OpF-STM : Optimized Persistent Overhead Fail-safety software transactional memory in Non-volatile memory

Jihyun Kim
Hanyang University
Seoul, Korea
wlgus3018@hanyang.ac.kr

Youjip Won
Hanyang University
Seoul, Korea
yjwon@hanyang.ac.kr

ABSTRACT
Non-volatile memory is emerged such as PCM and 3D XPoint. With the advent of Non-volatile memory, Software platforms have also been developed to manage Non-volatile memory areas. Recently those platforms support PTM system(Persistent transactional memory) which provides transaction system and guarantee crash-consistency of transaction at the main memory level. For ensuring crash-consistency of transaction, PTM system should use frequently hardware-instruction. Because ensuring persistent boundary has been changed volatile memory/storage to volatile cache/Non-volatile memory. This has a huge adverse effect on PTM system. In this paper, we propose a three techniques. Append-only dynamic log can support compact and dynamic log area. Lazy and bulk persistence aggressively delay persistence phase to commit phase. Non temporal persistence can provide enhanced memory copy function. Above techniques aim to reduce persistent overhead as many as possible. Our result shows that those techniques can enhance averagely 117% / 140% transaction performance.

CCS Concepts
• Information systems→Database management system engines

Keywords
Non-volatile Memory; Persistent software transactional memory; Crash-consistency; Multi-thread programming; Concurrency control.

1. INTRODUCTION
Next-generation memory (Non-volatile characteristics) such as PCM (Phase-change memory) [1] and STT-RAM and 3D XPoint [2] have been developed. Non-volatile memory can preserve data even if system is turned off abnormally, and makes it possible to access to data as byte granularity. Non-volatile memory is attracting attention as a next-generation memory device with fast read / write performance such as volatile DRAM. In recent years, there has emerged software platform that provides interfaces for managing Non-volatile memory area [3] [4] [5] [6] [7] [8] [9] [10]. Each mechanism supports user-level interface such as allocator / deallocator which can allocate/deallocate in Non-volatile memory area. Some of them also provide a transactional system by using STM (software transactional memory) which can support transaction system at main memory level and control the concurrency in multi-thread environment. Traditional STM can not guarantee Durability of ACID property (Atomicity, Consistency, Isolation, Durability). Because it has operated in volatile DRAM. However, NVRAM uniquely can provide persistence. It spawn many PTM system (Persistent transactional system) [5] [6] [8] [9] [10] which can guarantees transaction Durability. PTM systems try to guarantee crash-consistency of transaction and also provide recovery mechanism at main memory level.

ICICC'18, January 5–7, 2018, Manila, Philippines.

DOI: http://dx.doi.org/10.1145/12345.67890

To sure crash-consistency of transaction, PTM systems have established redo/undo logging. PTM systems have faced new issue which is persistent overhead. The boundary of ensuring crash-consistency has gone up 'volatile memory/storage' to 'Volatile cache/Non-volatile memory'. Ensuring cache and Non-volatile memory persistence is equally expensive using fsync() / fdatasync() [11] to guarantee file system consistency. The main reason why PTM system has a huge persistent overhead because it require expensive and frequent hardware instruction for flush CPU cache-line. It also get worse for logging because logging need additional memory write and flush operation. Recent research point out that those make system performance down dramatically and also they [5] [8] [9] [12] try to reduce persistence overhead to use their own mechanism. LSNVMM (Log structured Non volatile main memory) [5] use 'address mapping' between virtual home address and log address so that only one flush operation is needed. BP (Blurred persistence) delays data persistence until their static log is full [9]. It can reduce usage of flush operation. WB+tree (write atomic b+tree) [12] use append-only key-value inserting. This is very good for reduce additional write and flush operation because they do not have to sort key-value entry.

In this paper, our purpose is to aggressively reduce the persistent overhead in PTM system. We suggest three techniques, Append-only dynamic log area, LBP (Lazy and Bulk persistence) and NTP (Non-temporal persistence). We have implemented those skills in HEAPO: F-STM (Fail safety software transactional memory) [6] which are same as current PTM system. Append-only dynamic log write the log record to end of the log. This compacting technique can minimize to use CPU flush operation. Since adjacent 64byte of log record are cached in same cache-line. LBP is delaying persistence phase to commit phase. LBP also reduces CPU flush instruction since we can collect the log records as many as possible. Another profits of LBP is dynamic allocating log area. We can know precisely the number of log entry only in commit phase. This can save log space and increase space availability. NTP uses 'movntq' instruction which bypass CPU cache. This instruction is averagely a good performance for store operation [13] than other flush instruction. We constructed enhanced memcpy() function by using 'movntq' instruction for reducing persistent overhead. We confirmed that result of our

experiment can enhance the whole system performance. LBP improves transaction performance averagely 117%. NTP also enhances the transaction performance averagely 140% in HEAPO: F-STM.

2. BACKGROUND

2.1 Non-volatile memory

Non-volatile memory is the next generation of devices where data remains in memory after the system is turned off. Currently, There are PCM (phase change memory) [1], STT-MRAM (Spin Transfer Torque Magnetoresistive RAM), Intel 3DXpoint [2] and MARM. The read / write latency of the PCM is 100ns / 20ns, respectively. The STT-MRAM is 10 ns. Volatile DRAM read/ write latency is 10ns / 10 ns. Table 1 explains the read / write latency of Non-volatile and Volatile memory. Non-volatile memory has almost the same read / write latency as DRAM. In addition, since it has a unique feature called non-volatile, Future computer system based on non-volatile memory will be normalized.

Table 1. Features of Memory

Name	Latency (read/write)	Byte-access	Non-volatile
DRAM	10ns/10ns	yes	No
PCRAM	100ns/20ns	yes	yes
MRAM	12ns/12ns	yes	yes
STT-MRAM	10ns/10ns	yes	yes

2.2 HEAPO (Heap-Based Persistent Object Store)

HEAPO [6] is a software platform for managing Non-volatile memory areas. HEAPO reserves a part of the process virtual address space as 'Persistent Heap'. Persistent heap area is managed by object. Figure 1 represents the virtual address space of HEAPO. Each object has a unique name. Programmer can access those object area by using their own unique name. Each object virtual address is mapped to the physical page frame of NVRAM. HEAPO also provides interfaces such as object creation / deletion and memory allocation / deallocation which is similar to glibc malloc() / free(). Those interfaces is a user-level library implemented in c language. The pos_create() function creates an object in the 'persistent heap' area. pos_delete() is a function that deletes an object. pos_map() and pos_unmap() are functions that map / unmap the object to the physical page frame of NVRAM. pos_malloc() / pos_free() is a function that allocates and frees memory in byte granularity.

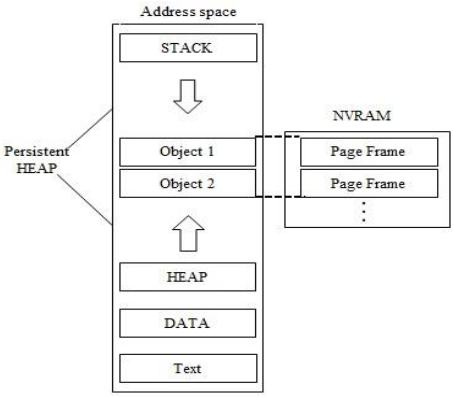

Figure 1. HEAPO (Heap- Based Persistent object

2.3 F-STM (Fail safety software transactional memory)

Blocking [14] based lock scheme is normally used for overcoming synchronization problem in multi-thread environment. However, lock sometimes cause Dead lock and convoying problem which provoke serious problem in system. STM (software transactional memory) is a technique for solving synchronization problems in shared memory access of multiple threads. STM borrowed the concept from a DB (Database) transaction [15]. All of read / write to memory is managed by transaction. STM basically utilizes the non-blocking technique. Non-blocking technique is very simple. All threads only check whether memory address which they want to write is written or not. Assuming one thread 'A' want to write data to memory address 'a'. Another thread 'B' also wish to write some data to address 'a'. Thread B firstly check address 'a'. Thread B now know that Thread A is writing. Hence Thread B abort it own transactions and retry repetitively until Thread A finish to write. Since STM borrowed the concept of DB transaction, STM transactions can basically guarantee ACID properties (Atomicity, Consistency, Isolation, Durability). However, it is impossible to assure Durability because STM is working on DRAM.

The advent of Non-volatile memory, STM transactions can guarantee Durability. It means that we can recover transaction at main-memory level only if we implement logging or any other skills for consistency. We have applied word-based [16] TinySTM [17] to HEAPO. TinySTM is light-weight software transactional memory. We have moved TinySTM data structures (metatdata , log storage and so on) to NVRAM by using HEAPO interface. We called to this HEAPO: F-STM (Fail safe software transactional memory). Our F-STM can support both of Redo logging and Undo logging. Redo logging firstly store the memory write information in log area. At commit phase, a pair of log record (address/value) checkpoint the actual memory location. Undo logging do 'in-place update' for memory write. Next step is that they record the old information in log area.

Figure 2 show sample code of F-STM. F-STM make all of memory store/load by MACRO. The code block from TM_INIT_THREAD to TM_EXIT_THREAD is a transaction. F-STM has log storage for each thread. Each storage stores all memory write information. TM_INIT_THREAD initialize transaction descriptor which has information of read/write set. TM_START initializes the log repository for rollback information. TM_LOAD / TM_STORE each perform memory read / write. TM_COMMIT validate confliction with multi thread and reflects all of contents in memory store/load information in the log.

```
00 TM_INIT_THREAD
01 TM_START
02 TM_STORE(&node.key,key);
03 TM_STORE(&node.value,value);
04 TM_STORE(&node.next ,TM_LOAD(head);
05 TM_STORE(&head,node);
07 TM_COMMIT
08 TM_EXIT_THREAD
```

Figure 2. Sample code of HEAPO: F-STM

3. PROBLEM

In a DRAM-based computer system, it is not need to guarantee crash-consistency of data at the main memory-level. This is

because all data should be disappeared when the system is powered off. Recently, STM-based PTM (Persistent transactional memory) [5] [6] [8] [9] [10] system is emerged. It is possible to assure crash consistency of transaction at main memory level. The existing file system guarantees the crash-consistency of transactions by using techniques such as journaling [18] and soft-update [19]. Our HEAPO: F-STM [6] also supports techniques such as Redo / Undo logging to guarantee transaction crash-consistency. All memory write information in the transaction is logged in the NVRAM log area. In general, file systems use operating system calls such as 'fsync()' or 'fdatasyne()' to store files in storage. Emerging PTM system, persistent boundary has moved memory-storage to cache-memory. To ensure the persistence of cache-memory data, PTM system should perform a cache-line flush operation. CPU cache is generally non-volatile. If we want to keep data consistency in CPU cache-line. We should have to use flush operations. Hardware instructions such as 'clflush' and 'clflushopt' are provided. Figure 3 shows the code that guarantees transaction durability in a typical PTM system. Each log records (address, value) perform a flush operation immediately after being written to the log area of NVRAM.

To sure persistence of transaction in the PTM system, we should explicitly insert a flush instruction immediately following the code where the memory write occurred. Because we never recognize the log record is only updated on the CPU cache-line or it is written to the actual memory. CPU eviction policy is another problem. CPU-cache is hardware-controlled, it can not predict when cache eviction is performed. In conclusion, this explicit flush operation should be needed because of characteristic CPU cache. Those frequent flush operation is main factor that degrades the performance of PTM system. Several recent studies have proposed some techniques to reduce persistent overhead. We should have to minimize persistent overhead as many as possible to enhance the whole of PTM system performance.

```
00 fstm_wbetl_write(log, address ,value){
01 log->address = address;
02 clflush_mfence(&log->address);
03
04 log->value = value;
05 clflush_mfence(&log->value);
06
}
```

Figure 3. Traditional guarateeing method about transaction in PTM

4. IMPLEMENTATION

Our techniques was implemented in HEAPO:F-STM. F-STM choose Redo logging. We can not define what is the best-fit for F-STM. Redo logging is a little better than intensive write transaction [20]. We chose Redo logging for experiment. We also used DRAM buffers for logging. DRAM buffers help protect the consistency of data since hardware-controlled cache unpredictably evicts cache-line (cache-eviction) in log area. Because They use same CPU bus, write overhead is mitigated for buffering. We re-construct our log area are as Append-only dynamic log and apply LBP and NTP.

4.1 Append-only dynamic log

We implemented the log area as Append-only dynamic log. All log records are compactly inserted at the end of the log. We also allocate the log area at commit phase. Append-only log can reduce the number of flush instruction. All of CPU caching and flushing operation are performed in unit of 'cache-line size (64byte/128byte)'. If the log records are inserted tightly in log area, it is possible to reduce the number of flushes. Because explicit flush operation is not required for adjacent log records within cache-line size (64byte / 128byte). If we know the start address of log area, we just can call flush operation in multiples of cache-line size (64 byte / 128 byte). At commit phase, we can count the number of log record. It means that we can allocate log area as we need. Dynamic log can efficiently use in Non-volatile main memory.

4.2 Lazy and Bulk persistence

We introduces transaction-aware LBP (Lazy and Bulk persistence). The key idea of LBP is to delay flushing log record until commit phase. Recovery scope of PTM system is a transaction unit. Hence, we do not care about every memory store/load [21]. We just wait to stack all of log records and flush them all at commit phase. But, we should have to ensure the ordering of commit phase and persistence phase. Before modifying commit mark, we perform 'bulk flush' of all records in the log by using 'clflush+mfence' instruction. Figure 4 illustrates LBP. The base line system flushes every log records when log is written. This causes a large flush overhead. LBP is beneficial in two aspects. First, we can reduce the use of the frequent flush instruction. Append-only dynamic log can help to reduce more flush operation. We flush all of log data exactly (log size / cache-line size) + 1. Second, we can count exact the number of log records since we delay flush operation until commit phase. It means that we can allocate log area dynamically. Therefore, we can utilize the log area Non-volatile memory as much as we need. This improve space availability in Non-volatile memory.

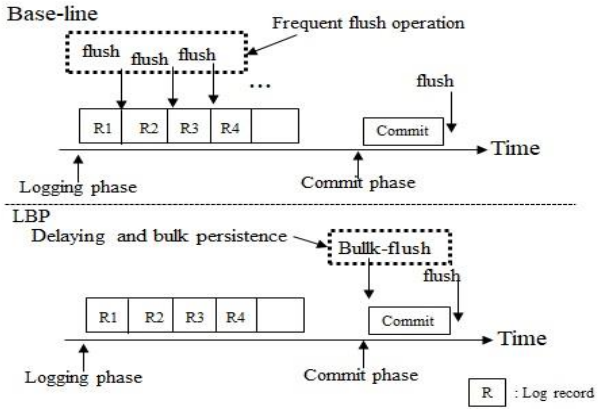

Figure 4. LBP(Lazy and Bulk persistence)

4.3 Non-temporal persistence

We exploited memcpy_movntq() function which is highly efficient memcpy() by using 'movntq' instruction. 'movntq' instruction directly move data register to memory bypassing CPU cache. To write log record directly log area, first, we should move the log records to mmx register by using 'mov' instruction. Normally, the number of mmx registers is eight. That means we only can write 64byte data from register to memory atomically. If we finish to move log record to mmx register, and then we start to write log record from mmx register to actual memory location by using 'movntq' instruction. NTP also flush size of log area. It is good for space availability. Before using advanced memcpy_movntq() function, we should insert memory barrier instruction 'sfence' for ordering and flushing write combing buffer [13]. And also put above instruction at last. Figure 5 shows

our efficient memcpy function. First, all log record is stored in DRAM buffer. We can collect write information in write_set and copy to log area by using memcpy_movntq().

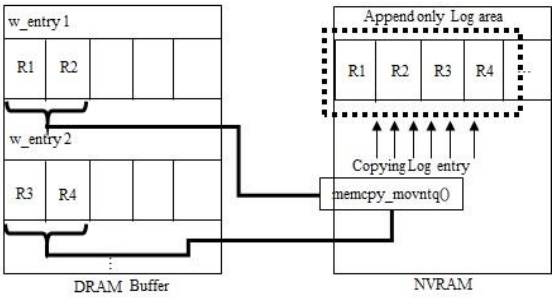

Figure 5. NTP(Non-temporal persistence)

5. EVALUATION

5.1 setup

Table 2 shows our setup environment. We use Ubuntu 10.04 applied Linux 'kernel 2.6.38'. HEAPO : F-STM has also implemented.

Table 2 Enviorment of Experiment

CPU	Intel i7-3820
Memory	Samsung DDR3 SDRAM 12GB
OS/Kernel	Ubuntu 10.04/Linux 2.6.38

5.2 Experiment

We carried out our experiment using virtual NVRAM consist of SDRAM. The workload is the insert operation in linked-list. All information such as pointer and values that occurs in the process of inserting a node into a linked list is called 'one transaction'. We insert one million node. The size of the node is 64bytes. The number of threads is each 1/2/4. Each thread try to make only one linked list data structure. Each of them tries to insert 250 thousand of node. We compared NP (No persistence), Base (Baseline system), LBP (Lazy and bulk persistence) and NTP (Non-temporal persistence).

5.3 NP (No-persistence) / Baseline system

NP (No persistence) shows the best transaction throughput. It never use flush instruction such as 'clflush + mfence' instruction. It can not sure the consistency of data. It is very dangerous to use NP. Baseline system use flush operation for every record when is written as shown figure 4. Baseline system has a worst transaction throughput as show figure 6.

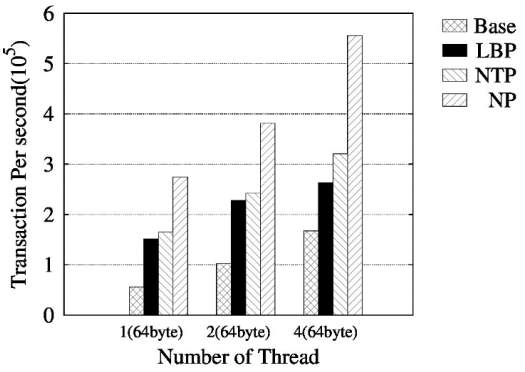

Figure 6 Transaction throughput in single/multi thread

5.4 LBP (Lazy and bulk persistence) / NTP (Non temporal persistence)

LBP uses 'clflush+mfence' flush operation. It can reduce the number of flush operation owing to combining Append only dynamic log and LBP. Figure 6 shows transaction throughput is averagely 117% high than baseline system. NTP shows the best performance in our F-STM system. Transaction throughput is as high as 140% of the baseline system as shown figure 6.

5.5 Multi-thread evaluation

In multi- mode, Best performance is NTP expect for NP. Next is LBP. NTP has more 194%, 136% and 91% performance than baseline system. LBP has more 170%, 122% and 57% performance than baseline system. Compared to NTP and LBP NTP has averagely 13% transaction throughput. Our experiments showed that the best performance was achieved in four threads. Figure 6 shows our results of experiments. The tendency of transaction throughput has not been changed even though we changed number of thread 1/2/4.

6. CONCLUSION

PTM (persistent transactional memory) system can support transaction system and also provide main memory-level recovery. However , it has faced big issue called 'persistent overhead' by maintaining crash consistency of transaction. In this paper, we attempt to reduce persistent overhead in HEAPO: F-STM using three techniques (Append-only dynamic log, LBP, NTP). Append-only dynamic log give a chance to insert log records tightly. LBP is possible to flush all of records at one point. NTP uses upgraded memcpy() function using cache-bypass hardware instruction so that it dramatically reduce persistent overhead. We show that that our suggestions practically enhance transaction throughput averagely 117% / 140%.

7. ACKNOWLEDGMENTS

This work was supported by the BK21 plus program through the National Research Foundation (NRF) funded by the Ministry of Education of Korea, the ICT R&D program of MSIP/IITP (R7117-16-0232, Development of extreme I/O storage technology for 32Gbps data services), Basic Research Laboratory Program through the National Research Foundation (NRF) funded by the Ministry of Science, ICT & Future Planning (MSIP) (No. 2017R1A4A1015498), and the MSIP(Ministry of Science, ICT&Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2016-H8501-16-1006) supervised by the IITP(Institute for Information&communications Technology Promotion).

8. REFERENCES

[1] Qureshi, M.K., Srinivasan, V., and Rivers, J.A. Scalable high performance main memory system using phase-change memory technology. *Proceedings of the 36th annual international symposium on Computer architecture - ISCA* 09, (2009).

[2] Intel and Micron. Intel and micron produce breakthrough memory technology, 2015. https://newsroom.intel.com/news-releases/

[3] Coburn, J., Caulfield, A.M., Akel, A., et al. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. *ACM SIGPLAN Notices 47*, 4 (2012), 105.

[4] Giles, E.R., Doshi, K., and Varman, P. SoftWrAP: A lightweight framework for transactional support of storage class memory. 2015 3 *1st Symposium on Mass Storage Systems and Technologies (MSST)*, (2015).

[5] Hu, Q., Ren, J., Badam, A., Mosciboroda, T,. Log Structured Non-Volatile Main Memory. *In Proceedings of 2017 USENIX Annual Technical Conference (USENIX ATC)*, (2017)

[6] Hwang, T., Jung, J. and Won, Y. 2014. HEAPO: Heap-Based Persistent Object Store. *ACM Transactions on Storage*. 11, 1 (2014), 1–21.

[7] Intel, The NVM Library. http://pmem.io/, 2016.

[8] Liu, M., Zhang, M., Chen, K., Qian, X., Wu, Y., Zheng, W. and Ren, J. 2017. DudeTM: Building Durable Transactions with Decoupling for Persistent Memory. *ACM SIGOPS Operating Systems* Review. 51, 2 (Apr. 2017), 329–343.

[9] Lu, Y., Shu, J., and Sun, L. Blurred persistence in transactional persistent memory. 2015 31st Symposium on *Mass Storage Systems and Technologies (MSST)*, (2015).

[10] Volos, H., Tack, A.J., and Swift, M.M.Mnemosyne: Lightweight Persistent Memory. *ACM SIGPLAN* Notices 47, 4 (2012), 91

[11] Pillai T S, Chidambaram V, Alagappan R, Al-Kiswany S, Arpaci-Dusseau AC, Arpaci-DusseauRH. Crash consistency. *Communications of the ACM*, 2015, 58(10): 46-51.

[12] Pillai T S, Chidambaram V, Alagappan R, Al-Kiswany S, Arpaci-Dusseau AC, Arpaci-DusseauRH. Crash consistency. *Communications of the ACM*, 2015, 58(10): 46-51.

[13] Jung, J. and Won, Y. nvramdisk: A Transactional Block Device Driver for Non-Volatile RAM. *IEEE Transactions on Computers* 65, 2 (2016), 589–600.

[14] Shavit, N. and Touitou, D. Software transactional memory. *Distributed Computing* 10, 2 (1997), 99–116.

[15] The SQLite, http://www.sqlite.org/

[16] Felber, P., Fetzer, C., And Riegel, T. Dynamic performancetuning of word-based software transactional memory. In Proceedings of the 13th *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2008), PPoPP '08, ACM, pp. 237–246.

[17] Tinystm: A lightweight and efficient software transactional memory implementation in c

[18] Seltzer, I, M., Ganger, R, G., McKusick, K, M., Smith, A, K., Soules, N, A, C., Stein, A, C., Journaling versus soft updates: Asynchronous meta-data protection in file systems *In USENIX Annual Technical Conference*, General Track (2000).

[19] Ganger, G.R., Mckusick, M.K., Soules, C.A.N., and Patt, Y.N. Soft updates: a solution to the metadata update problem in file systems. *ACM Transactions on Computer Systems 18*, 2 (2000), 127–153.

[20] Wan, H., Lu, Y., Xu, Y., and Shu, J. Empirical study of redo and undo logging in persistent memory. 2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA), (2016).

[21] Kim, W.-H., Kim, J., Baek, W., Nam, B., and Won, Y. NVWAL: Exploiting NVRAM in Write-Ahead-Logging. *ACM SIGOPS Operating Systems* Review 50, 2 (2016), 385–398.SELTZER, M. I., GANGER, G. R., MCKUSICK, M. K., SMITH, K. A., SOULES, C. A., AND STEIN, C. A.

[22] Scherer, W.N. and Scott, M.L. Advanced contention management for dynamic software transactional memory. Proceedings of the twenty-fourth annual *ACM SIGACT-SIGOPS symposium on Principles of distributed computing - PODC* 05, (2005).