e., number zero to fifteen) represents PPN, and the letters A to D represent LPN. Blank rectangles mean free flash pages while blue rectangles indicate the pages which have been written. When transaction T0, T1, and T2 concurrently update logical page A, NVCTX maintains three new-version page A (T0: A for T0 transaction, T1: A for T1 transaction, and T2: A for T2 transaction) in the flash memory. TX: A (X = 0, 1, 2) for different transactions does not interfere each other, but also does not affect old-version page A. To avoid multiple committed transactions updating NVM Mapping Table concurrently, NVCTX uses a simple lock to protect the availability of mapping entries in NVM Mapping Table and FTL Mapping Table. For recording the sequence of transactions commit, NVCTX handles a global variable in volatile memory. When a transaction accomplishes commit operation, its status value in In-Process Table is delivered to the global variable. NVCTX can also guarantee data consistency even if system faces the following two situations.

1) One or multiple transactions concurrently updating the same pages are aborted. In Fig. 9, we assume that T0 is aborted. Because NVCTX identifies whether a flash page is valid by FTL Mapping Table and NVM Mapping Table, and committed transactions could change mapping entries of two tables, even if T0 is aborted, it also does not have an impact on T1 and T2. NVCTX only needs to delete T0's metadata in NVM-based disk cache, and does not update mapping entries of T0's requests to NVM Mapping Table, which indicates indirectly that flash page 0, 2, 3, and 9 in Fig. 9 will be invalid.

2) When a power failure occurs, NVCTX writes data in NVM-based disk cache to flash memory. NVCTX reads data of SPO in flash memory to NVM-based disk cache to restore In-Process Table in the moment of power on. If there are multiple committed transactions in In-Process Table, NVCTX updates in the correct order mapping entries in In-Process Table to NVM Mapping Table according to the

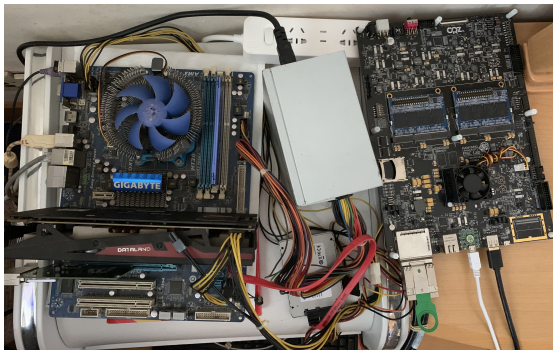


Fig. 10. OpenSSD connected to host system.

status value of committed transactions.

NVCTX supports high concurrency for different transactions, but it does not consider read requests in concurrent transactions. For example, in Fig. 9, T3 has a read request for page A when T0, T1, and T2 all do not commit, and NVCTX cannot determine which page T3 should read from flash page 0, 1, 6, or old-version page A. A lock manager is required to avoid transaction read acquiring data in unfinished transactions or getting the outdated data. As we have discussed in Section 3, NVCTX does not focus on read isolation. We implement the lock manager for read isolation in host machine instead in the SSD.

4 Implementations

In this section, we present the implementation details of the NVCTX architecture on the OpenSSD platform, the modified file system called NVExt4, NVCTX implementation for databases, and the NVM-based disk cache partition.

NVCTX implementation in OpenSSD. OpenSSD is an open-source development platform by the OpenSSD Project [33]. It can develop the open source SSD firmware on OpenSSD platforms so that researchers could verify and improve their solid-state drive (SSD) technology. In this paper, we have implemented the NVCTX protocol on an OpenSSD platform called Cosmos+ FPGA platform based on the HYU Tiger 4 controller, which is a real hardware board and operates as a real solid-state drive. Thus, it has the same performance characteristics as a common SSD device. The Cosmos+ FPGA platform uses a pure page-level mapping strategy with 16 KB (Kilobyte) page for flash memory management, and the board is attached to a host machine through the NVMe over PCIe (Peripheral Component Interconnect express) interface as shown in Fig 10.

For prototyping NVCTX, the Cosmos+ FPGA platform extends the interface as shown in Table 1 and the components as shown in Fig 3. In the Cosmos+ FPGA platform, we use In-Process Table to track active transactions. For a write request, the OpenSSD platform allocates a physical page and updates the mapping metadata (LPN, PPN), in which LPN denotes the logical page number, and PPN means physical page number, to In-Process Table. When a transaction commits, the mappings metadata of the transaction's pages are updated to NVM Mapping Table and removed from In-Process Table. The whole mapping entries of committed transaction consist of NVM Mapping Table and FTL Mapping Table. In the Cosmos+ FPGA platform, we implement NVM Mapping Table by recording the information of whole mapping table pages. The information contains the last accessing time of the mapping page and the flag used to determine whether the mapping page is in the NVM Mapping Table or not. If a mapping metadata does not exist in NVM Mapping Table, a LRU (least recently used) algorithm is employed to execute replacement policy of mapping page for accessing mapping table. The replaced NVM mapping page is written to flash memory.

SCC and BPCC protocols in TxFlash and X-FTL protocol are also implemented in the Cosmos+ FPGA platform. They all extend the interface and FTL components which are slightly different from NVCTX. SCC and BPCC keep metadata in the DRAM embedded on the board. The metadata of TxFlash is linked by cycle list for each transaction to guarantee system data consistency. X-FTL uses the X-L2P table to combine with L2P table, and flushes the entire X-L2P table to the flash memory when a transaction commits. To evaluate the trade-offs, we need to consider the traditional commit protocol (WAL) which is used in host machine (e.g., DBMS and file system), and runs it on the top of a standard SSD. Therefore, we also implement WAL traditional commit protocol in the Cosmos+ FPGA platform. In WAL, the log area is mapped to the DRAM embedded on the board, and the metadata of pages in log area is recorded by list. The size of log area is set to 32M. The data of log area is written to data area when its free pages are less than a preset threshold value.

NVExt4. As TxFlash did, we also reemploy the journaling module (jbd) of Ext4 file system instead of designing a new transaction abstraction in file system. We modify the jbd module to use the transactional interfaces as shown in Table 1 to build the NVExt4 file system, which exists no separate checkpointing process as in Ext4. Transaction ids (i.e., TxIds) are managed by the NVExt4 file system. In the meantime, for supporting the new transactional commands, we extend the NVMe driver that TWRITE(TXID, LBA, Len), COMMIT(TXID) and ABORT(TXID) are added to the standard NVMe command set as new commands.

NVCTX implementation for Databases. Most modern databases rely on transaction mechanism to ensure atomicity and durability of data. However, since we implement NVCTX inside SSD with expanding existing FTL layer interfaces to support transaction mechanism, the databases don't have to run in transaction mode. As X-FTL did, we can just simply turn them off. Nevertheless, when the databases run in transaction off mode, it cannot deal with the abort transaction because of the steal policy. To address this problem, the databases should be modified so that the aborting transaction can pass to NVCTX. This is the only modification in database systems, and in this paper, we revise PostgreSQL database source code as the database of TPC-H and TPC-C workloads to evaluate NVCTX and previous transaction protocols (i.e., WAL, TxFlash, and X-FTL).

NVM-based Disk Cache Partition. Due to the very limited NVM-based disk cache size, we need to consider carefully how to allocate In-Process Table and NVM Mapping Table from NVM-based disk cache. Through repeated experiments and statistical analysis, we get a set of appropriate parameter values, namely, α is set to 0.5, β is set to 0.9, and γ is set to 2. (See Section 5.4 for experimental statistic). In the beginning, both the In-Process Table and NVM Mapping Table respectively acquire a half of NVM-based disk cache size when the system starts. To take full advantage of NVM-based disk cache, NVCTX will adjust the size between In-Process Table and NVM Mapping Table according to dynamic allocation algorithm and dynamic nature of In-Process Table.

5 Evaluation

In this section, we compare NVCTX with TxFlash (SCC and BPCC), X-FTL, and WAL in the following aspects: 1) performance; 2) the impact of transactional abort ratio on the system performance; 3) the impact of NVM-based disk cache allocation and hybrid storing; and 4) system recovery time.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3004518, IEEE Transactions on Knowledge and Data Engineering

5.1 Experimental Setup

The Cosmos+ FPGA platform is equipped with HYNIX H27Q1T8YEB9R flash memory chips with 16KB pages and 128 pages per block. The controller of Cosmos+ FPGA platform is HYU Tiger4 with up to 1GHz ARM processor, and has 1GB (Gigabyte) DRAM to store the data like FTL Mapping Table. The total size of Cosmos+ PFGA platform is 1TB (940GB available). The default size of NVM-based disk cache is set to 1MB (the impact of NVM-based disk cache allocation is evaluated in Section 5.4).

The host machine is an AMD Athlon II 2-core processor running at 2.7GHz, with 4GB of RAM and 256G SSD. We install Ubuntu 18.04 based on Linux kernel 4.19.12 with the NVExt4 file system and the modified NVMe driver. The Cosmos+ FPGA platform is connected to the host machine by an external PCIe cable and an external PCIe adapter on host platform board.

We use a variety of workloads including TPC-H, TPC-C, SysBench, and FIO (Flexible I/O) for the evaluation. TPC-H [34] is a decision support benchmark supplied by the TPC (Transaction Processing Council). It is a database workload that includes twenty-two queries. We use the revised PostgreSQL database which is mentioned in Section 4 as the on-line database. The workload contains 125 million tuples with an average size of 88 bytes (range from 28 bytes to 598 bytes). TPC-C benchmark is also an on-line transaction processing (OLTP) benchmark, and we use the DBT2 tool [38], [18] which is a fair use implementations of TPC-C for our experiments, and a write-intensive workload is created by setting the ratio of five transaction types (i.e., Payment 43%, New Order 45%, Stock Level 4%, Delivery 4%, and Order Status 4%). The DBMS on top running DBT2 is also the modified PostgreSQL database. SysBench [16] is a multi-threaded benchmark tool which is most frequently used for database benchmarks. In the prepare phase, we generate a total of 900G data size files for our evaluation. the read/write ratio is set to 3:2, and the block size equals to 16KB. FIO benchmark [35] is an I/O tool for testing the storage system performance. It is run on the random write performance of a file system, and the size of a page is 16KB. We run these four workloads in this section to evaluate NVCTX and previous transaction protocols (WAL, TxFlash, and X-FTL).

5.2 Performance

We evaluate the performance using transaction throughput in the case of different workloads for WAL, TxFlash (including SCC and BPCC), X-FTL, and NVCTX. The experimental results are shown in Fig. 11. These results are measured with the condition that the abort ratio is 0. Compared with NVCTX, X-FTL has a little low performance. The reason is that the entire X-L2P table in X-FTL will be written to flash memory with every transaction committing. TxFlash and NVCTX show the same performance. This is because the garbage collection mechanism of TxFlash is the same as normal SSD GC, and TxFlash can acquire the best performance in the case of no abort transaction. However, there generally are aborted transactions in file system and DBMS [39], [24]. With rising of the transaction abort ratio, the performance of TxFlash and WAL will come down faster than that of NVCTX. We describe the situation in the following section.

5.3 Impact of Abort Ratio

To evaluate the impact of abort ratio on different transactional commit protocols, we measure four respects for different workloads including transaction throughput, commit latency, endurance, and garbage collection (GC) time.

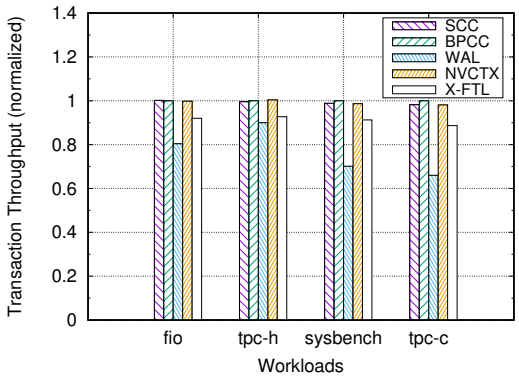
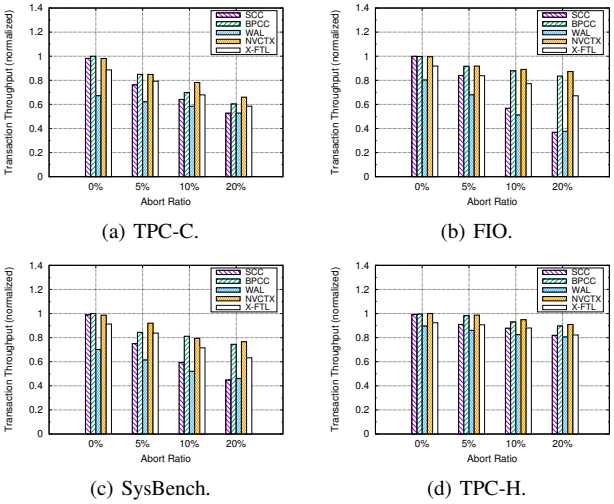


Fig. 11. Transaction throughput.

Performance. To evaluate the performance for different abort ratios, we measure the transaction throughput of committed transactions. As shown in Fig. 12, the transaction throughput degrades along with the increase of abort ratio. Among five protocols, because SCC needs to erase aborted pages when system writes new pages, it shows sharp degradation for different abort ratio. WAL shows slower degradation than SCC since the pages of aborted transactions don't need to be written back to home location when the system executes checkpoint operation. Fig. 12(a) shows that the performance of NVCTX outperforms that of SCC, BPCC, WAL, and X-FTL by 25%, 9.4%, 25.1%, 12.9% respectively when the abort ratio is 20% for TPC-C workload. When the abort ratio is 20% in Fig. 12(b), NVCTX enhances the throughput of transaction by 136.5%, 4.5%, 131.6% and 29.9% in FIO workload compared to SCC, BPCC, WAL, and X-FTL, respectively. NVCTX promotes the throughput for SysBench workload by 70.9%, 2.9%, 66.5% and 21.2% compared to SCC, BPCC, WAL, and X-FTL respectively when the abort ratio is 20%, as shown in Fig. 12(c). For TPC-H workload, NVCTX achieves 10.9%, 2%, 13% and 11% improvement respectively compared with SCC, BPCC, WAL, and X-FTL when the abort ratio is 20% in Fig. 12(d). At the same time, Fig. 12 prompts that NVCTX shows an average decrease in performance with the increasing of abort ratio.



(a) TPC-C. (b) FIO.

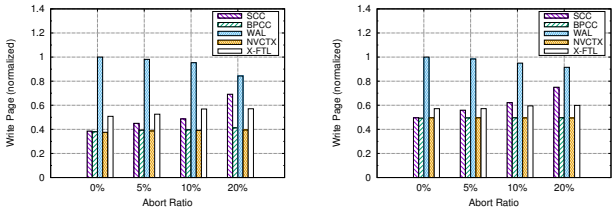
(c) SysBench. (d) TPC-H.

Fig. 12. Performance under different abort ratios.

Endurance. We evaluate the endurance of flash memory by measuring the number of pages written to flash. TPC-C, FIO, SysBench and TPC-H workloads are chosen to count the number of pages written to flash memory. As shown in Fig. 13(a) for TPC-C workload,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3004518, IEEE Transactions on Knowledge and Data Engineering

comparing with WAL, SCC reduces the number of writing pages by 61.5%, 54.1%, 49%, and 18.1% respectively when abort ratio is 0%, 5%, 10%, and 20%; BPCC decreases the number of writing pages by 62%, 60.6%, 58.9%, and 53.1%; NVCTX decreases the number of writing pages by 62.4%, 60.6%, 58.9%, and 53.1%; X-FTL lower the number of writing pages by 49.2%, 46.4%, 40.3% and 32.3%. Comparing with SCC, NVCTX reduces the number of writing pages by 2.6%, 14.2%, 19.6%, and 42.8% respectively when the abort ratio is 0%, 5%, 10%, and 20%. Comparing with BPCC, NVCTX reduces the number of writing pages by 1.1%, 1.9%, 1.1%, and 4.1% respectively when the abort ratio is 0%, 5%, 10%, and 20%. And comparing with X-FTL, NVCTX decreases the number of writing pages by 26%, 26.5%, 31.2% and 30.7% respectively when the abort ratio is 0%, 5%, 10%, and 20%. In summary, among the five commit protocols, NVCTX writes fewest pages to flash memory, which reduces write amplification and prolongs the lifetime of flash memory. The five protocols show similar result in Fig. 13(b), Fig. 13(c) and Fig. 13(d) respectively for other three workloads including FIO, SysBench, and TPC-H. In addition, with the raising of abort ratio, NVCTX writes fewer pages to flash memory than other four protocols. There is also one observation from the Fig. 13, with the increasing of transactional abort ratio, WAL writing pages become slightly fewer. The main reason is that WAL does not need to write pages aborted in the log area back to home location when system executes checkpoint operation, and system can immediately reclaim these pages.

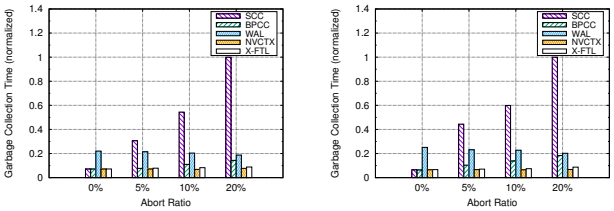


(a) TPC-C. (b) FIO.

(c) SysBench. (d) TPC-H.

Fig. 13. Endurance under different abort ratios.

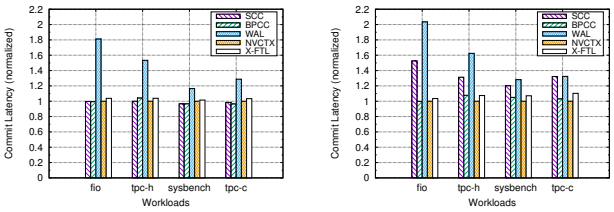
GC Time. To provide better understanding for the performances under the different abort ratios, we measure the garbage collection time of five protocols for TPC-C and FIO workloads in the case of the different abort ratios. Four observations are found in the Fig. 14. First, the garbage collection cost of NVCTX has no significant change under different abort ratios. Second, SCC has the highest garbage collection overhead. The main reason is that SCC needs to erase uncommitted pages when system writes new pages, and leads to triggering garbage collection frequently. Third, the garbage collection time of NVCTX is lower than that of other three protocols. Fourth, the garbage collection time of TxFlash (SCC and BPCC) becomes longer and longer with the increasing of abort ratio. Especially, SCC's garbage collection time takes on sharp growing trend. This can be explained as follows: SCC must enforce to erase an old-version page uncommitted before a new-version page is written to flash memory. When a block contains valid pages and uncommitted pages, its valid pages all need to be migrated to free pages, which incurs additional write cost. However, BPCC needs to reclaim the pages of straddle responsibility set (SRS) before reclaiming an uncommitted page. If SRS of an uncommitted page is not null, the page is moved to free page when garbage collection mechanism reclaims the block which contains the page. These restrictive conditions cause high garbage collection overhead. As shown in Fig. 14(a), comparing with SCC, BPCC, WAL, and X-FTL, NVCTX outperforms by 92.3%, 46.2%, 58.8% and 12.5% respectively for TPC-C workload when the abort ratio is 20%. Similarly, NVCTX achieves the improvement of 93.2%, 63%, 66.5% and 22.1% respectively when abort ratio is 20% in Fig. 14(b). Similar results can be acquired for SysBench and TPC-H workloads. We conclude that NVCTX has the lowest garbage collection overhead compared with WAL, SCC, BPCC, and X-FTL.



(a) TPC-C. (b) FIO.

Fig. 14. GC time under different abort ratios.

Commit Latency. To view whether the improved performance comes at the expense of worse latency, we evaluate the average commit latency for five commit protocols when the transactional abort ratio is 0% and 20% respectively. Fig. 15 shows the experimental results. In Fig. 15(a), NVCTX, SCC, BPCC and X-FTL show almost the same commit latency for four workloads when the abort ratio is 0%. WAL increases the commit latency by 79.9%, 50.5%, 18.1%, and 29.5% respectively for FIO, TPC-H, SysBench, and TPC-C comparing with NVCTX, SCC, BPCC and X-FTL. In Fig. 15(b), NVCTX has less commit latency than other protocols for different workloads. For example, for TPC-H workload, NVCTX reduces the commit latency by 18%, 7.3%, 19.2% and 7% comparing with SCC, BPCC, WAL and X-FTL when abort ratio is 20%. The main reason is that NVCTX provides fast metadata access and high hit ratio to NVM-based disk cache. Fig. 15 demonstrates that NVCTX has less commit latency for different workloads when there exist aborted transactions, and it also does not increase obviously the commit latency even when abort ratio is 0.



(a) Commit Latency (abort ratio 0%). (b) Commit Latency (abort ratio 20%)

Fig. 15. Commit latency under different abort ratios.

5.4 Impact of NVM-based Disk Cache Allocation

In NVM-based disk cache, on the one hand, we keep the metadata of activate transactions in In-Process Table; on the other hand, NVM-based disk cache is efficiently exploited for lessening write frequency to flash memory for persistent mapping table. To tradeoff these two

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3004518, IEEE Transactions on Knowledge and Data Engineering

techniques, NVCTX can dynamically adjust the size of In-Process Table according to Algorithm 1.

The three statistics parameters in Algorithm 1 are acquired by experiment and statistical analysis. Table 2 lists a part of corresponding experimental results under SysBench workload. From the statistical results, the transaction throughout achieves optimum when α, β, and γ are respectively set to 0.5, 0.9, and 2. Similar statistical results are also obtained for FIO, TPC-H and TPC-C workloads.

TABLE 2
Three statistics parameters

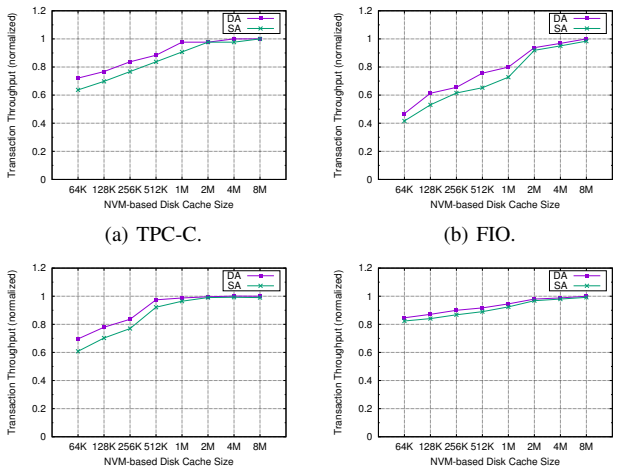
α	β	γ	Transaction Throughput (Mb/sec)
0.5	0.9	2	352.87
0.2	0.9	2	314.85
0.5	0.9	3	322.37
0.5	0.8	2	323.37
0.2	0.8	2	321.39
0.5	0.8	3	323.54
0.5	0.6	2	323.23
0.2	0.6	2	315.89
0.5	0.6	3	320.67

We evaluate static allocation (SA) and dynamic allocation (DA) by using different NVM-based disk cache sizes and four different workloads. Experimental results are shown in Fig. 16. From the figure, for both SA and DA, the transaction throughput is increasingly better with the increase of NVM-based disk cache size, and tends to a stable value after the size of NVM-based disk cache is greater than 1M. The critical factor which brings on the phenomenon is locality. Transaction requests from the same workloads have better locality which can further heighten hit ratio. With the increase of NVM-based disk cache size, more mapping pages are kept in NVM-based disk cache, which enhances hit ratio and reduces mapping pages written to flash memory. For example, when the size of NVM-based disk cache is greater than 2M, the cache size allocated to NVM Mapping Table for SA can store all mapping pages which are accessed to update mapping relation for SysBench, TPC-H and TPC-C. It is unnecessary to place mapping pages to flash memory. Thus, DA and SA show the same transaction throughput for SysBench, TPC-H and TPC-C. Due to the poor locality of FIO, the transaction throughput of SA is lower than that of DA because the cache size allocated to NVM Mapping Table for SA does not have enough space to store all mapping pages.

From Fig. 16, it can also be viewed that DA improves performance by 15.6%-5.4% higher than of SA, when the size of NVM-based disk cache is less than 1M. This is because the size of In-Process Table is dynamically changed according to the number of operative transactions. DA can exploit the dynamic nature of In-Process Table to adjust the size of NVM Mapping Table for higher hit ratio. However, the size of In-Process Table and NVM Mapping Table is statically allocated in SA, and cannot be regulated. Due to sparse transactional metadata, spare In-Process Table may be wasted. The experimental results illustrate that DA can effectively reduce write amplification.

5.5 Impact of Hybrid Storing

In this section, in order to demonstrate the availability of hybrid storing method, we measure the performance and endurance for SysBench workload under hybrid storing and single storing (transactional metadata all is stored in NVM-based disk cache) by changing NVM-based disk cache size.

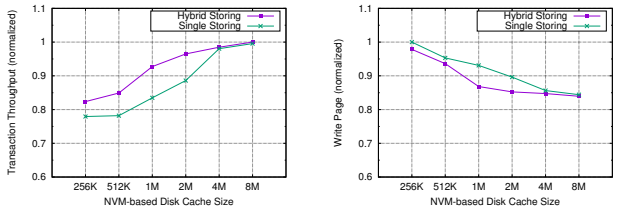


(a) TPC-C. (b) FIO.

(c) SysBench. (d) TPC-H.

Fig. 16. Transaction throughput of DA and SA under different NVM-based disk cache size.

Fig. 17(a) shows the transaction throughput of hybrid storing and single storing under different NVM-based disk cache sizes. When the size of NVM-based disk cache is less than 4M, the performance of hybrid storing outperforms that of single storing. For example, the transaction throughput is respectively 330.1 (Mb/sec) and 297.1 (Mb/sec) for hybrid storing and single storing in the case of 1M of NVM-based cache. Hybrid storing shows an approximately 11.1% increase in transaction throughput compared to single storing. This is because the size of NVM Mapping Table decreases with operative transactional metadata increasing. If NVCTX adopts single storing, the hit ratio of mapping lowers, which leads to more mapping pages written to flash memory. However, hybrid storing reserves appropriate size of NVM Mapping Table by flushing transactional metadata to SPO area, and ensures high hit ratio of mapping. Fig. 17(b) shows the endurance of hybrid storing and single storing under different NVM-based disk cache size. Single storing writes more mapping pages to maintain mapping persistency than hybrid storing (e.g. 195589 and 182437 in 1M of NVM-based cache), while the number of pages written to flash reduces by 3% on average by using hybrid storing method. When the size of NVM-based disk cache is greater than 4M, the performance of hybrid storing is the same with that of single storing. The main reason is that the cache size allocated to NVM Mapping Table from NVM-based disk cache can plentifully store all mapping pages. Therefore, hybrid storing in NVCTX improves transaction performance by promoting hit ratio in the case of NVM-based disk cache with small capacity, and prolongs flash lifetime by decreasing write traffic.



(a) Transaction throughput under different NVM-based disk cache size. (b) Endurance under different NVM-based disk cache size.

Fig. 17. Impact of hybrid storing method.

5.6 Recovery Time

We also measure the recovery time to evaluate five commit protocols. Because they all require expending the same time to read persistent mapping table during system recovery, we can exclude the time of reading persistent mapping table when measuring system recovery time. In addition, TxFlash does not use battery backed RAM so that SCC and BPCC need to scan whole flash memory to be sure of a page's status. WAL scans log area (32M) for system recovery. However, NVCTX only needs to read persistent In-Process Table whose page number is few in the flash memory. X-FTL also just requires to read the X-L2P table from the flash chips, but when a transaction commits, the whole X-L2P table has to be written to the flash memory every time, which decreases the performance and results in a write amplification. We measure the recovery time in four workloads and average them. The recovery time of different protocols is shown in Table 3. SCC and BPCC take 5586.28ms and 5554.52ms, WAL takes 205.13ms, and X-FTL takes 3.94ms. NVCTX only requires 2.17ms to recover to system consistency status. Thus, NVCTX can completely achieve the goal of fast recovery.

TABLE 3
Recovery time of five commit protocols

Protocol	SCC	BPCC	WAL	X-FTL	NVCTX
Recovery Time (ms)	5586.28	5554.52	205.13	3.94	2.17

From Table 3, the recovery time of WAL whose log area size is 32M also is not too long. However, the recovery time of WAL will grow linearly with increase of the log area size. SPO area size is only associated with the metadata size of running transactions, so the recovery time of NVCTX is not changed too much.

6 Conclusion and Future Work

This paper presents the design, implementation, and evaluation of NVCTX, an embedded transaction protocol inside SSDs managing the small-capacity NVM-based disk cache as transactional metadata storing to reduce transaction overhead and provide fast recovery. With efficient data structures and algorithmic designs for non-volatile cache storage management, NVCTX expands only existing FTL layer interfaces to achieve good throughput and low recovery time in the case of high transaction abort ratio, while hiding embedded transaction operations in the FTL layer to retain modularity and composability. NVCTX combines the small-capacity non-volatile disk cache with flash memory to make sure data consistency and durability. Both the two techniques (i.e. dynamic allocation algorithm and hybrid storing method) enable better performance and lower transaction overhead. Evaluations using DBMS and file system workloads show that NVCTX improves transaction throughput by up to 136.5%, 9.4%, 131.6% and 29.9%, reduces write traffic to flash memory by up to 42.8%, 4.1%, 62.4%, and 31.2%, lowers garbage collection overhead by up to 93.2%, 63%, 66.5%, 22.1%, and shortens recovery time to 1/2574, 1/2559, 1/95 and 1/2, respectively compared with SCC, BPCC, two typical embedded transaction protocols, WAL which is a traditional transaction protocol, and X-FTL which is a transactional FTL for SQLite databases.

In this paper, we have built a prototype system based on OpenSSD Platform. However, because the Cosmos+ FPGA platform could not change its capacity, the experiment for the impact of SSD capacity couldn't be evaluated in OpenSSD Platform. We also do not show the simulator evaluation in this paper because of the page limits. Modifying the underlying FPGA code to change the capacity of OpenSSD Platform is a viable approach. Besides, a comparison to ARIES, which is an optimized WAL-based approach with no-force and steal strategies and learned by many databases, should be considered. The work is still in progress. In the future work, with a stable code base of protocol, there exist numerous possibility we could extend. We intend to explore the feasibility of extending NVCTX to multiple SSDs, and study the effect on performance with mixed workloads. Another area of research will be to consider improving the OpenSSD Platform to enhance performance, since it is also a SSD prototype platform which has improvement space.

7 Acknowledgments

We are grateful to three anonymous reviewers for their criticism and comments on our previous submission of the manuscript. The research is partially supported by the National Natural Science Foundation of China under grants 61672218, and Hunan Provincial Innovation Foundation For Postgraduate under grants CX2018B229.

References

[1] Philip A Bernstein and Nathan Goodman. The failure and recovery problem for replicated databases. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 114–122. ACM, 1983.

[2] Philip A Bernstein and Nathan Goodman. An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Transactions on Database Systems (TODS)*, 9(4):596–615, 1984.

[3] Jin-Yong Choi, Eyee Hyun Nam, Yoon Jae Seong, Jin Hyuk Yoon, Sookwan Lee, Hong Seok Kim, Jeongsu Park, Yeong-Jae Woo, Sheayun Lee, and Sang Lyul Min. Hil: A framework for compositional ftl development and provably-correct crash recovery. *ACM Transactions on Storage (TOS)*, 14(4):1–29, 2018.

[4] Joel Coburn, Trevor Bunker, Meir Schwarz, Rajesh Gupta, and Steven Swanson. From aries to mars: Transaction support for next-generation, solid-state drives. In *Proceedings of the twenty-fourth ACM symposium on operating systems principles*, pages 197–212. ACM, 2013.

[5] Klaus Elhardt and Rudolf Bayer. A database cache for high performance and fast restart in database systems. *ACM Transactions on Database Systems (TODS)*, 9(4):503–525, 1984.

[6] Yulei Fan and Xiaofeng Meng. Mixsl: an efficient transaction recovery model in flash-based dbms. In *International Conference on Web-Age Information Management*, pages 393–404. Springer, 2013.

[7] Filebench benchmark. http://sourceforge.net/projects/filebench/.

[8] Shen Gao, Jianliang Xu, Theo Härder, Bingsheng He, Byron Choi, and Haibo Hu. Pcmlogging: Optimizing transaction logging and recovery performance with pcm. *IEEE Transactions on Knowledge and Data Engineering*, 27(12):3332–3346, 2015.

[9] Goetz Graefe and Harumi Kuno. Definition, detection, and recovery of single-page failures, a fourth class of database failures. *arXiv preprint arXiv:1203.6404*, 2012.

[10] Jim Gray, Paul R. McJones, Mike W. Blasgen, Bruce G. Lindsay, Raymond A. Lorie, Thomas G. Price, Gianfranco R. Putzolu, and Irving L. Traiger. The recovery manager of the system r database manager. *ACM Computing surveys*, 13(2):223–242, 1981.

[11] Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992.

[12] Terry Ching-Hsiang Hsu, Helge Brügner, Indrajit Roy, Kimberly Keeton, and Patrick Eugster. Nvthreads: Practical persistence for multi-threaded applications. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 468–482. ACM, 2017.

[13] Jhuyeong Jhin, Hyukjoong Kim, and Dongkun Shin. Optimizing host-level flash translation layer with considering storage stack of host systems. In *Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication*, page 75. ACM, 2018.

[14] Yanqin Jin, Hung-Wei Tseng, Yannis Papakonstantinou, and Steven Swanson. Kaml: A flexible, high-performance key-value ssd. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 373–384. IEEE, 2017.

[15] Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Gi-Hwan Oh, and Changwoo Min. X-ftl: transactional ftl for sqlite databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 97–108, 2013.

[16] Alexey Kopytov. Sysbench manual. *MySQL AB*, pages 2–3, 2012.

[17] Tirthankar Lahiri, Amit Ganesh, Ron Weiss, and Ashok Joshi. Fast-start: quick fault recovery in oracle. In *ACM SIGMOD Record*, volume 30, pages 593–598. ACM, 2001.

[18] Scott T Leutenegger and Daniel Dias. *A modeling study of the TPC-C benchmark*, volume 22. ACM, 1993.

[19] Youyou Lu, Jiwu Shu, Jia Guo, Shuai Li, and Onur Mutlu. High-performance and lightweight transaction support in flash-based ssds. *IEEE Transactions on Computers*, 64(10):2819–2832, 2015.

[20] Youyou Lu, Jiwu Shu, and Peng Zhu. Txcache: Transactional cache using byte-addressable non-volatile memories in ssds. In *2014 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 1–6. IEEE, 2014.

[21] Chen Luo and Michael J Carey. On performance stability in lsm-based storage systems. *arXiv preprint arXiv:1906.09667*, 2019.

[22] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux symposium*, volume 2, pages 21–33, 2007.

[23] C Mohan, Don Haderle, Bruce Lindsay, Hamid Pirahesh, and Peter Schwarz. Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems (TODS)*, 17(1):94–162, 1992.

[24] C Mohan, Bruce Lindsay, and Ron Obermarck. Transaction management in the r* distributed database management system. *ACM Transactions on Database Systems (TODS)*, 11(4):378–396, 1986.

[25] Gap-Joo Na, Sang-Won Lee, and Bongki Moon. Dynamic in-page logging for b+-tree index. *IEEE Transactions on Knowledge and Data Engineering*, 24(7):1231–1243, 2011.

[26] Dushyanth Narayanan and Orion Hodson. Whole-system persistence. *ACM SIGARCH Computer Architecture News*, 40(1):401–410, 2012.

[27] Sai Tung On, Jianliang Xu, Byron Choi, Haibo Hu, and Bingsheng He. Flag commit: Supporting efficient transaction recovery in flash-based dbmss. *IEEE Transactions on Knowledge and Data Engineering*, 24(9):1624–1639, 2011.

[28] Xiangyong Ouyang, David Nellans, Robert Wipfel, David Flynn, and Dhabaleswar K Panda. Beyond block i/o: Rethinking traditional storage primitives. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 301–311. IEEE, 2011.

[29] Vijayan Prabhakaran, Thomas L Rodeheffer, and Lidong Zhou. Transactional flash. In *OSDI*, volume 8, 2008.

[30] Wei Shi, Dongsheng Wang, Zhanye Wang, and Dapeng Ju. Möbius: A high performance transactional ssd with rich primitives. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–11. IEEE, 2014.

[31] Gregory Smith. *PostgreSQL 9.0: High Performance*. Packt Publishing Ltd, 2010.

[32] Yongseok Son, Jaeyoon Choi, Jekyeom Jeon, Cheolgi Min, Sunggon Kim, Heon Young Yeom, and Hyuck Han. Ssd-assisted backup and recovery for database systems. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 285–296. IEEE, 2017.

[33] Yong Ho Song, Sanghyuk Jung, Sang-Won Lee, and Jin-Soo Kim. Cosmos openssd: A pcie-based open source ssd platform. *Proc. Flash Memory Summit*, 2014.

[34] Tpc-h benchmark. http://www.tpc.org/tpch/.

[35] Jens Axboe. Fio-flexible io tester. *URL http://freecode.com/projects/fio*.

[36] Tianzheng Wang and Hideaki Kimura. Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores. *Proceedings of the VLDB Endowment*, 10(2):49–60, 2016.

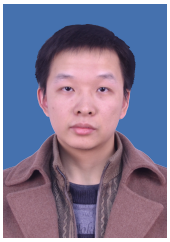
[37] Tianzheng Wang, Duo Liu, Yi Wang, and Zili Shao. Ftl 2: a hybrid flash translation layer with logging for write reduction in flash memory. In *ACM SIGPLAN Notices*, volume 48, pages 91–100. ACM, 2013.

[38] OSDL Database Test 2 (DBT-2). http://osdldbt.sourceforge.net.

[39] Charles P Wright, Richard Spillane, Gopalan Sivathanu, and Erez Zadok. Extending acid semantics to the file system. *ACM Transactions on Storage (TOS)*, 3(2):4–es, 2007.

[40] Hua Yan, Yong Huang, Xinzhi Zhou, and Yinjie Lei. An efficient and non-time-sensitive file-aware garbage collection algorithm for nand flash-based consumer electronics. *IEEE Transactions on Consumer Electronics*, 65(1):73–79, 2018.

[41] Wenting Zheng, Stephen Tu, Eddie Kohler, and Barbara Liskov. Fast databases with fast durability and recovery through multicore parallelism. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 465–477, 2014.



Yanjie Tan is a Dr. Degree candidate in the Department of Computer Science and Technology at Hunan University. He received the BS degree and the MS degree from Huazhong University of Science and Technology, China, in 2011 and 2015. His current research interests include phase change memory and flash-based storage systems.

Huailiang Tan received the BS degree from Central South University, China, in 1992, and the MS degree from Hunan University, China, in 1995, and the PhD degree from Central South University, China, in 2001. He has more than eight years of industrial R&D experience in the field of information technology. He was a visiting scholar at Virginia Commonwealth University from 2010 to 2011. He is currently a full professor of Computer Science and Technology with Hunan University, China. His research interests include high performance I/O, image and video processing, and embedded systems.

Peng Zhu is an M.S. degree candidate in the Department of Computer Science and Technology at Hunan University. He received his B.S. degree in Computer Science and Technology from Hunan Institute of Technology in 2012. His current research interests include phase change memory and flash-based storage systems.

Youyou Lu is a Ph.D. candidate in the Department of Computer Science and Technology at Tsinghua. His current research interests include non-volatile memories and file systems. He obtained his B.S. degree in Computer Science from Nanjing University in 2009. He is a student member of the IEEE.

Zaihong He received the BS degree from Hunan University, China, in 1992, and the MS degree from Hunan University, China, in 2007. She is currently a Ph.D candidate in the Department of Computer Science and Technology at Hunan University. She is also an Assistant Professor at College of Information Science and Engineering, Hunan University, China. Her research interests include big data storage system and high performance I/O.