Efficient Hardware Redo Logging for Secure Persistent Memory

Zhan Zhang*, Jianhui Yue†, Xiaofei Liao*, Hai Jin*
**National Engineering Research Center for Big Data Technology and System,*
Services Computing Technology and System Lab, Cluster and Grid Computing Lab,
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
†*Computer Science Department, Michigan Technological University, Houghton, Michigan*
{*zhangzhan, xfliao, hjin*}@hust.edu.cn, *jyue*@mtu.edu

Abstract—**Write-ahead log and data encryption technologies are employed to ensure both crash consistency and data security for *persistent memory* (PM). The encryption/decryption of both data and log requests increase the memory request latency, degrading the system performance. To tackle this issue, this paper proposes a novel log-aware memory encryption scheme to reduce encryption/decryption operations, without compromising data security. Specifically, redo log blocks are stored as ciphertext for their corresponding data blocks to avoid encryption/decryption of log data blocks when in-place update operations are applied to them. We further design a compact log record layout with fewer encryption metadata in log records to reduce logging traffic. Our simulation results show that the transaction throughput of the proposed design outperforms the baseline design and the state-of-the-art design by 87.3% and 55.3% on average, respectively.**

Index Terms—**Persistent Memory, Crash Consistency, Computer Architecture**

I. Introduction

Persistent memory (PM)'s non-volatility, large capacity, fast speed, and byte-addressability make it become a promising candidate to reduce the gap between main memory and traditional external storage. When PM is widely deployed, data crash consistency and data security should be guaranteed. Data crash consistency means that all data should be recovered to a consistent state upon a system crash. The non-volatility of PM enables attackers to access data stored in it if PM is stolen. To ensure data security, an encryption engine is deployed in the commodity PM controller [1]. However, it is challenging to simultaneously ensure data crash consistency and data security for PM without significant performance overhead.

Crash consistency requires that all data modifications within a transaction are persisted to PM always in a nothing-or-all manner, even upon a system crash. Traditional systems adopt write-ahead logging, such as undo logging, redo logging, or a combination of both, to guarantee crash consistency. A transaction update is applied to in-place data only after its log of modification is stored in PM. The ordering constraint between logging and in-place update makes the logging method suffer from inferior performance because their execution is on the critical path. In addition, software logging methods not only put more burden on programmers but also potentially introduce bugs. To address those limitations, hardware-assisted logging methods [2]–[5] have been proposed to move the logging out of the critical path and mitigate programming burden. Since redo logging is superior to undo logging [5], we consider hardware-assisted redo logging in this paper.

Since PM is non-volatile, attackers can easily access data stored in PM if a powered-off PM device is stolen. Counter mode encryption encrypts data stored in PM to make data secure and hides data decryption latency [6]. Writing encrypted data blocks to PM needs update corresponding counters. The maintenance of crash consistency [6] requires that a modified data block and its counter are atomically persisted to PM. Otherwise, an inconsistent counter fails to decrypt the encrypted data.

It is challenging to achieve crash consistency and data security simultaneously, without suffering significant performance loss. Firstly, counter mode encryption can effectively hide decryption latency but still exposes the encryption latency to writing data blocks. In the conventional system, write operations typically are not on the critical path, and the exposed encryption latency does not hurt system performance much. However, the redo logging causes the writing log entry operations to be on the critical path. This is because log operations for an active transaction must be completed before proceeding to the next transaction. Secondly, maintaining counter update atomicity introduces updated counters to log operations, exacerbating the slow PM bandwidth issue.

To address the issues above, we propose the efficient hardware-assisted redo logging design for encrypted PM. To mitigate the encryption latency impact, we design a novel log-aware memory encryption method that removes a significant portion of data encryption and decryption operations without compromising data security. Specifically, we propose to encrypt a logged data block in a log entry with the security metadata for its home address, rather than with the security metadata for the logged data block address. In this way, we can avoid a decryption operation and an encryption operation when executing an in-place update for a log entry. The reduced operations on an encryption engine can significantly decrease

This work is supported by National Natural Science Foundation of China under grants No.61825202, 61732010, 61929103, 62072198, and USA NSF 1745748.

data encryption latency, improving system performance. To minimize PM write operations caused by counters in log entries, we propose a new log record layout that collates partial counters and address metadata into a log record's header. The proposed method avoids storing a full counter block, which is as large as the data payload in log entries, greatly decreasing the write operations for log entries. However, the introduced log record layout increases the counter cache miss latency. To address the aforementioned latency issue, we introduce the counter buffer and the counter-mapping table to reduce counter cache miss latency, which is critical for PM read and write operations, to improve system performance further. We evaluate our designs and the simulation results demonstrate that our proposed designs can collectively improve transaction throughput by 87.3% and 55.3% over the baseline system and the state-of-the-art design respectively.

Overall, this paper makes the following contributions:

- It proposes the log-aware memory encryption scheme that reduces a significant portion of encryption and decryption operations and minimizes data encryption latency, which is exposed to the critical path in the transaction system with log operations.
- It proposes the compact log record to decrease a large portion of write operations for counter logging, efficiently using PM write bandwidth.
- It reduces the counter cache miss latency by introducing the counter buffer and the counter-mapping table.

We organize the rest of the paper as follows. We discuss PM crash consistency and memory encryption in Section II. We present our design and evaluation in Section III and IV. We summarize related work in Section V and conclude this paper in Section VI.

II. Background

A. PM Crash Consistency

Crash consistency ensures that data on PM can be recovered into a consistent state after the system crash or power loss. Prior studies [2], [3], [5], [7] have proposed multiple logging schemes to achieve crash consistency. To support logging operations, PM is divided into the home region and the log region. The home region is visible to programs, the log region is only visible to the memory controller in the hardware-assisted logging system, which generates and issues log requests. While a data block's address in the home region is referred to as the home address, the address for a logged data block in the log region is referred to as the logged data block address. These logging schemes fall into two categories: undo logging and redo logging, according to log content. The undo log entries store the original data contents before modified data is written to the home region, redo entries store modified data contents before modified data is updated to the home region. Undo logging scheme can start the next transaction after data modifications are applied to their home locations. Redo logging scheme can proceed to the next transaction only after the current transaction's log entries are written to PM, which is referred to as committing a transaction.

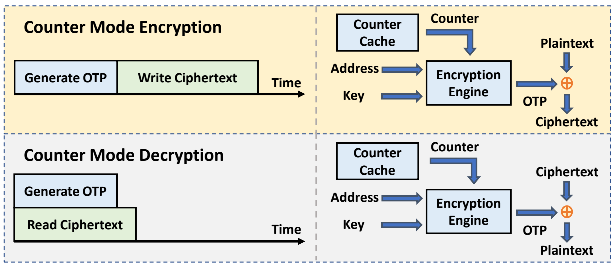

Fig. 1. Encryption and Decryption in Counter Mode

As a result, speeding up the logging operations can improve the transaction throughput under the redo logging scheme.

To optimize PM write bandwidth, Ref. [2] proposes a log organization scheme where seven 64B logged data blocks share a 64B header block. The header contains home addresses for these seven logged data blocks. Seven logged data blocks and their header are referred to as a log record. This log organization realizes the high write efficiency since its payload ratio is $7/8$.

B. Data Block Encryption

Due to PM's non-volatility, data is still accessible in powered-off PM. Attackers who steal the powered-off PM device can stream out data from it, causing security issues. Data blocks in cache hierarchy are plaintext since they are in the security domain. To ensure the off-chip data security, data must be encrypted and the corresponding ciphertext is persisted to PM. Fig. 1 illustrates how the counter mode encryption/decryption works. Before encrypting a data block, an encryption engine generates a *one-time-pad* (OTP) with the key, the data block address, and its counter. The corresponding ciphertext is generated by XORing the data block with the OTP. To decrypt a ciphertext, the encryption engine produces the OTP with the key, the data block address, and its counter, and then recovers the plaintext by XORing the ciphertext with the OTP. Every update to a data block will increase its counter. 40ns data decryption latency [6] increases memory read latency, significantly degrading performance. Therefore, the counter cache is proposed to reduce data decryption latency [8]. When a counter hits the counter cache, the counter mode encryption can overlap the computation of OTP with the reading encrypted data from PM, reducing memory read latency as shown in Fig. 1. Note that Intel's SGX [9] only encrypts the volatile memory using the counter mode encryption, without ensuring crash consistency for PM. However, this paper targets the hardware-assisted secure PM to guarantee crash consistency.

A data block can be written to secure PM only after being encrypted. Accordingly, the counter mode encryption does not hide the data encryption latency for memory write operations, which is not an issue for conventional workloads whose write operations are off the critical path. However, write latency is on the critical path for workloads with log operations. For example, the redo logging requires that log write operations for an active transaction are done before starting the next transaction. Therefore, it is critical to reduce the memory

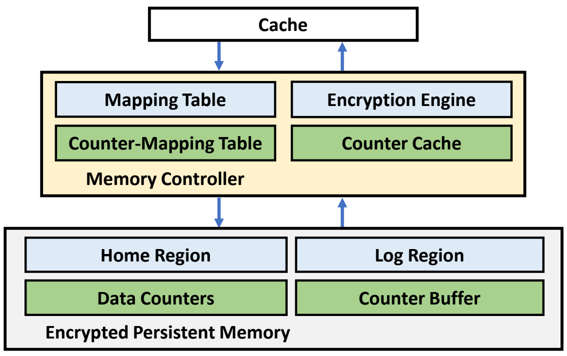

Fig. 2. Architecture Overview

encryption latency to improve system performance. However, it is challenging to deploy multiple encryption engines in a memory controller due to encryption engine circuit overhead. In this work, we propose a novel log-aware memory encryption scheme to minimize operations on the encryption engine and decrease encryption latency.

Persisting an encrypted data block requires its counter also to be persisted, so that the data can be correctly decrypted with the consistent counter after a system crash. To support the counter update atomicity [6], a log entry contains an encrypted data block, its home address, and its corresponding counter block. A data block's address and its counter together are referred to as its security metadata. Since the counter cache manages the counters in the granularity of 64B [6], the counter block component in a log entry is also 64B. This log entry scheme realizes the low payload ratio, $64/(64+8+64)$, where a data block, its address, and its counter block are 64B, 8B, and 64B, respectively. The low payload ratio not only wastes PM write bandwidth but also hurts the PM lifespan. In this paper, we propose an efficient log record layout to reduce counters' logging overhead and increase its payload ratio.

III. Design

A. Design Overview

We present our system architecture in Fig. 2. The secure memory system encrypts data blocks stored in the persistent memory with the counter mode encryption, leaving the plaintext data blocks in the cache. A 64B data block is encrypted with its dedicated 8B counter, which is buffered in the counter cache inside the memory controller to speed up data decryption. Each counter cache entry is 64B and stores eight counters for eight adjacent 64B data blocks, same as [6].

To support crash consistency, we use the transaction mechanism that allows programmers to use *tx_begin* and *tx_end* to indicate the scope of a transaction which requires crash consistency. The memory controller generates redo log entries for modifications of data blocks and their counters occurring in a transaction, and these log entries are encrypted before being written to the log region in PM. To mitigate the encryption overhead, we present our proposed log-aware memory encryption scheme in Sec. III-B. To reduce the security metadata overhead, we describe the compact log record in Sec. III-C.

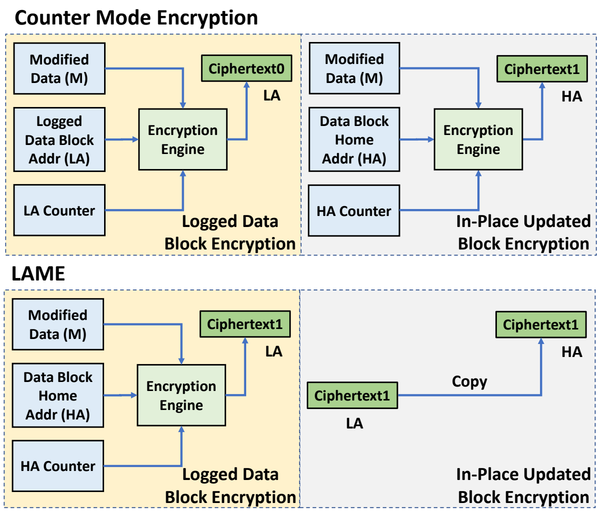

Fig. 3. Compare LAME with Counter Mode Encryption Scheme

After log entries are written to the log region, the mapping table is updated to indicate that the latest version of the modified blocks is in the log region and redirects read requests of these blocks to the log region. Upon a counter cache miss, the memory controller may need to read multiple counters in the log region, introducing performance overhead. To address this issue, we present the counter-mapping table and counter buffer in Sec. III-D. The memory controller asynchronously performs in-place update in background. Specifically, it applies the committed log entries to the home region, reclaims the log memory space, and updates the mapping table and the counter-mapping table, which is detailed in Sec. III-E.

B. Log-Aware Memory Encryption

In the secure redo logging, a transaction can proceed to the next transaction only after the on-going transaction's log entries are all encrypted and written to PM. Counter mode encryption fails to hide encryption latency for writing log entries and exposes it to the critical path of transaction execution, leading to higher transaction committing latency and lower transaction throughput. In addition, encryption and decryption log entries increase pressure on the counter cache, causing higher counter access latency. However, it is challenging to deploy multiple encryption engines and larger capacity counter cache in the memory controller, due to encryption engine and counter cache area overhead.

To reduce encryption latency, we propose the *Log-Aware Memory Encryption* (LAME) scheme that reduces workloads for the encryption engine and the counter cache, without compromising data security and crash consistency. The proposed LAME is motivated by the following observation: the conventional counter mode memory encryption scheme encrypts/decrypts all data stored in the physical memory space in the same manner, without considering the contents of the plaintext. Assuming a modified data block content is M, and its home address is HA. Before writing to HA, the content M is encrypted and persisted to the logged data block address

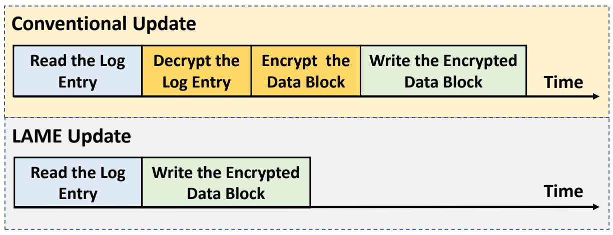

Fig. 4. Compare LAME In-Place Update with the Conventional In-Place Update

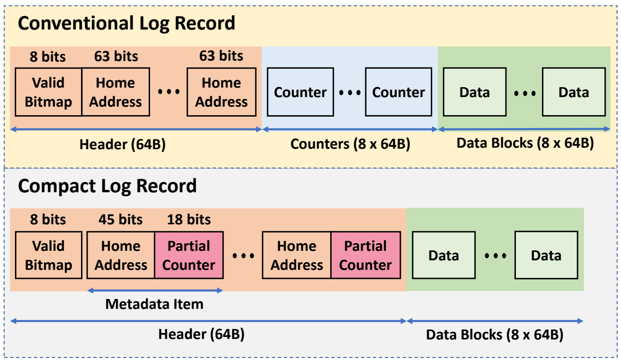

Fig. 5. Compare the Compact Log Record with the Conventional Log Record

LA as the block HA's log entry. Fig. 3 shows that the counter mode encryption encrypts the block in HA and the block in LA with different addresses and counters, and produces different ciphertexts; even their plaintexts are identical. Memory encryption operations could be avoided for two blocks with identical plaintext. The main idea of the proposed LAME is that a logged data block is encrypted with data block's home address, the counter of the home address, and the modified data block so that we can avoid a decryption operation and an encryption operation when performing the in-place update for this log entry. Fig. 3 illustrates how LAME and the counter mode encryption encrypt a logged data block differently. The conventional counter mode encryption uses the modified data content, M, the logged data block address, LA, and the counter of LA, to generate the $Ciphertext0$ in LA for the logged data block. However, LAME encrypts the modified data, M, with the modified data block's home address, HA, and the counter of HA to generate $Ciphertext1$ for the block in LA shown in Fig. 3. Doing so allows LAME to copy the content of logged data block, $Ciphertext1$, from the address LA in the log region to the address HA in the home region when performing an in-place update. Therefore, LAME avoids encryption operations required by the conventional counter cache mode. Furthermore, LAME can eliminate a decryption operation to obtain the plaintext of log data content M in the in-place update process. More details of LAME's in-place update are presented later. LAME does not require any changes to the encryption engine because it computes a logged data block's OTP with the same encryption engine as counter mode but with different security metadata.

LAME has two following benefits. Firstly, LAME can eliminate a decryption operation and an encryption operation when performing an in-place update for a log entry, to mitigate encryption/decryption latency. Fig. 4 shows the difference between the conventional in-place update and the proposed in-place update. Since the conventional log encryption treats writing logged data block as normal memory write operation, the logged data block's address and its counter are used to encrypt the logged data block itself. When performing an in-place update, the conventional scheme decrypts a logged data block to recover its plaintext with its address and its counter. Then, the conventional scheme encrypts this plaintext data block with its home address and the home address counter before writing the encrypted data block to the home address. Therefore, the conventional in-place update operation involves a decryption operation and an encryption operation. However, without using the encryption engine, LAME can directly copy the encrypted logged data block to its home address when performing an in-place update. This is because LAME's logged data block is encrypted with its home address and home address counter so that future memory accesses to the updated block can be correctly decrypted with the request address and the counter. Secondly, LAME can reduce the working set for the counter cache, to further reduce encryption/decryption latency. LAME's log entries' generation and in-place update do not access counters for the log region, leading to a higher hit rate for a given counter cache. Note that LAME's primary goal is not to improve the in-place update operations but to reduce encryption/decryption operations. These reduced operations can decrease the counter access latency and the queuing time for the encryption engine, minimizing the encryption latency. The decreased memory encryption latency can speed up the encrypting log entries on the critical path of transaction execution. The faster log entry encryption translates to shorter transaction commit latency, improving transaction throughput.

LAME and its in-place update scheme can ensure data security. LAME stores encrypted modified data blocks in the home region with the counter mode. LAME encrypts a logged data block by using the same procedure as counter mode but with different security metadata, which are its home address and counter of home address. If an attacker has the counter and the home address for a ciphertext in a log entry, the ciphertext still can not be decrypted, due to lack of the key, which can not be leaked in the counter mode encryption. A ciphertext stored in the log region is as secure as a ciphertext stored in the home region. When performing an in-place update, LAME just copies an encrypted logged data block to its home address, without causing security issues.

C. Compact Log Record

Counter mode encryption requires recording updated data block counters, making them part of log entries' payload. In conventional log record layout, when a modified 64B data block A generates a log entry, the 64B counter cache entry where the A's counter resides is also written to the log entry. This is because the access granularity for a counter in PM is 64B [6]. Therefore, the counter payload is as large as the updated data payload in the log entry shown in Fig. 5. The large size of log entry consumes PM write bandwidth, degrading system performance.

To reduce the size of the counter payload in log entries, we devise a compact log record and its in-place update scheme. This design is based on the following two observations. First, each counter update only increases its value by one, and a full 64B counter cacheline is not necessarily stored in a log entry. Second, the address component in a log entry unnecessarily has 63 bits, because PM's capacity is much smaller than 2^{63}B and a logged data block is 64B. Therefore, we store both data blocks' home addresses and their counters' less significant bits in a 64B log record header, as shown in Fig. 5. A log header contains eight metadata items and each item consists of a 45 bits address and an 18 bits counter. The 45 bits address is sufficiently large to cover 2^{45} cachelines, which is 2PB, and the 18 bits counter is the 18 less significant bits of a data block's counter. The remained 8 bits in the header are used to indicate the valid items in a record. In this way, we can significantly reduce the counter logging overhead.

Before performing an in-place update for a counter, we need to read the old counter block stored in its home address, since the log record header only has the less significant bits for this counter. After reading the old counter block, we replace the old counter's less significant bits with the corresponding partial counter stored in the log record header and then persist the updated 64B counter block to its home address. In this way, we can correctly update counters with negligible overhead. This optimization introduces an extra overhead to read an old counter block. However, this overhead is much lower than the performance gain brought by the reduction of log write traffic. This is because PM write operations are slower than read operations.

Since a log record header only has less significant bits of a counter, the overflow of the partial counter will incorrectly update the counter in the home address. To avoid this issue, we store the full 64B counter block to the data block in the record. As shown in Fig. 5, each log record consists of a header and eight logged data blocks. A metadata item in the header is associated with a logged data block in the log record. Upon partial counter overflow for the i-th metadata item, we store its home address and the encrypted data block in the i-th metadata item and i-th logged data block respectively, and set this item's partial counter to be zeros, to indicate the overflow. In addition, its full counter is stored in the $(i+1)$-th logged data block, and hence the header's $(i+1)$-th valid bit is set to be invalid. If a partial counter overflow occurs in the 8-th metadata item, this log entry is stored in the next log record, by leaving 8-th item unused.

D. Counter Buffer and Counter-Mapping Table

The compact log record dramatically reduces the log entry size and minimizes the log write bandwidth consumption. However, it increases the counter cache miss latency. To handle the counter cache miss, we need to load the requested counter cacheline which has eight counters for eight data blocks. It is possible that some of these eight counters' latest partial versions are stored in log entries and these counters in the home addresses have not been updated. In this scenario, we need to read these latest partial counters scattered in multiple log entries and the counter cacheline in home address to get the latest counter cacheline. In the worst case, eight log entries are read since the missed counter cacheline contains eight counters. These extra accesses to log entries increase counter cache miss latency.

To address the problem discussed above, we introduce the counter buffer and the counter-mapping table. When a dirty counter cacheline is evicted from the counter cache, it is written to the counter buffer in the PM. Since a dirty counter cacheline could have multiple speculatively modified counters, it can not be written to the home address. Otherwise, these speculatively modified counters can not correctly decrypt uncommitted data blocks stored in the home region after the system recovers from a power loss. The counter-mapping table, stored in the memory controller, maintains the mapping from a counter block home address and the counter block address stored in the counter buffer. To serve a counter request, the memory controller consults the counter-mapping table and the counter cache simultaneously. Upon hitting the counter-mapping table, a speculative counter can be served by the table. In the case of hitting the counter cache, a speculative counter can be accessed from the counter cache. The speculative counter cannot hit both the mapping table and cache at the same time. When a speculative counter misses both the mapping table and the cache, the memory controller fetches the counter block from its home address. When a dirty counter cacheline is evicted from the counter cache and is not in the counter buffer, its mapping entry is inserted into the counter-mapping table after the cacheline is written to the counter buffer. After in-place updates for log entries are done, their corresponding counter-mapping entries are freed. It is unnecessary to ensure the crash consistency for both the counter-mapping table and the counter cache because they are introduced to reduce the counter read latency, and the corresponding counters can be accessed from log entries during the recovery process.

E. Put It All Together

Write Data: Upon a transaction write request, the memory controller allocates the log memory space to log this write request and a header block in case of no spare header entry to use. When the request's home address counter is available in the counter cache, the memory controller increases the counter, and the proposed LAME encrypts the write request with the updated counter and its home address before writing it to the log region. In addition, the proposed compact log record scheme only stores the counter's less significant bits and home address to the header. When the header is full, it is flushed to PM. After the encrypted write request written to PM, the memory controller updates the mapping table.

Read Data: To serve a read request, the memory controller needs to retrieve the latest encrypted data block and its counter by looking up the mapping table and the counter-mapping table. The memory controller consults the mapping table with the request home address and reads the log entry indicated by the

TABLE I
SYSTEM PARAMETERS

Cores	4 out-of-order cores @2GHz,192 ROB entries, 48 STQ entries
L1 I/D Cache	private, 32KB, 2 cycles, 8 way
L2D Cache	private, 256KB, 8 cycles, 8 way
LLC	shared, 2MB per core, 64B cacheline, 25 cycles, 16 way
Memory Controller	32 write queue entries, 64 read queue entries 512KB counter cache, 512KB mapping table 512KB counter-mapping table
Persistent Memory	2 ranks, 16 banks, 16GB 300(48) ns write(read) [6], [10]
En/Decryption	40ns latency [6], [8], 16-stages pipe AES engine [8]

mapping table entry in case of matching the mapping table. Otherwise, the encrypted data block is read from the home region.

To access the counter, the memory controller looks up the counter-mapping table and the counter cache at the same time. Since a counter is exclusively stored in the counter cache and the counter buffer, the requested counter is read from where the counter resides. When hitting the counter-mapping table, the counter cacheline is also inserted to the counter cache, followed by being removed from the counter-mapping table. If both the counter cache and counter-mapping table miss, the counter is read from the home address and then inserted into the counter cache. When the insertion of the counter cacheline evicts a dirty counter cacheline from the counter cache, the victim counter cacheline is inserted into the counter buffer and its entry is inserted to the counter-mapping table. When the counter is available, the memory controller can decrypt the encrypted data before returning its plaintext to the CPU cache.

In-Place Update: The in-place update involves updating both data blocks and their counters in the home region. To update data blocks, the memory controller copies the logged data blocks to the home region specified by the corresponding headers for these committed transactions, then the related mapping entries are removed from the mapping table. To update counters, the memory controller reads the full counters from their home addresses and merges them with the partial counters stored in the headers. Since the partial counters are the latest less significant bits of updated counters, the merged counters represent the full and latest counters. After the full latest counters are persisted to PM, the memory controller may update the counter cache, the counter-mapping table, and the counter buffer. If these counters hit the counter cache and they have the same values as the counters in the counter cache, these counter cache entries are changed from dirty to clean because these modified counters have been written to PM. If these counters have the same values stored in the counter buffer, their corresponding counter-mapping entries are removed. After in-place updates are done, the memory controller reclaims log records memory space.

IV. EVALUATION

A. Experiment Setup

We implement the proposed designs in the simulator ChampSim [11] with DRAMSim2 [12]. ChampSim models the out-of-order execution CPU including detailed memory access behaviors. To accurately model PM accesses, our simulator incorporates the cycle-accurate DRAMSim2 to ChampSim, and supports *tx_begin* and *tx_end* to demarcate a transaction boundary. We list the simulator's configuration of CPU and memory system in Table I. The workloads include commonly used PM data structures, which are $RBtree$, $Hash$, $BPlustree$, $Btree$, and $Skiplist$. One transaction in the workload inserts or updates a key-value pair. The keys of these workloads follow the Zipfian distribution, similar to SSP [13].

We evaluated the following designs.

- *Secure Redo Logging* (SRL): We implement hardware-assisted redo logging with conventional counter mode encryption. After a modified data block is encrypted with the logged data block address and the counter of logged data block address, the corresponding ciphertext is stored to the log region. Before performing in-place update for a log entry, its ciphertext is decrypted with the log entry information, and the resultant plaintext is encrypted with the home address and the counter of home address. A log entry has a 64B counter block.
- *Log-Aware Memory Encryption* (LAME): We implement our log-aware memory encryption scheme for hardware-assisted redo logging. A log entry has a 64B counter block. It has the counter buffer and the counter-mapping table.
- *Compact LAME* (CLAME): We implement the log record optimization over LAME with the counter buffer and the counter-mapping table.
- *EASY-PM*: We implement EASY-PM [14], which is a state-of-the-art secure undo logging for PM. For fair comparisons, a header is not encrypted in EASY-PM in that headers stored as plaintext do not leak the data content.

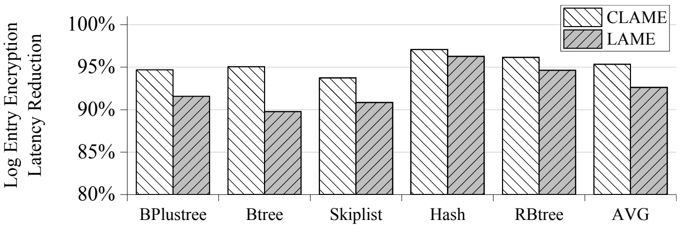

Fig. 6. Log Entry Encryption Latency Reduction

B. Log Entry Encryption Latency and Write Traffic

Fig. 6 shows log entry encryption latency reductions for CLAME and LAME, compared with SRL. The log entry encryption latency includes the counter access latency and the encryption engine access latency. CLAME and LAME can reduce the encryption latency by 95.3% and 92.6% on average, compared with SRL. Unlike the SRL, CLAME and LAME do not access counters of logged data block addresses and reduce the counter cache working set. The counters of these logged data block addresses have poor temporal locality, and the removal of accesses to these counters can increase the counter cache hit rate, reducing the counter access latency. CLAME and LAME can eliminate memory encryption and decryption operations for their in-place update operations and

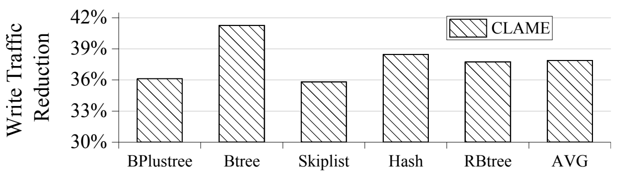

Fig. 7. Write Traffic Reduction

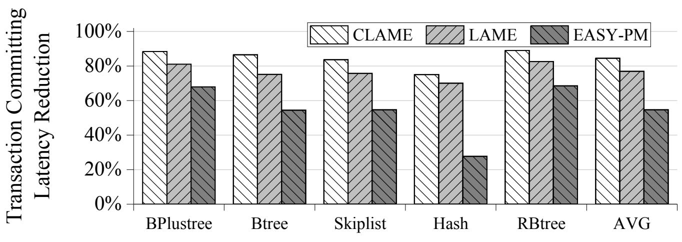

Fig. 8. Transaction Committing Latency Reduction

significantly reduce pressure on the encryption engine, leading to the lower queuing time. In addition, CLAME achieves shorter encryption latency due to the compact log record layout, which can efficiently reduce PM write traffic shown in Fig. 7.

Fig. 7 shows that CLAME can reduce write traffic by 37.9% on average, compared with LAME. This is because CLAME uses the spare bits in log headers to store the counter updates and dramatically reduces log record size, resulting in less log writing traffic. Therefore CLAME reduces memory write traffic over LAME.

C. Transaction Committing Latency Reduction

Fig. 8 shows CLAME, LAME, and EASY-PM can reduce the transaction committing latency by 84.5%, 76.9%, and 54.6% on average, compared with SRL. The redo logging can commit a transaction only after its log entries are written. Since writing these log entries accesses the encryption engine, a larger counter cache working set and more encryption and decryption operations lead to long encryption latency for writing these log entries. Since CLAME and LAME successfully reduce counter cache working set and workloads on the encryption engine, the encryption latency is significantly reduced, as shown in Fig. 6. The shorter encryption latency can speed up log write operations, leading to shorter transaction committing latency. However, EASY-PM, which is based on undo logging, has more synchronizing write operations than redo logging. Before committing a transaction, EASY-PM writes its log entries and performs in-place updates for its log entries. The extra write operations in the committing stage are one of the reasons for EASY-PM's longer committing latency. More importantly, EASY-PM fails to reduce encryption operations compared with CLAME and LAME; therefore EASY-PM takes a longer time to finish these write operations during committing. It is noticed that EASY-PM commits faster than SRL. This is because SRL's in-place updating encryption operations executed in the background compete with logging encryption operations occurred in a transaction's committing stage, while EASY-PM's undo logging's committing operations are not interfered with background updating operation. Furthermore, SRL uses extra logged data block counters to encrypt log entries, which has larger counter cache working set than EASY-PM.

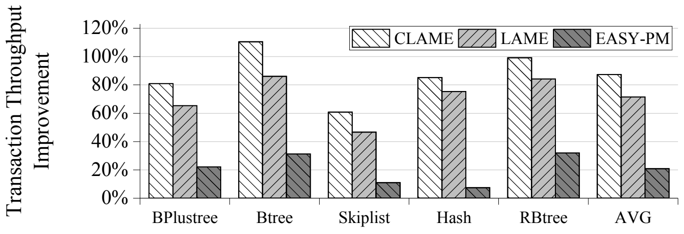

Fig. 9. Transaction Throughput Improvement

D. Transaction Throughput Improvement

Fig. 9 shows CLAME, LAME, and EASY-PM can improve transaction throughput by 87.3%, 71.5%, and 20.8% over SRL on average. Compared with LAME and EASY-PM, CLAME can improve transaction throughput by 9.2% and 55.3% on average, respectively. LAME can eliminate encryption and decryption operations during in-place update and avoid access to counters for the log region. CLAME's compact log record layout reduces the log write traffic, to improve throughput further.

V. Related Work

Crash Consistency. The adoptions of software logging to legacy systems not only require software modifications but also are prone to introduce bugs, increasing programmers' burden. Additionally, software logging introduces performance overhead caused by synchronizations between logging and updating. Research efforts on hardware logging approaches seek to address limitations of software logging [2], [3], [5], [7], [15]–[19]. ATOM [2] proposes a hardware undo logging design that moves the logging off the critical path and reduces the undo log metadata overhead. To speed up in-place update for redo logging, ReDU [5] proposes the hardware redo-logging which stores transaction updates in a large DRAM-cache, and asynchronously flushes them from DRAM-cache to PM, without reading from the log in the slow PM. Ref. [16] proposes a hardware undo+redo logging method, that can leverage existing caching policies to relax memory data persistent order constraints. HOOP [17] introduces fine-granularity logging, efficient address remapping, and the merging in-place update operations across transactions, to improve redo logging system performance. MorLog [15] proposes a hardware logging design that avoids unnecessary data logging, and also dynamically selects log encoding scheme to reduce write traffic.

Secure PM. Due to its non-volatility, PM devices are vulnerable to attacks and PM encryption is necessary. After identifying the counter atomicity overhead for the counter mode encryption, Ref. [6] proposes the selective counter atomicity to reduce the overhead, which applies counter atomicity to memory updates keeping data recoverability. DeWrite [20] applies memory deduplication to reduce PM write traffic, and also applies encryption operations for the secure PM. SuperMem [21] leverages a write-through counter cache to persist data and its counter atomically, and proposes a counter write coalescing scheme and cross-bank counter storage scheme to reduce the counter write overhead. Recently, there are works focus on PM coupled with memory encryption and integrity verification [10], [22]–[26]. Osiris [22] re-purposes ECC to sanity-check counter validity and avoids premature counter evictions from the counter cache to decrease PM write traffic. STAR [26] persists the modifications of the integrity tree parent nodes in their child nodes to efficiently recover dirty metadata, and it also conducts a multi-layer index to speed up recovery process. Janus [10] proposes parallelizing and pre-executing memory encryption, memory verification, and others, to improve PM system performance. Triad-NVM [23] discusses the security metadata persisting and studies the efficient recovery of the memory integrity tree. EASY-PM [14] is the most related to our work. However, our proposed methods remove a significant portion of memory encryption/decryption operations and reduce logging write traffic, outperforming EASY-PM.

VI. Conclusion

Achieving both memory encryption and crash consistency is required for a persistent memory system. The conventional memory encryption scheme fails to hide memory encryption latency for a memory write operation and this encryption latency is on the critical path for log entry write operations, which is required for crash consistency, degrading system performance. We propose a novel *Log-Aware Memory Encryption* (LAME) scheme to reduce encryption/decryption operations, without compromising data security, to tackle this issue. Specifically, LAME encrypts the updated data block with its security metadata and stores the ciphertext in the log entry, avoiding encryption/decryption operations of log data blocks in the process of in-place updating log entries. Furthermore, to mitigate the persistent overhead of encryption metadata in log records, we design a novel compact log record layout for logging encryption metadata efficiently. Simulation results show that our proposed designs boost the transaction throughput by 55.3% on average, over the state-of-the-art design.

References

[1] J. Yang, J. Kim, M. Hoseinzadeh, J. Izraelevitz, and S. Swanson, “An empirical guide to the behavior and use of scalable persistent memory,” in *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Santa Clara, USA, 2020.

[2] A. Joshi, V. Nagarajan, S. Viglas, and M. Cintra, “ATOM: Atomic durability in non-volatile memory through hardware logging,” in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Austin, USA, 2017.

[3] S. Shin, S. K. Tirukkovalluri, J. Tuck, and Y. Solihin, “Proteus: A flexible and fast software supported hardware logging approach for NVM,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Cambridge, USA, 2017.

[4] S. Shin, J. Tuck, and Y. Solihin, “Hiding the long latency of persist barriers using speculative execution,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, Toronto, Canada, 2017.

[5] J. Jeong, C. H. Park, J. Huh, and S. Maeng, “Efficient hardware-assisted logging with asynchronous and direct-update for persistent memory,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[6] S. Liu, A. Kolli, J. Ren, and S. Khan, “Crash consistency in encrypted non-volatile main memory systems,” in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, Austria, 2018.

[7] T. Nguyen and D. Wentzlaff, “PiCL: A software-transparent, persistent cache log for nonvolatile main memory,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[8] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, “Improving cost, performance, and security of memory encryption and authentication,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, New York, USA, 2006.

[9] V. Costan and S. Devadas, “Intel SGX explained.” *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[10] S. Liu, K. Seemakhupt, G. Pekhimenko, A. Kolli, and S. Khan, “Janus: Optimizing memory and storage support for non-volatile memory systems,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, New York, USA, 2019.

[11] Champsim. https://github.com/ChampSim/.

[12] Dramsim. https://github.com/umd-memsys/DRAMSim2.

[13] Y. Ni, J. Zhao, H. Litz, D. Bittman, and E. L. Miller, “SSP: Eliminating redundant writes in failure-atomic NVRAMs via shadow sub-paging,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Columbus, USA, 2019.

[14] Z. Zhang, J. Yue, X. Liao, and H. Jin, “Efficient hardware-assisted crash consistency in encrypted persistent memory,” in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*, Grenoble, France, 2020.

[15] X. Wei, D. Feng, W. Tong, J. Liu, and L. Ye, “Morlog: Morphable hardware logging for atomic persistence in non-volatile main memory,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, Valencia, Spain, 2020.

[16] M. A. Ogleari, E. L. Miller, and J. Zhao, “Steal but no force: Efficient hardware undo+redo logging for persistent memory systems,” in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Vienna, Austria, 2018.

[17] M. Cai, C. Chance, and H. Jian, “Hoop: Efficient hardware-assisted out-of-place update for non-volatile memory,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, Valencia, Spain, 2020.

[18] A. Joshi, V. Nagarajan, M. Cintra, and S. Viglas, “DHTM: Durable hardware transactional memory,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, Los Angeles, USA, 2018.

[19] Y. Deng, J. Yue, Z. Lu, and Y. Zhu, “Efficient hardware-assisted out-place update for persistent memory,” in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*, Grenoble, France, 2021.

[20] P. Zuo, Y. Hua, M. Zhao, W. Zhou, and Y. Guo, “Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[21] P. Zuo, Y. Hua, and Y. Xie, “SuperMem: Enabling application-transparent secure persistent memory with low overheads,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, New York, USA, 2019.

[22] M. Ye, C. Hughes, and A. Awad, “Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Fukuoka, Japan, 2018.

[23] A. Awad, M. Ye, Y. Solihin, L. Njilla, and K. A. Zubair, “Triad-NVM: Persistency for integrity-protected and encrypted non-volatile memories,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, New York, USA, 2019.

[24] K. A. Zubair and A. Awad, “Anubis: Ultra-low overhead and recovery time for secure non-volatile memories,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, New York, USA, 2019.

[25] A. Freij, S. Yuan, H. Zhou, and Y. Solihin, “Persist level parallelism: Streamlining integrity tree updates for secure persistent memory,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Athens, Greece, 2020.

[26] J. Huang and Y. Hua, “A write-friendly and fast-recovery scheme for security metadata in non-volatile memories,” in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Seoul, Korea, 2021.