MixSL: An Efficient Transaction Recovery Model in Flash-Based DBMS

Yulei Fan and Xiaofeng Meng

School of Information, Renmin University of China, Beijing, China
{fyl815,xfmeng}@ruc.edu.cn

Abstract. With the development of flash technologies, flash disks have become an alternative to hard disk as external storage media. Because of the unique characteristics of flash disks such as fast random read access and out-place update, shadow paging technology can be adopted to support transaction recovery in flash-based DBMS. Inspired by shadow paging and logging, we propose a new transaction commit model named MixSL which can be used in databases built on MLC flash disks. Based on MixSL, we detail normal processing, garbage collection and recovery. For improving system performance and raising the utilization ratio of flash disks, we extend MixSL to support group commit. Our performance evaluation based on the TPC-C benchmark shows that MixSL outperforms the state-of-the-art recovery protocols.

Keywords: Flash Memory, Recovery, Database, Shadow Page.

1 Introduction

With the development of semiconductor technologies, flash disks have been a competitive alternative to traditional magnetic disks as external storage media in portable devices, new-generation laptops and enterprise servers. Flash disks have the unique characteristics such as fast random access, low power consumption, high shock resistance, small dimensions and light weight[1]. In fact, single-level-cell (SLC) flash disks and multiple-level-cell (MLC) flash disks have been two popular flash devices as external storage. SLC flash disks have a favorable characteristic of partial page programming, but their capacity are usually less than MLC flash disks because of the density of single bit per cell, which leads to decrease in the unit of read, write and erase operations. This leads to high production costs, so SLC flash disks are very expensive. MLC flash disks have the preponderance of price and capacity because of lower production costs and higher density of multiple bits per cell. But the lifetime of MLC flash disks is less than SLC flash disks. In this paper, we contribute to flash-based DBMSs built on MLC flash disks by investigating how transaction recovery can be supported to improve performance and raise the utilization ratio of flash disks.

As a vital part of DBMSs, transaction recovery guarantees atomicity and durability of transactions. There are two predominant approaches to support transaction recovery for DBMSs, namely write ahead logging (WAL)[2] and

J. Wang et al. (Eds.): WAIM 2013, LNCS 7923, pp. 393–404, 2013.

shadow paging[3]. For WAL based on in-place update, in addition to undo/redo log records, we need some special log records to record significant events during transaction processing. For short transaction, WAL needs frequent write operations which are not preferable on flash disks. But group commit[3] can be used to improve the performance of transaction recovery. Different from WAL, shadow paging adopts out-of-place update style which is good suit for flash memory because of the erase-before-write limitation. In shadow paging, a page mapping table mapping page IDs to disk addresses is the key to access data, which is an indispensable component in flash translation layer(FTL)[4] of flash disks. Because of the unique characteristic of flash disks, some issues, such as maintaining the mapping between logical addresses and physical addresses, no longer exist for flash disks. These render shadow paging an appealing solution to support efficient transaction recovery on flash disks.

In this paper, based on shadow paging and logging, we propose a new commit scheme, named MixSL, for supporting efficient transaction recovery in flash-based DBMSs built on MLC flash disks. Then we detail normal processing, garbage collection and recovery. Simultaneously we extend MixSL to group commit for the system performance and the utilization ratio of MLC flash disks. On the whole, the contributions of this paper are summarized as follows:

- We propose a new commit model, called MixSL, assisting shadow paging with logging, which supports efficient transaction recovery in flash-based DBMSs built on MLC flash disks.
- Based on MixSL, we detail normal transaction processing including updating data pages, committing and aborting transactions. And then we present the garbage collection and recovery algorithms.
- We extend MixSL to support group commit for improving transaction processing performance and the utilization ratio of flash disks.
- We conduct a performance evaluation of MixSL based on the TPC-C benchmark.

The rest of this paper is organized as follows. Section 2 introduces the background and related work of our research, including flash disks and flash-based recovery approaches. In the section 3, we present MixSL and normal transaction processing, garbage collection, and recovery. Section 4 discusses how to extend MixSL to support group commit. We present the performance evaluation results of MixSL in section 5. Finally, we conclude this paper and discuss future directions in section 6.

2 Background and Related Work

In this section, we describe the relevant characteristics of MLC flash disks as well as the flash translation layer (FTL). And then we present related work about flash-based transaction recovery.

2.1 Flash Characteristics

A flash disk consists of a number of flash memory chips which are organized in many blocks. Each block is composed of a fixed number of pages. A typical page consists of 2K bytes data area and 64 bytes spare area. The spare area often stores metadata such as the error correction code (ECC) and logical block address (LBA), and so on. So, a typical block size is 128K+4K bytes (i.e., 64 pages). Erasure operations must be performed at the block level, while read and write operations are performed at the page level. There are two types of flash available in the current market: NOR and NAND. NOR flash is mainly used to store the programs. NAND flash is designed as the mass storage devices and also is the focus of this paper. Moreover, according to the number of bits stored on each cell, NAND flash can be categorized into single-level cell(SLC) and multi-level cell(MLC).

Flash disks possess a number of unique I/O characteristics[5]. First, because flash disks have no any mechanical components, the latency of a request is only linearly proportional to the amount of data transferred. Second, flash memory is subject to an erase-before-write constraint. Out-place updating is often supported through flash translation layer (FTL) for addressing the erase-before-write constraint. Third, another important characteristic of flash memory is asymmetric read, write and erasure speeds, while magnetic disks have the symmetric read and write speed and there are no erase operation for hard disks. Forth, each block can survive only a limited number of erasure operations, which means that it wears out and becomes unreliable after limited number of writes on flash disks. Typically, SLC flash supports 100K erasures per flash block, and the MLC flash only supports about 10K erasures because of the higher bit density.

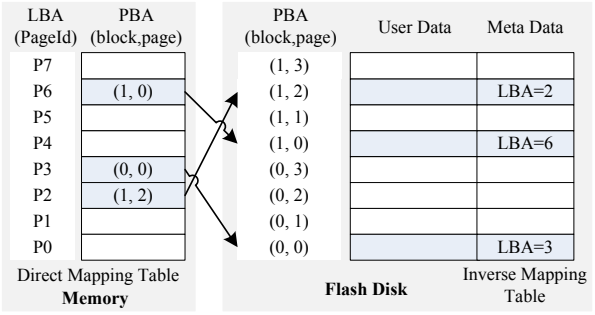

Fig. 1. FTL Mapping Table

Flash translation layer(FTL) is an important component to emulate a hard disk. FTL provides an interface to read and write flash pages and support transactional operations (page writes, commit, abort, and recovery). As shown in Figure 1, a direct mapping table in memory records the mappings between logical page addresses LBAs and physical page addresses PBAs; LBAs in the spare

areas of flash pages form an inverse mapping table, which is used to re-build the direct mapping table at boot time. The two mapping tables are crucial to implement out-place updating. Every out-of-place update operation leaves an obsolete page on the flash, so FTL maintains a free block list and has a garbage collection module to reclaim obsolete pages. Wear leveling module in FTL is used to spread writes/erasures uniformly across the entire disk space for lengthening the lifetime of flashes.

2.2 Related Work

Early works attempted to emulate the interfaces of traditional magnetic disks, so there are many work about FTL. According to mapping address granularity, FTL algorithms can be divided into four categories[6]: page-level mapping, block-level mapping, hybrid mapping with page and block granularity, and other mapping.

Recently, flash disks have been exploited to enhance the performance of file systems and database systems from various aspects. For flash-based storage management, log-based and log-structure mechanisms have been suggested to optimize random write operations by sequential write operations, such as IPL[7] and FlashStore[8], and so on. In addition, for index and buffer management, there are many new index structure[9] and new buffer replacement algorithms[10] proposed for flash devices. Simultaneously query processing techniques have also been intensively studied[11].

In contrast, not much work has been done on flash-aware transaction management. For flash-based file system, Prabhakaran et al.[12] have developed a shadow paging-based scheme, called cyclic commit. Based on this idea, they proposed two protocols, SCC and BPCC, but they are not fit for flash-based DBMSs. First, the current shadow page must be buffered until the arrival of the next shadow page for forming a cyclic linked list. Second, uncommitted pages must be frequent erased in SCC. Third, in BPCC, a complicated garbage collection mechanism need be performed. In [13], Wu proposed a fast recovery scheme for flash-based log-based file systems, so the log records are much different from log records in DBMSs. And the log records are committed into a special area on flash disks in order to read the special area during recovery.

For flash-based DBMSs built on SLC flash disks, based on shadow paging, Sai Tung On et al.[14] proposed a new flagcommit scheme for efficient transaction recovery. It addresses these issues presented in[12]. Because SLC flash disks have an unique characteristic of partial page programming, which allows a flash page to be updated a few times before an erasure becomes mandatory, so they can exploit this partial page programming feature to keep track of the transaction status. Flagcommit stores a flag about a transaction status in spare area of every flash data page. So transaction commit processing can be accomplished by updating some flags in the style of in-place. But MLC flash disks have no the unique characteristic of partial page programming. And some extra read operations need be performed for locating the data page when the transaction is aborted or committed. Moreover, during collecting the obsolete pages, some in-place operations have to be done for keeping track of the transaction status.

3 MixSL: Shadow Paging + Logging

Inspired by shadow paging and logging, we presents the new commit model, MixSL, as shown in Figure 2, which exploits out-of-place update style of Flash-based SSDs and keeps track of the transaction status by logging. And then we will detail normal transaction processing, garbage collection and recovery.

3.1 Commit Model: MixSL

The basic idea of commit model MixSL is to use shadow paging to keep track of update operations, and use logging to keep track of transactions' status. Based on MixSL, there are a link and the transaction ID in each shadow page. The link pointing to the preceding version page of the shadow page, can be used for garbage collection and transaction recovery. The transaction ID is the identifier of the transaction producing the page. However, different from SLC flash disks, MLC flash disks have no unique characteristic of partial page programming in SLC flash disks, so we must keep track of the transaction status by logging. Due to good performance of writing fixed area[5], log records related to transaction commit protocol are maintained in the fixed flash area. Log records in the fixed flash area and metadata in every shadow page can be used to undo the updates and recovery during transaction rollback or system restart.

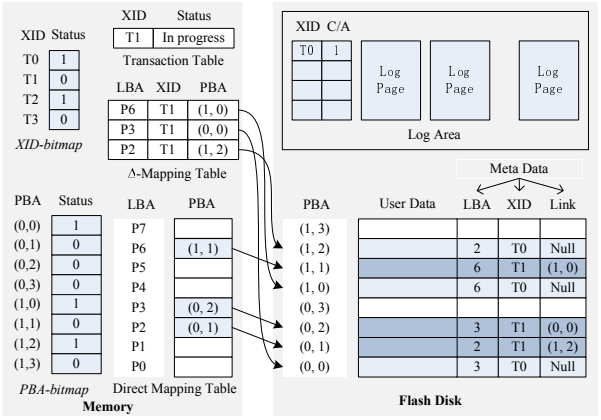

Fig. 2. Commit Model: MixSL

In traditional disk-based DBMSs, data buffer pool caches the frequently accessed disk pages of the database, and log buffer pool caches the log records produced by running transaction. The buffer management policy has an impact on the transaction recovery mechanism. In this paper, MixSL model works with steal and force polices. For log buffer management, because there are only some transaction log records but no redo/undo log records, force policy is adopted in

log buffer management. In other words, when a transaction commit, log buffer page must be flushed into the fixed flash area of flash disks. For data buffer management, we need make some assumptions. First, a page-level concurrency control protocol is adopted to handle update conflicts. Second, we used a write-back buffer: a shadow page is not created on the flash disk until the page is evicted from the buffer pool or the corresponding transaction commits. For improving the write performance of DBMSs and the lifetime of flash disks, so in section 4, we extend MixSL to support group commit.

To complete normal transaction processing, garbage collection and recovery, some data structures should be maintained in memory as shown in Figure 2. ***Transaction Table*** stores each transaction ID *XID* and its status *Status* (i.e., in progress, committed, or aborted). ***Δ-mapping Table*** stores all in-progress transactions IDs *XID* as well as the logical-to-physical mappings <*LBA, PBA*> produced by them. ***Direct mapping Table*** maintained the newest committed logical-to-physical mappings <*LBA, PBA*>. For each transaction, there is a bit in ***XID-bitmap*** (aborted transaction: '0' (default value), committed transaction: '1'). For each physical page, there is a bit in ***PBA-bitmap*** (valid page: '1' (default value), invalid page: '0'). XID-bitmap and PBA-bitmap are only built during garbage collection and recovery.

As shown in Figure 2, the whole flash disk space is divided into two regions: ***Data Area*** and ***Log Area***. Every flash data page consists of a data area and a spare area. Data area stores users' data, but spare area stores some metadata including *LBA*, *XID*, *Link*. *LBA* is the logical page ID of this physical flash data page. *XID* is the ID of the transaction *T* producing this data page. *Link* points to the preceding version data page (NULL if it is the first version data page), by which all obsolete data pages are chained together. Log area stores all log records keeping track of all transactions' final status. Every log record consists of *XID* and *C/A*. *XID* stores a transaction ID. *C/A* represents the transaction's final status ('1' if transaction is committed; otherwise, '0').

3.2 Normal Processing

Based on MixSL, we detailed normal transaction processing including updating a data page on flash disk, committing a transaction and aborting a transaction.

Update. When a transaction *T** with the ID *xid* updates a logical page *lp* with the logical address *lpa*, MixSL performs the following three steps. First, if the transaction *T** doesn't have existed in transaction table, the transaction *T** and its status *in-progress* are inserted into it. Second, we create a shadow page *pp'* on flash disks with the physical address *ppa'*, and simultaneously filling in the LBA, Link and XID fields in the spare area of *pp'* with *lpa*, *ppa'* and *xid*. Last, an entry <*lpa, T*, ppa'*> is inserted into *Δ*-mapping table. It is used to complete the transaction *T** and guarantee the correctness of direct mapping table.

Commit. When a transaction *T** with the ID *xid* commits, MixSL performs the following three steps. First, a log record <*T*, 1*> is inserted into log buffer and then flushed into fixed flash area. Second, all entries <*lpa, T*, ppa*> of

the transaction T^* in Δ-mapping table are merged into direct mapping table to replace entries <*lpa, ppa'*> with the same *lpa*. Simultaneously these entries <*lpa, T*, ppa*> of the transaction T^* need be deleted from Δ-mapping table. Last, transaction T^* must remove from transaction table.

Abort. When a transaction T^* with the ID *xid* aborts, MixSL performs the following three steps. First, a log record <T^*, *0*> is inserted into log buffer and then flushed into fixed flash area. Second, all entries same as <–, T^*, –> of the transaction T^* must be deleted from Δ-mapping table. Last, transaction T^* must remove from transaction table.

3.3 Garbage Collection

When the amount of free space on flash disks becomes lower than some preset threshold, garbage collection is triggered to reclaim the obsolete pages. The obsolete pages include three following types: (1) the uncommitted page; or (2) the aborted page; or (3) the committed out-of-date page. The transaction status becomes critical on identifying the 1st and 2nd kinds of pages, so we need create

Algorithm 1. Garbage Collection

```
begin
    Initialize_XID_Bitmap(0);
    Initialize_PBA_Bitmap(1);
    for each page in fixed log area do
        for each record in the page do
            xid = XID field of log record;
            stat = Status field of log record;
            if stat == 1 then
                Set_XID_Bitmap(xid, 1);
    for each page on flash disks do
        xid = XID field in the spare area;
        lpa = LBA field in the spare area;
        ppa = Physical address of the data page;
        prelink = Link field in the spare area;
        stat = Get_XID_Bitmap(xid);
        if stat == 0 then
            Set_PBA_Bitmap(ppa, 0);
        else
            Set_PBA_Bitmap(prelink, 0);
    for each bit in PBA_bitmap do
        flag = the bit value;
        if flag == 0 then
            ppa = Transfer_Location();
            Free_Page(ppa);
            Add_Free_Block_List(block_num);
end
```

XID-bitmap. The transaction status in XID-bitmap and *Link* field in spare area of flash pages are combined for identifying the 3rd kind of page. So the garbage collection is shown in algorithm 1.

The first step is to create and initialize XID-bitmap and PBA-bitmap with 0 and 1 respectively. The first two *For* loops set XID-bitmap by reading log records. The 3rd *For* loop reads and processes data pages on flash disks one by one. If the transaction producing a data page is uncommitted or aborted, the data page can be collected, i.e., it is invalid. So the bit corresponding to the data page in PBA-bitmap is set as 0. If the transaction is committed, we can make sure that pre-version of the data page is invalid, so the bit corresponding to *Link* in PBA-bitmap is set as 0. By now, we identify whether every data page is valid or invalid, so we can perform garbage collection for every block including invalid data pages in the last *For* loop. During collecting garbage, we need change the direct mapping table because valid data pages need be moved. At last, the erased blocks need be inserted into free block list.

3.4 Recovery

After a normal shutdown or system failure, a recovery procedure is invoked when the system restarts. It recovers the newest committed version data page and rebuilds the direct mapping table. As same as garbage collection, we need identify whether every data page on flash disks is valid or invalid. So the first three steps of recovery procedure are very like these of garbage collection as shown in algorithm 2.

The first 3 *For* loops are same as these in algorithm 1, so the status (i.e., valid or invalid) of every data page can be known by getting every bit value of PBA-bitmap. In the last *For* loop, we random read valid data pages, and then gain their *LBA* fields in spare area, and insert <*lpa, ppa*> into direct mapping table. Recovery is only related to random read, because of good random read

Algorithm 2. Recovery

```
begin
    Initialize_XID_Bitmap(0);
    Initialize_PBA_Bitmap(1);
    //The first 3 For loops are same as these in algorithm 1;
    page_num = 0;
    for each bit in PBA_bitmap do
        flag = Get_PBA_Bitmap(page_num);
        if flag == 1 then
            Random_Read_Page(page_num);
            lpa = LBA field in the spare area;
            ppa = Physical address of the data page;
            Insert_Direct_Mapping_Table(lpa, ppa);
        page_num = page_num + 1;
end
```

performance of flash memory, so the performance of recovery is only determined by the number of needed pages.

4 Extended MixSL: Group Commit

In this section, we extend MixSL to support group commit for improving the performance of flash-based databases and advancing the utilization ratio of flash space. Because a force buffer policy is adopted by log buffer, for every transaction, if it commits or aborts, we must immediately flush log buffer page into flash disk. But this may bring several problems, first, it leads to frequent write operations. Second, the flushed log page can be not full, so the flash space is dissipated. Because the unit of read and write operations is page, the log buffer page can not be smaller than one flash page. Group commit means that some transactions can be committed at the same time. So some transactions must be postponed committing or aborting. There are two protocols based on group commit. The first is to group commit fixed number of transactions, where the number can be specified by user according the buffer size and the application system. The second is to group commit dynamic number of transactions, which is more suitable to a complex application system. For the second protocol, we can group commit transactions when all the log records produced by these transactions can be fill in one log buffer page.

5 Performance Evaluation

In this section, we evaluate the performance of our proposed MixSL based on the TPC-C benchmark. Firstly, we describe the experiment setup for simulating MixSL and FlagCommit [14]. And then we compare MixSL with the CFC and AFC protocols in FlagCommit.

5.1 Experiment Setup

For showing performance, we implement a trace-driven SSD simulator which can simulate SLC SSD and MLC SSD by configuring its parameters. For comparing the performance of FlagCommit and MixSL, we configured the simulator to emulate a 32GB SLC SSD and a 32G MLC SSD with a wear-aware garbage collection. Similar to [12], 10% of the flash blocks were reserved for handling garbage collection, and the threshold to trigger the garbage collection was set to 5%. And then we implemented MixSL, CFC and AFC protocols based on FlagCommit with a Strict Two-Phase Locking (2PL) protocol. For the buffer management, LRU strategy is applied to cache previously disk pages accessed. The on-line transaction processing workload, TPC-C benchmark, is used for evaluating the performance of MixSL and FlagCommit. We generated the workload trace by executing TPC-C transactions on PostgreSQL 8.4 and recording their data access requests. For the generator, we set 50 clients and 20 data warehouses(2G

Table 1. Default Parameter Settings

Parameter	Default Setting
SLC Block/Page size	128KB/2KB
SLC Page write/read latency	0.2ms/0.08ms
SLC Partial programming latency	0.2ms
SLC Block erasure delay	1.5ms
MLC Block/Page size	512KB/8KB
MLC Page write/read latency	0.65ms/0.25ms
MLC Focus programming latency	0.5ms
MLC Block erasure delay	5ms
Logical page size	8KB
Buffer pool size	512

data). Simultaneously, the transaction abort ratio was set to 5% by default. As shown in [14], the cost of partial programming was also same as that of a page write. So we summarize the default parameter settings in table 1.

We conducted our experiments on a HP computer running Ubuntu Linux with an Intel Xeon E5620 2.40 GHz cpu. We measured the transaction throughput, transaction execution time, recovery cost, and garbage collection overhead.

5.2 Comparison with FlagCommit

We compare the proposed MixSL with CFC and AFC protocols [14].

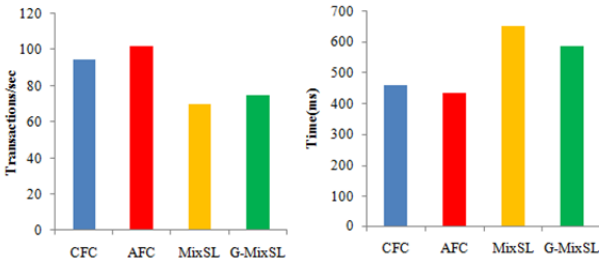

Fig. 3. Transaction Throughput and Execution Time

Transaction Throughput and Execution Time. As shown in Figure 3(a), transaction throughput of MixSL does not outperform these of CFC and AFC. Simultaneously the average running time of every transaction in MixSL is bigger than these in CFC and AFC. The main reason is that read/write latency of SLC flash is less than that of MLC flash. But the gap of transaction processing performance between SLC flash and MLC flash is less than the gap of read/write performance between them because page size of MLC is more than that of SLC.

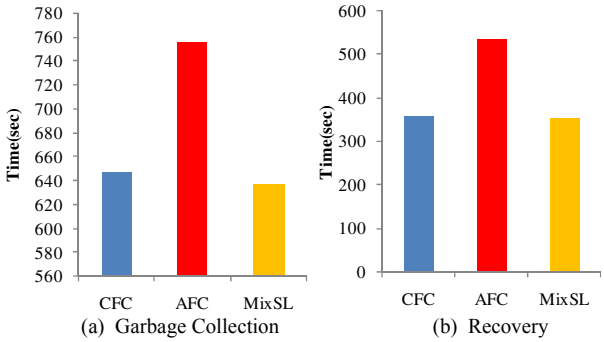

Fig. 4. Recovery and Garbage Collection

Recovery and Garbage Collection. To gain more insight, we further measure their garbage collection overhead and plot the results in Figure 4(a). MixSL outperforms AFC because garbage collection in AFC need read some pages for locating the first valid page and marks the page when the list of pages belonging one transaction is split. we can also see that MixSL is same as CFC, because of good performance of read/write of SLC flash and fast locating the page which need be marked in CFC. Figure 4(b) shows the recovery performance results which is similar with garbage collection because the part procedure of recovery is same as part of garbage collection.

Group Commit. Because of group committing transactions, flushing transaction log records into fixed area of flash memory is decreased. Group committing transactions leads to bring down the average running time of every transaction. Transaction throughput of MixSL supporting group commit (short for G-MixSL) is bigger than MixSL. So as is shown in Figure 3, the average execution time of every transaction in G-MixSL is less than that in MixSL.

6 Conclusions and Future Works

In this paper, we have proposed a new transaction commit model MixSL for database built on MLC flash memory. Our main idea is to exploit the fast random read access, out-place updating and per-page metadata to optimize the performance of transaction processing and recovery by combining shadow paging and logging technologies. Simultaneously we extend MixSL to support group commit for improving the performance of flash-based databases and advancing the utilization ratio of flash space. Our performance evaluation based on the TPC-C benchmark shows that MixSL outperforms the state-of-the-art recovery protocols. As for future work, we plan to extend MixSL to support for no-force buffer management, fine-grained concurrency control and checkpoint, and so on.

Acknowledgments. This research was partially supported by the grants from the Natural Science Foundation of China (No. 60833005, 61070055, 91024032, 91124001); the National 863 High-tech Program (No. 2012AA010701, 2013AA013204); the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University(No. 11XNL010).

References

1. Gray, J., Fitzgerald, B.: Flash disk opportunity for server applications. Queue 6(4), 18–23 (2008)
2. Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P.: ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database System 17(1), 94–162 (1992)
3. Ramakrishnan, R., Gehrke, J.: Database Management Systems. McGraw-Hill (2003)
4. Chung, T.-S., Park, D.-J., Park, S., Lee, D.-H., Lee, S.-W., Song, H.-J.: A survey of Flash Transalation Layer. Journal of Systems Architecture - Embedded Systems Design(JSA) 55(5-6), 332–343 (2009)
5. Bouganim, L., Jónsson, B.T., Bonnet, P.: uFLIP: Understanding flash IO patterns. In: Proceedings of CIDR (2009)
6. Ma, D., Feng, J., Li, G.: LazyFTL: A Page-level Flash Translation Layer Optimized for NAND Flash Memory. In: Proceedings of ACM SIGMOD 2011 (2011)
7. Lee, S.-W., Moon, B.: Design of flash-based DBMS: An in-page logging approach. In: Proceedings of ACM SIGMOD (2007)
8. Debnath, B., Sengupta, S., Li, J.: FlashStore: High throughput persistent key-value store. In: Proceedings of VLDB (2010)
9. Agrawal, D., Ganesan, D., Sitaraman, R., Diao, Y.: Lazy-adaptive tree: An optimized index structure for flash devices. In: Proceedings of VLDB (2009)
10. Ou, Y., Härder, T., Jin, P.: CFDC: a flash-aware replacement policy for database buffer management. In: Proceedings of DaMoN (2009)
11. Tsirogiannis, D., Harizopoulos, S., Shah, M.A., Wiener, J.L., Graefe, G.: Query processing techniques for solid state drives. In: Proceedings of SIGMOD (2009)
12. Prabhakaran, V., Rodeheffer, T.L., Zhou, L.: Transactional flash. In: Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2008), San Diego, CA, USA (2008)
13. Wu, C.-H., Kuo, T.-W., Chang, L.-P.: Efficient initialization and crash recovery for log-based file systems over flash memory. In: Proceedings of the ACM Symposium on Applied Computing, SAC 2006 (2006)
14. On, S.T., Xu, J., Choi, B., Hu, H., He, B.: Flag Commit: Supporting Efficient Transaction Recovery in Flash-based DBMSs. IEEE Transactions on Knowledge and Data Engineering (2011)
15. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrecy Control and Recovery in Database Systems. Addison Wesley (1987)