Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186–5140
Volume 5, Number 1, pages 59–63, January 2016

Durability Guarantee in Software Transactional Memory

Keonwoo Lee
Department of Computer Software
Hanyang University
Seoul, Korea
kunulee@hanyang.ac.kr

Youjip Won
Department of Computer Software
Hanyang University
Seoul, Korea
yjwon@hanyang.ac.kr

***Abstract*—Non-volatile memories are next generation storage devices which maintain data on memory cell regardless of system power. However, non-volatile memories are hard to guarantee transaction's ACID properties when abnormal system crash occurs. In this paper, we implement fail-safe software transactional memory(F-STM). F-STM guarantee durability by locating F-STM's data structure to non-volatile memory. Using F-STM, computer structure based on non-volatile memory can provide data integrity.**

I. Introduction

Non-volatile memory can read/write data in the memory cell without separate electrical approach. In addition, it can maintain data in non-volatile memory cell without maintenance of electricity like auxiliary storage device (e.g, SSD,HDD,Flash Memory). Because of Non-volatile memory is bytes-addressable, its rapidly rising to a next-generation device which can endure weaknesses of existing volatile memory and flash memory.

At present, many systems are based on volatile memories. Many studies are making an effort to change computer structure based on volatile memory to the structure which can utilize non-volatile memory[4],[18],[9]. For the representative studies in the field of non-volatile memory, there are development of middleware platform helping writing non-volatile memory area or design of data structures for only non-volatile memory. For the representative middleware platform managing the field of non-volatile memory, there are NV-HEAP [4], Mnemosyne [18], HEAPO [9], etc.

Volatile memory extinguishes data completely if the process finishes abnormally because of the system error. On the contrary, data is remaining in non-volatile memory in spite of the abnormal error, so it can be reused after the recovery. Therefore, systems utilizing non-volatile memory need the method guaranteeing ACID of transaction unit for the work reading/writing with non-volatile memory. For example, lets consider the case ACID of transaction unit is not guaranteed for the errors made during the process to insert the node of list data structure with non-volatile memory. If there is any abnormal error at the moment when node for insertion is allocated, the node will be garbage after the recovery. It requires expensive garbage collection[3]. If there is any error during the update of the pointer of node or head node for insertion, address of nodes point can be incorrect.

Computer structure based on volatile memory causes a problem violating ACID of the data between volatile memory and auxiliary memory device. Existing computer structure (based on DRAM) guarantees ACID between two classes with methods such as journaling [15], shadow paging [20], versioning [17] and copy-on-writte[8] in the class of file system. In computer structure utilizing non-volatile memory, there is a problem violating ACID of data between volatile CPU cache and non-volatile memory. Representative studies redesigned by hardware so that CPU cache can have non-volatile characteristics[11] or solved by separate storage like journaling.

In this paper, we guarantee transaction's ACID properies by using fail-safe software transactional memory(F-STM). This paper is organized by 3 chapters. The first part is related to background knowledge and introduces platform and library, this paper used for guaranteeing ACID of data. The second part introduces problems while non-volatile memory uses STM. Finally show the performance about Key-Value insert operation on F-STM.

II. Background

A. Non-Volatile Memory

All of computer systems use relatively slow storage (e.g, HDD,SSD) as an auxiliary storage device. Slow storage uses volatile memory as a buffer and relieves the bottleneck of storage reading/writing(IO) for an alternative of high speed. In comparison with IT technology which is rapidly developed, minute process technology of the memory is staying at 30nm. To endure the technical limits, existing computer structure is rapidly changed. For a representative study, there is non-volatile memory. Many studies are finding methods to replace computer structure of memory storage with non-volatile memory. New memory can maintain and manage data in the memory without separate electrical approach and is rapidly rising to the next-generation device which can keep data all the time without maintenance of electricity like an auxiliary storage device.

For the representative non-volatile memories, there are STT-MRAM(Spin Transfer Torque-Magnetoresistive RAM), PCRAM(Phase Change RAM), ReRAM(Resistive RAM), etc.

Through the Table I, characteristics of each non-volatile memory were compared. Non-volatile memory has respectively physical characteristics and fields using it should be different minutely. For example, STT-MRAM has speed similar to DRAM with magnetic characteristics and is a representative non-volatile memory replacing DRAM, solving power consumption, a troublesome problem of DRAM. On the other hand, PCRAM will store status of data with material changes and be used to replace buffer cache of volatile memory, used in large-quantity server. In addition, FRAM is thoroughly low-density and highly expensive, but reading/writing(IO) speed is high.

Name	speed	Byte addressable	nonvolatile
PCRAM	75ns	Yes	Yes
FRAM	100ns	Yes	Yes
STT-MRAM	10ns	Yes	Yes
ReRAM	75ns	Yes	Yes
DRAM	10ns	Yes	No

TABLE I
CHARACTERISTIC OF NONVOLATILE MEMORY [10],[13],[19].

B. HEAPO

HEAPO(Heap-Based Persistent Object Store) [9] is a middleware platform managing non-volatile memory. Middleware platform managing non-volatile memory not only provides convenient interface so that programmers can freely use the field of non-volatile memory after the allocation of the field of non-volatile memory but also solves several issues (e,g. allocation/deallocation of non-volatile physical page, guaranteeing of ACID of transaction unit) occurred for managing non-volatile memory.

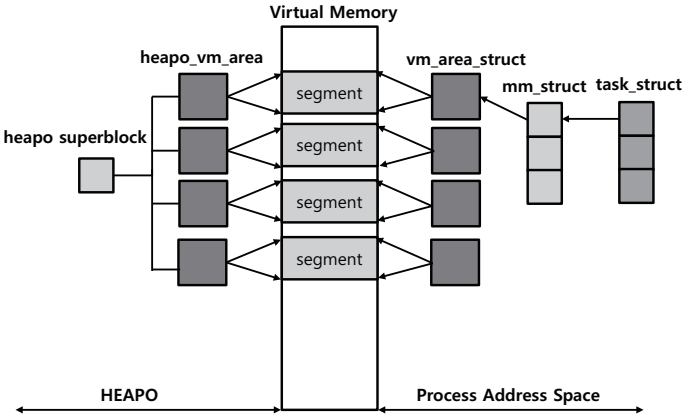

Fig. 1. Process Address Space in HEAPO

Fig. 1 shows virtual address space of the process allocated by non-volatile memory through the HEAPO. If HEAPO allocates non-volatile memory, its mapped to the space of virtual address in the field of heap. The address for the mapping of non-volatile physical page is provided by letting HEAPO use a part of heap field, a space of process address used for dynamic allocation(`malloc()`,`free()`). HEAPO provides `pos_malloc()`/`pos_free()` functions so as to allocate/deallocate non-volatile physical page by chunk units as if invoking `malloc()`/`free()` for allocation/deallocation of dynamic memory in glibc. Programmers who make application programs with non-volatile memory can receive dynamic allocation of non-volatile physical memory freely with interface provided by HEAPO.

C. Software Transactional Memory

Software Transactional Memory is the mechanism solving concurrency control by only software approach without any particular blocking [16]. Unlike lock mechanism, STM(referred to software transactional memory) can do works by simultaneous approach of several threads(non-blocking). TinySTM is Software Transaction Memory Library based on word[1]. TinySTM has separate log storage of each thread for concurrency control. Log storage of each thread keeps recovery information for memory store/load occurred in the transaction. TinySTM cognizes the problem occurred while transaction approaches address of other transaction during the writing work of particular memory address or phenomenon changing particular memory address while other transaction does writing work of the address after writing work of particular memory address, so aborts one transaction between two transactions and maintains the status of transaction consistently all the time. Aborted transaction rollbacks with memory store/load rollback information stored in log storage by threads.

```
struct R{
    void *value;
    int valid;
};

atomic set
{
    R.value = 0xC0F;
    R.valid = 1;
}

(TinySTM example)
TM_INIT_THREAD
  TM_START
     TM_STORE(&R.value, 0xC0F) ;

     TM_STORE(&R.valid, 1);
  TM_COMMIT
TM_EXIT_THREAD
```

Fig. 2. TinySTM example source code

TinySTM carries out concurrency control for memory store/load of transaction unit like Fig. 2. If the thread is made, log storage is made by TM_INIT_THREAD macro-threads. If the thread finishes, log storage is eliminated by TM_EXIT_THREAD macro. The start and end of the transaction are surrounded by TM_START and TM_COMMIT. TM_START initiates rollback information for memory store/load managed by log storage of each thread. TM_COMMIT tests the conflict by other transaction and reflects contents of memory store/load in the log storage before

the transaction is committed. Log storage of each thread is organized by two storages. Memory operation is organized by reading and writing, so log storage of each thread is managed by read_set managing rollback information for load and write_set managing rollback information for store. In transaction of TinySTM, memory load/store uses TM_STORE and TM_LOAD. TM_STORE stores rollback information in write_set of log storage and TM_LOAD stores rollback information in read_set of log storage. Rollback information for memory load/store of transaction by threads is stored for the rollback of two transactions related to the conflict after the operation of contention manager in TinySTM. Contention Manager reflects one transaction and aborts another transaction.

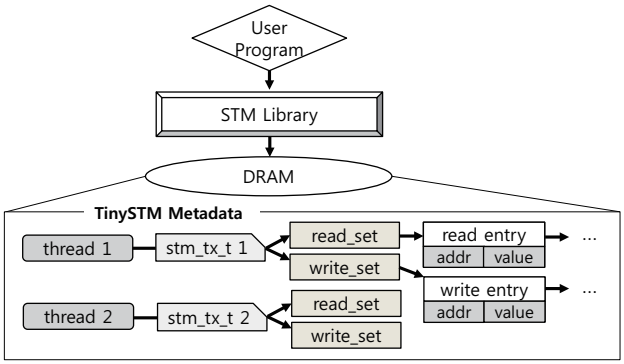

Fig. 3. TinySTM structure on DRAM

From now on, this chapter will examine data structure managing for the concurrency control of TinySTM in volatile memory and examine operative process of TM_STORE, memory writing macro provided by TinySTM, in detail. Fig. 3 indicates data structure of TinySTM in multi-thread environment. TinySTM has descriptor(stm_tx_t) in each thread. During the same transaction of two threads, there are status of transaction working for cognition of the conflict and abort of one transaction, address information returning to TM_START during the rollback of the transaction and addresses for write_set,read_set.

Memory store/load provided by TinySTM is stored in read_set and write_set, log storages of each thread, and there is one by each thread. Log storage has start address of the list related by number of rollback information. As a result of TM_STORE, rollback information related to write_set is write_entry. As a result of TM_LOAD, rollback information related to read_set is read_entry. write_entry and read_entry are organized by pair of value/address for reading or writing due to the TM_STORE,TM_LOAD. The structure is used for the rollback returning the transaction related to the conflict to the previous status during the conflict. TinySTM is data structure for only volatile memory and all of current data structures are allocated and managed in DRAM.

III. Problem

Consistency and Durability must be guaranteed so that Software Transactional Memory can guarantee ACD of transaction in non-volatile memory. In case of transaction commit, Consistency has a property to guarantee coincidence of the contents between volatile media and non-volatile media. Durability is a property to carry out the recovery process of data worked and guarantee perfectness of data during the intermediate commit of transaction by system crash.

STM allocates and operates data structures in volatile memory and all of transactions worked during the Power Failure are lapsed. Therefore, its not important that TinySTM guarantees the consistency between volatile CPU cache and volatile DRAM for itself. However, because of the introduction of non-volatile memory, its important to guarantee the consistency between CPU cache and NVRAM because data written in volatile CPU cacheline must guarantee the flush of non-volatile memory in order of the writing all the time. But, TinySTM, the library for only volatile memory, doesnt have the guaranteeing mechanism.

In addition, STM needs mechanism recovering system crash occurred during the transaction. Transaction is a set of arithmetic operations which should be occurred logically all at once. Although written in non-volatile memory, perfectness of data must be maintained by returning to undo of failed transaction or reflecting redo of transaction after the recovery of system crash. But, TinySTM, the library for only volatile memory, doesnt have the guaranteeing mechanism.

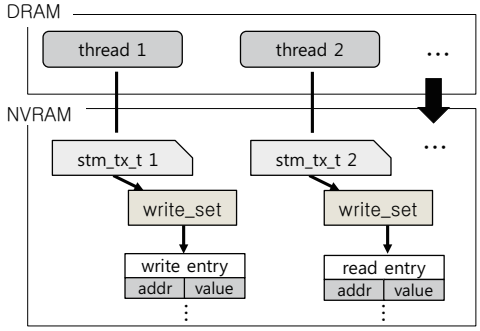

Fig. 4. TinySTM Structure in NVRAM

IV. Design and Implementation

A. Persistent Software Transactional Memory

The structure of computer based on non-volatile memory stores working status to non-volatile memory when power failure or system crash on program operation occurs. It is called as permanence of non-volatile memory. Therefore, middleware platforms managing non-volatile memory can reuse data remaining in non-volatile memory after system recovery. Data for reuse must have consistency after the recovery because it cannot be expected by user as a result of error. Data structure of TinySTM is currently on volatile memory. If TinySTM causes power failure or system crash during the concurrency control, all of data structures in progress of works are deleted. The library of TinySTM used by HEAPO didnt design mechanism guaranteeing Failure-Atomicity. This chapter suggests the method to let log storage of each thread,

managed by TinySTM, in non-volatile memory. TinySTM located in non-volatile memory can block deletion of data in progress of works due to the abnormal error and log storage of each thread is located in non-volatile memory, so Consistency and Durability for non-volatile memory can be guaranteed by stored log storage. Fig. 4 shows location of data structure TinySTM allocates in volatile memory to non-volatile memory. In consideration of Figure, descriptor and log storage of each thread are allocated in non-volatile memory. write_set is located in non-volatile memory, so recovery information stored by TM_STORE will be located in non-volatile memory. Suggested design stores recovery information in non-volatile memory after the error unlike TinySTM for only volatile memory deleting recovery information because of the abnormal error, so its possible to recover consistently.

B. Failure-Recovery Mechanism

This chapter suggests recovery method for system crack with TinySTM data structure located in non-volatile memory. TM_STORE is memory writing operation of TinySTM. Parameter of TM_STORE is organized by memory address and value and its function is to write the value in the memory as mentioned before. When TM_STORE is called, the value originally stored is logging in the memory address before TM_STORE writes the value in memory address.

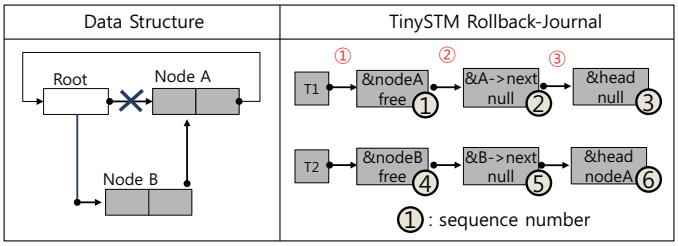

Fig. 5. Sequence number to manage recovery: (1)Alloc Node and Copy K-V,(2)Write Pointer,(3)Update Pointer

Therefore, if there is system crack, Memory writing storage (write_set) of TinySTM maintains recovery information in non-volatile memory as it is. Recovery information of write-through is values (write_entry) stored originally in memory address. This chapter suggests mechanism of rollback of previous undo managed in non-volatile memory. Recovery method for system crack is memory writing work reflected the most recently from the time of conflict firstly until the writing work reflected the most lately, in order. For it, recovery order of write_entries is necessary. In other words, TinySTM must be able to calculate the ordering number of write_entry while invoking TM_STORE. Fig. 5 is in progress of list insertion arithmetic operation of HEAPO in multi-thread environment. To control concurrency in multi-thread environment, TinySTM will cognize conflicts by storage of each thread will be located in non-volatile memory. For the recovery, write_entry is numbered in order of inserting to write_set.

Lets assume that system crack is occurred in the Figure situation. Users will recover the system and reuse the data. To use consistent data, ordering number of write_set stored in non-volatile memory is searched. Memory writing work reflected the most recently from the time of conflict is No. 6 and the work reflected the most lately is No.1. Data of hash table can maintain the consistency through the rollback from No.6 to No.1.

V. Evaluation

This section carries out performance evaluation of F-STM. This chapter measures KV data structure record insertion performance of HEAPO through the TUNA, a board of non-volatile memory.

A. Tuna Board

For the experiment of this paper, TUNA board, a board for only non-volatile memory, was used [2]. TUNA board was developed to use non-volatile memory and uses ARM based platform. TUNA board is designed to supply power separated from DRAM Chip and adopted hybrid based computer structure using both volatile memory and non-volatile memory, so its optimal to experiment hardware performance of NVRAM. Also, HEAPO, a middleware platform for non-volatile memory is installed and total 15 stages of write latency can be provided, so its possible to measure performance in various ways.

B. Experiment

HEAPO implemented three types of KV store in non-volatile memory for application programmer. Therefore, We evaluate insert operation performance for each types of KV store(list,hash,btree) about F-STM on Fig. 6.

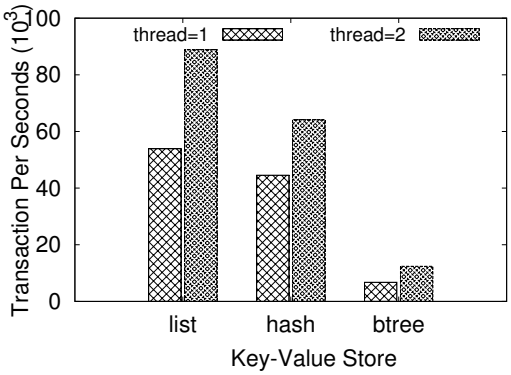

Fig. 6. Performance for each kv store

TinySTM performance is increased linearly as the number of threads is increased in multi-core environment because TinySTM permits free entrance of several threads to critical section for guaranteeing concurrency control. Fig. 6 confirms performance changes by increase of the number of TinySTM threads. The number of threads transaction enters Critical Section is increased and performance increase is confirmed. There are 2 cores of TUNA board for the experiment and the experiment is conducted by increasing the number of threads to 2.

Compared to HEAPO which do not guarantee fail-safe, F-STM lead to degradation of performance. Howewver, HEAPO which have special recovery mechanism for system crash must be needed. Because guaranting transaction's ACID properies is important issue of non-volatile characteristic.

VI. CONCLUSION

This paper locates storage managing TinySTM by threads in non-volatile memory and implements recovery mechanism. Therefore, suggested TinySTM guarantee Consistency and Durability. At present, TinySTM transaction must be directly made by macro provided by TinySTM. Memory reading or writing is same. In Future works, users can be convenient by letting compilers manage memory reading and writing of transaction automatically.

ACKNOWLEDGMENT

This work was supported bythe R&D program MKE/KEIT (No. 10041608, Embedded System Software for New-memory based Smart Device) and the ICT R&D program of MSIP/IITP.[R0601-15-1063, Software Platform for ICT Equipments] and the MSIP(Ministry of Science, ICT&Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2015- H8501-15-1006) supervised by the IITP(Institute for Information&communications Technology Promotion

REFERENCES

[1] Tinystm: A lightweight and efficient software transactional memory implementation in c. http://tinystm.org.

[2] Tuna: The platform for emulating nvm. http://opennvram.org.

[3] BAILEY, K., CEZE, L., GRIBBLE, S. D., AND LEVY, H. M. Operating system implications of fast, cheap, non-volatile memory. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems* (2011), USENIX Association, pp. 2–2.

[4] COBURN, J., CAULFIELD, A. M., AKEL, A., GRUPP, L. M., GUPTA, R. K., JHALA, R., AND SWANSON, S. Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. *SIGPLAN Not. 46*, 3 (Mar. 2011), 105–118.

[5] FELBER, P., FETZER, C., AND RIEGEL, T. Dynamic performance tuning 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.

[6] HERLIHY, M., LUCHANGCO, V., AND MOIR, M. A flexible framework for implementing software transactional memory. *SIGPLAN Not. 41*, 10 (Oct. 2006), 253–262.

[7] HERLIHY, M., LUCHANGCO, V., MOIR, M., AND SCHERER, III, W. N. Software transactional memory for dynamic-sized data structures. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing* (New York, NY, USA, 2003), PODC '03, ACM, pp. 92–101.

[8] HITZ, D., MALCOLM, M., LAU, J., AND RAKITZIS, B. Copy on write file system consistency and block usage, May 10 2005. US Patent 6,892,211.

[9] HWANG, T., JUNG, J., AND WON, Y. Heapo: Heap-based persistent object store. *Trans. Storage 11*, 1 (Dec. 2014), 3:1–3:21.

[10] KRYDER, M. H., AND KIM, C. S. After hard driveswhat comes next? *Magnetics, IEEE Transactions on 45*, 10 (2009), 3406–3413.

[11] NARAYANAN, D., AND HODSON, O. Whole-system persistence. *SIGPLAN Not. 47*, 4 (Mar. 2012), 401–410.

[12] PAYER, M., AND GROSS, T. adaptstm-an online fine-grained adaptive stm system. Tech. rep., Tech. rep., ETH Zurich, http://www. lst. ethz. ch/research/publications/ADAPTSTM 2010. html, 2010.

[13] PEREZ, T. *Evaluation of system-level impacts of a persistent main memory architecture*. PhD thesis, Pontifícia Universidade Católica do Rio Grande do Sul, 2012.

[14] SCHERER, III, W. N., AND SCOTT, M. L. Advanced contention management for dynamic software transactional memory. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2005), PODC '05, ACM, pp. 240–248.

[15] SELTZER, M. I., GANGER, G. R., MCKUSICK, M. K., SMITH, K. A., SOULES, C. A., AND STEIN, C. A. Journaling versus soft updates: Asynchronous meta-data protection in file systems. In *USENIX Annual Technical Conference, General Track* (2000), pp. 71–84.

[16] SHAVIT, N., AND TOUITOU, D. Software transactional memory. *Distributed Computing 10*, 2 (1997), 99–116.

[17] SOULES, C. A., GOODSON, G. R., STRUNK, J. D., AND GANGER, G. R. Metadata efficiency in versioning file systems. In *FAST* (2003), vol. 3, pp. 43–58.

[18] VOLOS, H., TACK, A. J., AND SWIFT, M. M. Mnemosyne: Lightweight persistent memory. *SIGPLAN Not. 47*, 4 (Mar. 2011), 91–104.

[19] WANG, K., AND AMIRI, P. K. Nonvolatile spintronics: perspectives on instant-on nonvolatile nanoelectronic systems. In *Spin* (2012), vol. 2, World Scientific, p. 1250009.

[20] YLÖNEN, T. Shadow paging is feasible.