Research and Summary on Various Mapping Methods in FTL

Author Name
School of Computer Science and Technology
Xi'an JiaoTong University
Xi'an, China
author@stu.xjtu.edu.cn

***Abstract*—The Flash Translation Layer (FTL) is a core component of the controller in flash-based devices such as Solid State Drives (SSDs). Its main functions include managing the mapping of logical addresses to physical addresses, implementing garbage collection, and performing wear leveling. Different FTL mapping schemes have a significant impact on the performance, lifespan, and cost of flash devices. The most commonly used mapping schemes today include block-level mapping, page-level mapping, and hybrid mapping. This paper provides a systematic review of the common mapping strategies in FTL layers, delving into the basic principles, advantages, disadvantages, and applicable scenarios of each mapping method. By comparing the performance in areas such as performance optimization, write amplification control, garbage collection efficiency, and wear leveling optimization, this paper summarizes the challenges faced by existing public technologies. The aim of this paper is to provide a comprehensive summary of FTL knowledge for newcomers to the flash domain, helping them get started faster and more effectively.**

***Index Terms*—Flash Memory, Flash Translation Layer, Mapping Method, Garbage Collection, Well Levelling**

I. Introduction

Flash memory has been widely adopted in storage devices due to its non-volatile storage characteristics, but its physical properties present unique challenges: mandatory erase-before-update operations, block-level erase granularity, and limited program/erase cycles. These inherent characteristics make direct application of traditional storage management methods infeasible, thus necessitating the Flash Translation Layer (FTL) as a core control architecture. Serving as the central component of flash device controllers, FTL establishes dynamic balance among device performance, lifespan, and storage costs through address mapping management, garbage collection mechanisms, and wear-leveling algorithms.

The mainstream FTL mapping strategies currently include block-level mapping, page-level mapping, and their hybrid architectures, with different schemes demonstrating significant variations in mapping granularity, metadata overhead, and write performance. This paper systematically reviews the evolutionary trajectory of address mapping mechanisms in FTL layers, thoroughly analyzes the operational principles of various mapping strategies, and establishes a multi-dimensional evaluation framework to compare their effectiveness in performance optimization, write amplification suppression, garbage collection efficiency, and wear leveling. By revealing common challenges existing technical solutions face in metadata management efficiency, random write performance optimization, and compatibility with emerging storage media, this research aims to construct a systematic knowledge framework for flash storage researchers and provide technical references for next-generation FTL design.

Please note that sections II–III below briefly introduce FLASH memory and FTL respectively. Section IV presents the early-stage FTL designs for NOR-based devices, while sections V–VI elaborate on block-level mapping schemes and hybrid mapping schemes respectively.

II. Flash Memory Overview

Contemporary flash memory architectures are primarily categorized into NOR and NAND types. NOR flash employs separate address and data buses, enabling byte-addressable random access operations. This architectural characteristic facilitates execute-in-place (XIP) functionality for direct code execution in embedded systems and firmware storage, establishing NOR as an optimal replacement for conventional ROM. However, its substantial write/erase latencies impose limitations on storage performance. In contrast, NAND flash utilizes a shared I/O interface for address and data transmission, exclusively supporting page-level block operations. The compact cell structure of NAND achieves superior storage density, making it the preferred solution for secondary storage media (this study primarily focuses on NAND flash technology). These two architectures exhibit complementary characteristics: NOR demonstrates advantages in read speed, while NAND excels in write/erase efficiency.

A. Structural Hierarchy in NAND Flash Packaging

NAND Flash Packaging Architecture The NAND flash package comprises multiple independently operable dies that share a common I/O bus (as shown in Fig.1). Each die contains multiple parallel planes, with each plane featuring dedicated data registers. The hierarchical structure consists of planes organized into fixed numbers of blocks, which are further subdivided into predetermined quantities of pages - the fundamental unit for read/write operations. Contemporary NAND designs incorporate a reserved spare area at the end of each page for storing Out-of-Band (OOB) metadata, including:

Logical Page Number (LPN), Error Correction Code (ECC), and page status flags.

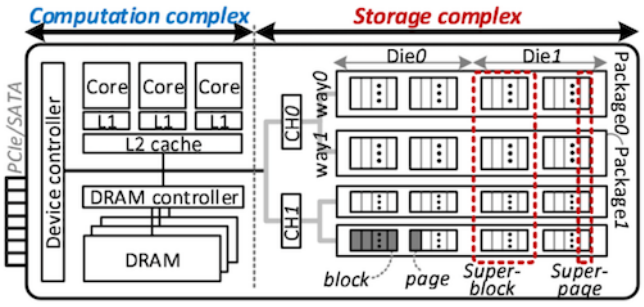

Fig. 1. SSD internal architecture.

B. Differences Between NAND and HDD

The fundamental distinctions between SSDs and HDDs originate from their storage medium characteristics. Unlike traditional storage media capable of in-place updates, flash memory devices are governed by physical write constraints: bit states in programmed pages only permit unidirectional transitions (1→0), requiring block-level erase operations to restore all-1 states. This erase granularity, significantly coarser than page-level read/write operations, necessitates full block migration during single-page updates—temporarily relocating valid pages to cache, performing block erasure, and rewriting data. This mechanism not only accelerates storage cell degradation but also introduces write amplification and potential data consistency issues. Consequently, modern flash controllers universally adopt out-of-place update strategies, directing data modifications to blank physical pages while maintaining logical address mapping.

Cell endurance represents another critical divergence. Each flash memory block experiences irreversible oxide layer degradation through electron tunneling effects after approximately 1,000 to 100,000 program/erase cycles. Manufacturers implement bad block replacement using pre-configured spare blocks, yet rely on wear-leveling algorithms to dynamically distribute erase operations, maintaining near-uniform wear across all physical blocks to extend device longevity.

Structural elimination of mechanical components fundamentally eradicates addressing latency. Without requiring physical actuator movement or rotational latency inherent to HDDs, SSDs achieve random access performance improvements spanning multiple orders of magnitude.

Finally, inherent charge manipulation characteristics create asymmetric read/write speeds. Write operations involving high-voltage-driven electron injection into floating gates demand substantially more time than read operations that merely detect charge states. This physical dichotomy remains universally observable across flash memory technologies.

C. Programming constraints

Programming constraints stem from physical characteristics of storage mediums. Single-Level Cell (SLC) permits arbitrary programming sequence within blocks, which is crucial for address mapping strategies in FTL layers requiring physical location alignment with logical addresses. In contrast, Multi-Level Cell (MLC) mandates strict sequential programming to prevent charge interference between adjacent pages caused by repeated writes. This restriction enforces programming operations to follow monotonically increasing page numbering, ensuring each page undergoes only a single programming cycle.

III. FTL Overview

FTL is a software abstraction layer built upon raw flash memory, primarily responsible for address translation, garbage collection, and wear leveling. It masks the inherent limitation of flash devices lacking in-place updates by presenting a standard block device interface. FTL implementations generally adopt two architectural paradigms:

- **Embedded Integration** Designed for resource-constrained environments, FTL is implemented as part of the file system that shares CPU resources with user applications. This approach eliminates independent mapping tables, requiring the file system to directly manage data migration processes;
- **Firmware-based Implementation** FTL resides in device firmware accessible through standardized interfaces. The firmware architecture comprises three core components: read-only memory (ROM) storing FTL code, static random-access memory (SRAM) maintaining runtime data and partial mapping tables, and a dedicated controller executing FTL logic.

In this section, we focus on presenting the core data structures of FTL, common performance metrics, and their preliminary categorization.

A. Core Data Structures of FTL

The FTL layer implements address translation through forward and reverse mapping tables. The forward mapping table converts Logical Block Numbers (LBN) to Physical Block Numbers (PBN), forming the core data infrastructure of FTL. Implementations may adopt search tree structures or simple arrays, typically residing in SRAM or flash memory.

The reverse mapping table consists of physical page/block identifiers persistently stored in flash media. Sequential scanning of physical pages/blocks reveals their corresponding logical addresses, a mechanism primarily designed for valid data identification during garbage collection and FTL state recovery after system failures.

It should be noted that direct mapping is not universally adopted across FTL implementations. Different designs employ varying storage strategies for mapping tables, with some systems retaining only critical mapping information in permanent storage.

B. Performance Evaluation Metrics

The performance metrics of FTL encompass five key dimensions:

- **Address Translation Efficiency** This metric is influenced by data structure design and storage medium characteristics. Typical implementations include: direct mapping with random lookup tables and hierarchical queries using B-tree structures. When mapping table size permits, full residency in SRAM enables rapid access; beyond storage capacity limits, flash I/O operations are required for address translation.
- **SRAM Resource Utilization** Storage capacity and architectural design jointly determine mapping table storage strategies: small-capacity devices or optimized architectures can store complete mapping tables in SRAM; large-capacity devices generally adopt tiered storage, retaining only frequently accessed metadata in SRAM while storing the remainder in flash. The latter's efficiency heavily depends on data access locality patterns.
- **Block Space Utilization** Defined as the average number of programmed pages before block erasure. This parameter directly determines erase operation frequency, consequently affecting overall device performance and endurance.
- **Garbage Collection Effectiveness** The garbage collection process involves valid page migration. Architectures employing hot-cold data separation strategies gain advantages by reducing cold data migration frequency. Collection efficiency shows significant correlation with storage medium wear-leveling characteristics.
- **Fault Tolerance Capability** Power failure resilience requires state recovery mechanisms: SRAM volatility introduces data loss risks. Primary solutions include reconstructing mapping tables during initialization, or implementing state recovery through flash-stored mapping table snapshots combined with incremental logs.

C. Classification of FTL Architectures

Early flash storage devices predominantly utilized NOR architecture, with FTL designs specifically developed for NOR's byte-addressable characteristics. This architectural foundation proved fundamentally incompatible with subsequent NAND flash technologies that became mainstream. The FTL schemes discussed in this study exclusively pertain to modern NAND flash controllers optimized for high-capacity storage devices such as SSDs.

FTL architectures are classified by address mapping granularity into five categories: page-level mapping, block-level mapping, hybrid mapping, log-structured hybrid mapping, and adaptive granularity mapping. While both NOR and NAND media support the first two basic schemes, advanced composite mechanisms (including log-structured optimization and variable-granularity designs) constitute exclusive technological advancements for NAND media.

1) Page-Level Mapping: The page mapping scheme achieves page-level address translation by establishing precise mapping relationships between logical page numbers (LPN) and physical page numbers (PPN). This approach maintains independent mapping entries for each logical page, recording address correspondences and metadata.

The core advantage lies in its hot/cold data separation strategy. Leveraging SSD's out-place update mechanism, modified data writes to new physical pages while marking original pages as invalid. During garbage collection (GC), blocks containing numerous invalid pages (hot data) exhibit low valid page ratios, enabling rapid recycling with minimal data migration. Conversely, blocks with concentrated valid pages (cold data) demonstrate high stability, allowing delayed recycling to extend lifespan. This dynamic separation mechanism reduces erase cycles while effectively mitigating write amplification.

The primary limitation stems from SRAM resource consumption. For a 1TB SSD with 4KB page size, both LPN and PPN require 28-bit address space, resulting in a $2^{28} \times 4B = 1GB$ mapping table. Combined with data migration buffers, SRAM consumption significantly impacts power and cost efficiency. Some optimized implementations store complete mapping tables in flash memory, employing dynamic loading of frequently accessed entries into SRAM as a compromise solution.

2) Block-Level Mapping: The block mapping scheme employs a structured address translation mechanism that decomposes logical page numbers (LPN) into logical block numbers (LBN) and intra-block offsets. Its core workflow involves two-stage address translation: initially mapping LBN to physical block numbers (PBN), followed by locating specific page addresses within target physical blocks and their associated replacement block groups through a replacement block management mechanism. While demonstrating notable resource efficiency advantages, this scheme exhibits weak data temperature awareness capabilities, making it difficult to implement effective hot/cold data separation strategies.

3) Hybrid Mapping: Hybrid mapping FTL strategically integrates page-level and block-level mapping mechanisms to leverage their complementary advantages. This architecture primarily manifests in two implementation paradigms: The first employs dual-granularity parallel management, dynamically selecting mapping granularity based on data access patterns (such as frequency or locality). The second establishes hierarchical storage structures, utilizing fine-grained mapping zones as update buffers for coarse-grained regions (typically implemented through log-structured designs), with periodic consolidation of modified data from fine-grained to coarse-grained areas. In log-structured implementations, page-mapped log block regions capture real-time updates while block-mapped data block regions maintain baseline data versions.

This technical framework also supports dynamic mapping granularity adjustment. By creating variable-length mapping units (which may cross traditional block boundaries) and organizing mapping tables through search tree structures, it enables intelligent storage resource allocation optimized for real-time I/O patterns. However, this flexible design requires sophisticated data localization algorithms for effective operation.

IV. FTL Design for Early NOR Devices

A. SRAM-based FTL Implementation

For NOR devices characterized by low storage density and limited capacity, systems predominantly employed flexible page-level mapping schemes that stored address translation tables in SRAM. While the SRAM storage overhead remained within acceptable limits, these implementations faced SRAM reliability challenges and volatility risks during power loss. Early studies [1] developed two principal solutions: The first integrated backup battery modules to ensure data migration from SRAM to non-volatile flash during power outages. The second reconstructed mapping tables through full-device page scanning during system reboots. While this process introduced boot latency, the compact capacity of NOR devices typically maintained such delays within tolerable thresholds.

B. CAM-based FTL Implementation

This technical approach [2] employs non-volatile Content-Addressable Memory (CAM) to store mapping tables, where each entry consists of four fields: usage flag (used=1), validity flag (invalid=1), logical page number (LPN), and physical page number (PPN). During query operations, the usage and validity flags are concatenated with the target LPN to form a composite search key. When updating a page, the system writes address information into a newly allocated mapping entry while invalidating the original entry. This design inherently avoids the write-in-place limitation associated with flash-stored mapping tables.

The scheme exhibits two critical limitations: First, multiple valid mapping entries may coexist for a single logical unit, resulting in CAM storage requirements that scale linearly with flash memory capacity. Second, the implementation demands intricate parallel comparison circuits within CAM cells, substantially compromising chip integration density.

C. Standard NOR-based FTL Implementation

This architecture [3], [4] employs a two-level page table structure where each logical page maintains independent mapping entries. The complete mapping table resides in NOR flash, partitioned into mapping pages according to physical page size. A secondary indexing structure in SRAM tracks physical locations of mapping pages, with single index entries associating multiple mapping pages to overcome flash's write-in-place limitation. Frequently accessed mapping pages are cached in SRAM based on locality principles to accelerate address translation.

The address translation involves two-phase operations: First, quotient-remainder computation decomposes the logical page number, using the quotient to retrieve mapping page's physical address from the secondary index. The intermediate address is generated by combining this physical address with the remainder. Subsequently, this intermediate address splits into block number and page offset, completing final translation through block-level mapping. This hierarchical design enhances scalability while enabling in-place migration of valid pages during garbage collection - only block mapping requires updates to maintain address validity.

A critical consistency challenge emerges when page updates trigger mapping modifications. The coexistence of dirty mapping pages in SRAM and persistent versions in flash introduces synchronization risks. Delayed write-back strategies risk data loss during power failure, while immediate synchronization causes write amplification factor (WAF) of 2 due to dual writes (data page + mapping page).

The solution implements synchronous dual-write mechanism: Leveraging NOR's byte-addressable capability, update operations create replacement page chains. Each mapping page contains multiple replacement pages forming a linked-list, with new entries written to the first available position. Chain exhaustion triggers merge operations that consolidate valid entries into new mapping pages. The design keeps the write amplification factor at a low level while ensuring data integrity in power loss scenarios.

Notably, this approach fundamentally relies on NOR's byte-addressable characteristics, making it non-transferable to block-oriented NAND devices.

D. Block-level Mapping Scheme: In-block Logging

This design [5] implements an intra-block logging mechanism by allocating dedicated log areas within each physical block. When the log area reaches full capacity, it initiates a consolidation process that migrates valid pages to a new block through garbage collection cycle, reconstructing data placement according to predefined offset rules. The FTL executes two-stage address resolution: first-stage mapping converts logical page numbers (LPN) to physical block numbers (PBN), followed by second-stage validation that checks page validity at corresponding intra-block offsets. If invalidated, subsequent log area lookup completes the physical page addressing.

The scheme exhibits two critical limitations: It reduces physical address space utilization efficiency by reserving block capacity for logging purposes, and complicates address translation processes by invalidating straightforward LPN-to-offset conversion through simple modular arithmetic operations.

E. Block-level Mapping Scheme: Migratory Block

This approach [6] optimizes for random update patterns by automatically allocating replacement blocks upon detecting page-level modifications within physical blocks. Each logical block maintains a dedicated bitmap structure that dynamically tracks valid page locations between original and replacement blocks through binary flags. The architecture features an integrated update pattern detection mechanism: when the dynamic page update pattern detection module identifies random write operations (characterized by multiple overwrites within the same physical block), or when replacement block capacity reaches predefined thresholds, the system initiates data migration. This process consolidates valid pages from the original block into the replacement block, followed by original block erasure.

F. Two-phase Translation Architecture

This architecture [7] implements hierarchical address translation through two distinct mapping phases. The first stage establishes Logical Block Number (LBN) associations via page mapping table lookups using Logical Page Numbers (LPNs), while the second stage completes physical address resolution through block mapping table access. Diverging from conventional block mapping designs, LBN derivation relies on table lookup rather than arithmetic computation. To resolve potential offset collision issues where multiple LPNs map to identical block offsets, each physical block contains an allocation mapping table recording actual page offset-LPN correlations.

The address resolution workflow comprises two critical steps: initial LBN retrieval via page mapping table query, followed by Physical Block Number (PBN) determination through block mapping table access. Post physical block localization, sequential scanning of the internal allocation mapping table enables precise page positioning. During garbage collection cycles, only block mapping table updates are required, ensuring page mapping table stability and significantly reducing metadata maintenance overhead. Although employing page-level mapping mechanisms, this scheme remains classified as block-level mapping due to its dynamic intra-block page allocation capability and block-granular metadata management. The key distinction lies in address translation flexibility: conventional schemes compute page positions through fixed offset formulas, whereas this architecture enables dynamic page placement via block-local mapping tables.

V. Block-Level Mapping Scheme

This section introduces NAND flash-oriented FTL designs, collectively referred to as the NAND Flash Translation Layer (NFTL), which encompasses multiple architectural variants based on implementation specifics.

A. NFTL: Log Block Architecture

This FTL scheme [8], specifically designed for NAND flash characteristics with page-level spare area support, has become an industry-standard implementation. The core operational mechanism comprises three critical processes:

During initial write operations, data is directly written to pre-determined physical page offsets within the main data block. When overwrite operations occur, the system dynamically allocates new log blocks and sequentially writes updated pages from the log block's starting position.

The read request handling employs a reverse traversal mechanism: NFTL searches for target LPNs from the log block's end towards its start. If a match is found, the physical page data is immediately returned; otherwise, the main data block is queried. This reverse retrieval strategy leverages the sequential write characteristics of flash memory, ensuring that higher-offset pages always represent the latest versions while lower-offset and main data block pages are automatically invalidated.

The log block maintenance mechanism achieves efficient traversal through NAND's spare area fast-indexing capability. Consolidation operations are triggered under any of the following conditions: (1) log block storage exhaustion, (2) garbage collection selection as a victim block, or (3) wear-leveling requirements. The consolidation process migrates valid pages from both the log block and associated data block to a new block, followed by erasure of the original blocks. Notably, when all pages in the log block exhibit identical logical and physical offsets, an optimized consolidation is performed by simply updating the block mapping table entry to the log block's PBN and erasing only the original data block.

B. NFTL: Replacement Block Chain

This method addresses NAND devices lacking fast-search spare areas by developing an enhanced lookup mechanism that avoids full-block traversal during read operations. The replacement block chain approach organizes spare blocks in linked structures for each data block.

During initial page programming, data writes target primary blocks. Update operations redirect modified pages to the first available slot at matching offsets within the replacement chain. Read operations retrieve the most recent valid page from the chain's tail position sharing the requested logical page number (LPN)'s block offset.

Diverging from NOR-based FTL implementations, this design triggers new block allocation when no chain slots match the required offset, appending blocks until either reaching free block thresholds or experiencing chain-induced latency degradation. Garbage collection activates when chain length exceeds operational limits, migrating valid pages from victim blocks to the chain terminal before erasure.

While incompatible with SSDs enforcing strict in-block sequential programming due to non-linear offset patterns in replacement chains, this method adapts effectively through log-structured modifications. By channeling all writes to dedicated log blocks that maintain sequential compliance, the design achieves compatibility with manufacturer-specific constraints.

C. State Transition FTL

STAFF (State Transition Applied Fast Flash Translation Layer) [9] implements efficient management through five block states: Free (F) for unused blocks; Main (M) for blocks with aligned logical-physical offsets and remaining capacity; Nested (N) for log blocks with offset misalignment; Sequential (S) for fully programmed aligned blocks; Obsolete (O) for erase-pending blocks. This scheme corresponds to replacement blocks and log blocks in traditional NFTL as M-state and N-state respectively.

- M-state: Main block where all pages reside at their original offsets with remaining free space
- N-state: Non-sequential block containing pages not at original offsets (used as log block)
- S-state: Sequential block with no free space
- O-state: Obsolete block requiring erasure

M-state blocks enable direct page addressing, while N-state blocks require traversal searches. Newly allocated blocks maintain M-state, with write operations prioritizing this state

preservation. Blocks transition to N-state when update writes cause offset misalignment. The system reorganizes N-state blocks through GC processes, still striving to maintain logical offset alignment even in this state.

State transitions are governed by two merge mechanisms: Swap operation allocates new M-state blocks when M-state blocks reach full capacity (converting to S-state), and Smart Merge reorganizes valid data from full N-state blocks into new M/S-state blocks. Both merge operations are explicitly triggered when free blocks fall below a threshold.

This FTL adopts a dual-PBN mapping mechanism per LBN, typically maintaining M-S state pairs. When writes occur to an LBN already mapped to an S-state block, the Swap operation efficiently replaces the obsolete S-state block by converting the original M-state block to the new S-state and allocating a fresh M-state block, eliminating data migration overhead.

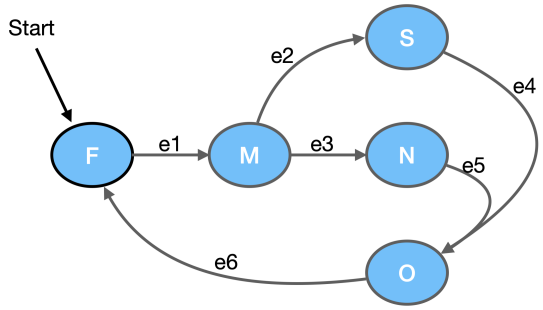

Fig. 2. STAFF state transition automaton.

D. Short Summary

Block mapping schemes effectively reduce SRAM utilization but inherently mix cold and hot data within blocks. During garbage collection, invalid pages generated by hot data trigger frequent GC operations, while valid pages (cold data) in the same block require compulsory migration to new blocks. This forced data movement not only degrades GC efficiency but also introduces redundant write operations, exacerbating write amplification and accelerating storage wear-out.

VI. Hybrid Mapping Schemes

Hybrid mapping schemes combine the low SRAM overhead of block mapping with the operational flexibility of page mapping. The primary research objective focuses on maintaining the block mapping framework while achieving effective segregation of hot and cold data.

A. Adaptive FTL

Adaptive FTL [10] partitions flash memory into two distinct zones. A smaller zone employs page-level mapping with complete metadata stored in SRAM for accelerated lookup operations, while the remaining capacity utilizes NFTL-based block-level mapping. Crucially, NFTL maintains full address space coverage, whereas the page-level mapping table selectively tracks specific addresses. This architecture necessitates consulting the page-level mapping table first during logical page number (LPN) accesses.

The zone switching mechanism operates as follows: When an NFTL log block reaches capacity, AFTL reclassifies it as hot data and migrates it to the page-mapped zone. Conversely, page-mapped zone overflow triggers eviction of least-recently-used pages back to NFTL-managed space. Since NFTL inherently spans the entire address space, such evictions simplify to standard page write operations.

B. HFTL

HFTL [11] implements hash-based hot data identification through a three-stage workflow. Each write request undergoes thermal detection via Bloom-filter-enhanced analysis. Confirmed hot pages route to page-mapped storage, while others enter coarse-grained mapping zones.

The thermal detection engine employs parallel hash functions to map each LPN across multiple reference counters. Each write operation increments corresponding counters, enabling real-time heat assessment by comparing counts against dynamic thresholds. This multi-hash design intentionally introduces controlled redundancy to minimize false positive identifications.

As an optimization over conventional hybrid mapping systems, HFTL demonstrates efficiency under stable access patterns with concentrated hot pages. However, dynamic workload fluctuations necessitate hot page evictions to coarse-grained zones, incurring non-trivial metadata maintenance overhead during access pattern transitions.

C. Log-Structured Hybrid Mapping Schemes

Log-structured hybrid mapping schemes partition flash memory into two primary areas: the Data Block Area (DBA) and Log Block Area (LBA). The DBA employs block-level mapping and constitutes the majority of flash capacity, while the LBA serves as a compact update buffer for DBA. Log blocks in LBA periodically merge with corresponding DBA blocks to free space for subsequent updates.

Three fundamental merge operations exist:

Full merge copies all valid data from both original data blocks and log blocks sharing the same logical block number into newly allocated blocks, subsequently erasing the original blocks. This method incurs substantial overhead.

- Partial merge occurs when all pages in a log block maintain correct offsets but the block remains partially filled. This process transfers remaining valid pages from the data block to the log block, then replaces the original data block with the updated log block.
- Switch merge activates when a log block contains fully sequential pages with correct offsets. The fully populated log block directly replaces its corresponding data block without data migration.
- Reducing merge operation overhead constitutes the principal design challenge for efficient log-structured hybrid mapping implementations.

D. Block-Associative Sector Translation

As the first log-structured hybrid FTL scheme, Block-Associative Sector Translation (BAST) [12] achieves notable read performance improvement by storing mapping information in SRAM. However, this design exhibits inherent limitations under random access patterns: hot pages with frequent updates rapidly fill their dedicated log blocks, while the non-sharable log block architecture intensifies this issue, triggering frequent merge operations. The exclusive binding mechanism leads to low utilization of cold data's log blocks, resulting in inefficient storage occupation and critical shortage of free blocks. When hot data requires new log blocks, the system is forced to reclaim partially filled log blocks and initiate cold block merges, creating a vicious cycle of block thrashing that degrades performance and accelerates flash wear-out.

The scheme's sequential page allocation policy, which mandates writing to the first available page in log blocks, fundamentally prevents efficient implementation of switch merges or partial merges, thereby exacerbating performance penalties.

E. Fully Associative Sector Translation

The Fully Associative Sector Translation (FAST) [13], [14] fundamentally differs from BAST by permitting random updates to populate any log block, with new blocks allocated upon saturation. This design effectively eliminates the block thrashing issue inherent to BAST.

While FAST enhances block utilization and reduces garbage collection frequency, it substantially increases log block recycling complexity. Since any data block can share log blocks, recycling a single log block may involve as many data blocks as pages contained within it, a characteristic termed the "full association" mechanism. The association degree, quantified by page count within log blocks, directly determines recycling overhead.

To leverage switch merge and partial merge mechanisms, FAST employs dedicated sequential log blocks for ordered updates. However, modern systems' high parallelism induces interleaved write streams from concurrent processes, overwhelming single sequential log blocks.

Research proposes a "second chance" strategy during garbage collection: retaining valid pages from victim blocks in LBA's tail region (following FAST's FIFO policy) instead of immediate merging, anticipating subsequent updates to invalidate them. This strategy's efficacy depends on hot data proportion within LBA - when random updates dominate, most valid pages remain active, making tail migration counterproductive.

F. Superblock FTL

In traditional log-based FTL schemes, BAST suffers from overly strict block associations leading to low log block utilization, while FAST's loose block associations result in excessive GC overhead. Superblock FTL [15] optimizes block association through balanced design.

This scheme employs block clustering, organizing N logically contiguous data blocks into D-Blocks groups that share K update blocks (U-Blocks), collectively forming a superblock. This structure effectively exploits spatial locality characteristics, aligning with modern operating systems' preference for contiguous logical address allocation.

Different from other hybrid FTLs, Superblock FTL implements a three-tiered mapping architecture: the primary block mapping table resides in SRAM, while secondary and tertiary page-level mapping information is stored in superblocks' OOB areas. Although this design enhances log block recycling efficiency, the multi-layer mapping structure increases implementation complexity, and OOB storage mechanisms constrain system flexibility.

VII. Conclusion

As an emerging storage medium, flash memory has gained rapid market adoption due to its high bandwidth and low latency, particularly for its significant improvements in random access performance compared to HDDs. The inherent limitation of in-place update prohibition necessitates flash translation layer (FTL) as a core component embedded in firmware or system software, which also handles critical functions including wear leveling and garbage collection management.

Given that SSD technologies remain proprietary assets of major semiconductor manufacturers, this paper systematically summarizes fundamental research achievements from patents, conference papers, and journal publications. The survey covers three principal mapping architectures: page-level mapping originating from NOR devices, block-level mapping widely implemented in NAND flash, and hybrid mapping schemes. This comprehensive review aims to provide foundational references for researchers entering the field of flash memory studies.

References

[1] M. Wu and W. Zwaenepoel, "eNVy: a non-volatile, main memory storage system," ACM SIGOPS Operating Systems Review, vol. 28, no. 5, pp. 86–97, 1994.

[2] M. Assar, S. Nemazie, and P. Estakhri, "Flash memory mass storage architecture," U.S. Patent No. 5,388,083, Feb. 1995.

[3] A. Ban, "Flash file system," U.S. Patent No. 5,404,485, Apr. 1995.

[4] Intel Corporation, "Understanding the flash translation layer (FTL) specification," Santa Clara, CA: Intel Corporation, Technical Report: AP-864, Dec. 1998.

[5] T. Shinohara, "Flash memory card with block memory address arrangement," U.S. Patent No. 5,905,993, May 1999.

[6] P. Estakhri and I. Berhanu, "Moving sequential sectors within a block of information in a flash memory mass storage architecture," U.S. Patent No. 5,930,815, Jul. 1999.

[7] B.-S. Kim and G.-Y. Lee, "Method of driving remapping in flash memory and flash memory architecture suitable therefor," U.S. Patent No. 6,381,176 B1, Apr. 2002.

[8] Micron Technology, Inc., "NAND flash translation layer (NFTL) 4.6.0. NFTL User Guide Rev. L," Boise, ID: Micron Technology, Inc., Feb. 2011.

[9] T.-S. Chung, S. Park, M.-J. Jung, et al., "STAFF: State transition applied fast flash translation layer," in Proc. Int. Conf. Architecture of Computing Systems, Heidelberg, Germany, 2004, pp. 199–212.

[10] C.-H. Wu and T.-W. Kuo, "An adaptive two-level management for the flash translation layer in embedded systems," in Proc. 2006 IEEE/ACM Int. Conf. Computer-Aided Design, San Jose, CA, 2006, pp. 601–606.

[11] H.-S. Lee, H.-S. Yun, and D.-H. Lee, "HFTL: Hybrid flash translation layer based on hot data identification for flash memory," IEEE Trans. Consumer Electronics, vol. 55, no. 4, pp. 2005–2011, 2009.

[12] B.-S. Kim and G.-Y. Lee, "Method of driving remapping in flash memory and flash memory architecture suitable therefor," U.S. Patent No. 6,381,176 B1, Apr. 2002.
[13] S.-W. Lee and B. Moon, "Design of flash-based DBMS: An in-page logging approach," in Proc. 2007 ACM SIGMOD Int. Conf. Management of Data, Beijing, China, 2007, pp. 55–66.
[14] S.-W. Lee, D.-J. Park, T.-S. Chung, et al., "A log buffer based flash translation layer using fully associative sector translation," ACM Trans. Embedded Comput. Syst., vol. 6, no. 3, 2007.
[15] J.-U. Kang, H. Jo, J.-S. Kim, et al., "A superblock-based flash translation layer for NAND flash memory," in Proc. 6th ACM & IEEE Int. Conf. Embedded Software, San Francisco, CA, 2006, pp. 161–170.