v1.0

State Machine Protocol of the ThyNVM Checkpointing Schemes

JINGLEI REN, Tsinghua University

1. INTRODUCTION

This document describes the state machine based protocol of the ThyNVM checkpointing schemes [Ren et al. 2015]. We first name all possible locations of different versions of data in Section 2, and then describe states and their transitions in Section 3. Finally we discuss the entry eviction issue in Section 4.

2. ADDRESS SPACE MANAGEMENT

As introduced in our paper [Ren et al. 2015], ThyNVM employs two address translation tables, *Block Translation Table* (BTT) and *Page Translation Table* (PTT), to maintain the mapping between the *physical address space*, which is exposed to the processor, and the *hardware address space*, which is the actual DRAM and NVM device address space managed by the memory controller.

Each physical address has a corresponding static *main hardware address* in NVM. Therefore, locating the main hardware address of a physical address does not need address translation via BTT/PTT. Without loss of generality, we assume the main hardware address equals to the physical address. Meanwhile, any physical address can be *dynamically* mapped to a different hardware address in NVM via BTT/PTT, according to the ThyNVM checkpointing schemes.

To facilitate memory management, we divide the hardware address space into six regions for different uses. Figure 1 depicts the address space organization in ThyNVM. Next we describe the use of each region.

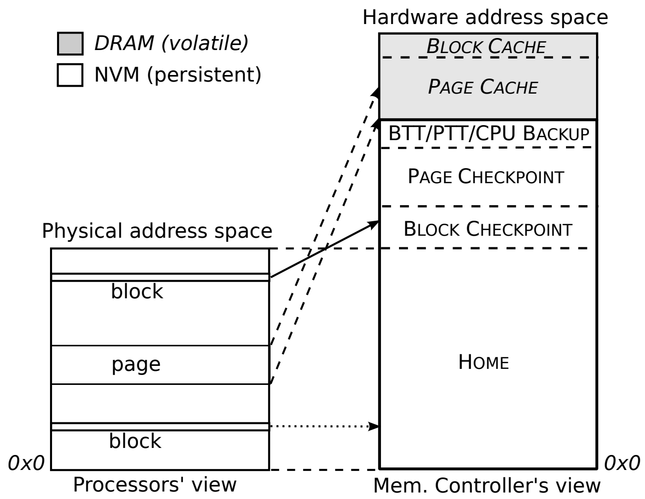

Fig. 1. The physical and hardware address spaces in ThyNVM.

HOME. This NVM region holds the main hardware addresses. Each physical address has a static main hardware address in this region. The size of the region is the same as that of the physical address space.

BLOCK CHECKPOINT. This NVM region is used for BTT to allocate a location for the checkpoint data (C_{last}^{block} or C_{penult}^{block}) at cache-block granularity. In the block remapping scheme, the working copy of a block (W_{active}^{block}) becomes C_{last}^{block} *without data movement* after its associated BTT entry is persistent. Hence, this region also stores W_{active}^{block} before its associated BTT is persistent, so that the data can become C_{last}^{block} without data movement after the BTT entry is persistent.

PAGE CHECKPOINT. This NVM region is used for PTT to store the checkpoint data at page granularity (C_{last}^{page} or C_{penult}^{page}). HOME and PAGE CHECKPOINT store C_{last}^{page} and C_{penult}^{page} in an alternative manner: if C_{penult}^{page} is stored in HOME, C_{last}^{page} gets placed in this region, and vice versa.

PAGE CACHE. A DRAM cache for the working copy of hot pages (W_{active}^{page}). In the checkpointing phase of each epoch, dirty pages in this region are written back as C_{last}^{page} to NVM (in either PAGE CHECKPOINT or HOME as aforementioned).

BLOCK CACHE. This DRAM region stores a temporary working copy of a block in a page that is being checkpointed. When checkpointing overlaps program execution, memory accesses to PAGE CACHE are handled by the block remapping scheme, and remapped to this region if necessary.

BTT/PTT/CPU BACKUP. The BTT and PTT corresponding to C_{last} are persisted in this NVM region during each checkpointing phase, so that the system can restore addresses of the checkpoint data after a crash. The CPU state at the beginning of the checkpointing phase is also stored in this region, and associated with the corresponding BTT and PTT backups. BTT/PTT/CPU backups associated with a checkpoint data copy prior to C_{penult} can be evicted.

Note that BLOCK CHECKPOINT and PAGE CHECKPOINT are described together as Checkpoint Region A in the ThyNVM paper [Ren et al. 2015], and BLOCK CACHE plus PAGE CACHE as Working Data Region.

3. STATE MACHINE FOR ADDRESS TRANSLATION

We use a state machine to decide which location a memory write should access, i.e., to translate the physical address of the memory write to a proper hardware address. Recall that the purpose of such address translation is to prevent data overwriting and maintain three versions of data (the working copy W_{active} , the last checkpoint C_{last} and the penultimate checkpoint C_{penult}). Besides the mapping from a physical address to a hardware address, each BTT entry also holds one of seven states to drive the control flow of the block remapping scheme. PTT reuses the same logic of the state machine but only four of the states. Next we focus on describing the behavior of BTT, and discuss how PTT fits into the model later on. Figure 2 depicts the whole view of the state machine. For clarity, we group states and introduce them one group after another.

3.1. States in Execution

This group of states mainly coordinates the scheme behavior during program execution (without checkpointing). To decide the proper location for a memory write, we need a state to record two pieces of information about the address mapping: (1) Whether the physical address has been written in the current active epoch. If so, we can directly overwrite the corresponding hardware address because memory writes to the same physical address in the same epoch can be coalesced. (2) Whether the hardware address is a main hardware address or in a checkpoint region. We would choose the other if we have to avoid overwriting one of them. Accordingly, there are four states to cover all combinations of the above two alternatives.

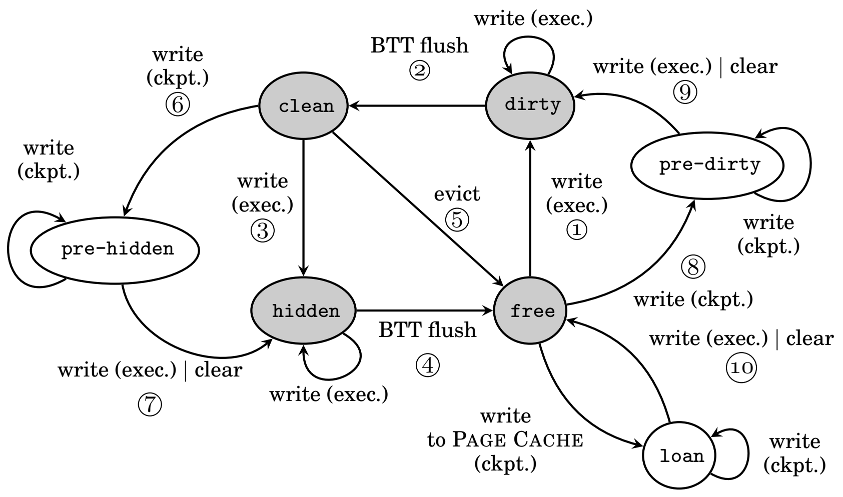

Fig. 2. States of a BTT entry in the block remapping scheme. Each memory write is marked with "exec." or "ckpt.", showing when the memory write comes, during execution (without checkpointing) or during checkpointing, respectively.

— `Free`: An empty entry without a valid address mapping. We also consider a physical address as `free` if it is not included in the BTT. It implies that the physical address has *not* been written in the current active epoch and its last checkpoint C_{last} is located at the main hardware address in HOME. A coming memory write on the physical address must allocate a new entry in BTT to remap the write to BLOCK CHECKPOINT.

— `Clean`: This state implies that the physical address has *not* been written in the current active epoch and its last checkpoint C_{last} is located in BLOCK CHECKPOINT. A coming memory write on the physical address must be remapped by BTT back to the main hardware address in HOME in order to protect the checkpoint data.

— `Dirty`: This state denotes that the physical address has been written in the current active epoch and its corresponding hardware address is in BLOCK CHECKPOINT. The mapping can be used to coalesce further writes on the physical address until the end of the current execution phase. After the execution phase ends, the data at the hardware address shall become the checkpoint data, so the entry has to be backed up in NVM to make this mapping persistent for locating the checkpoint data in case of a system crash.

— `Hidden`: This state denotes that the physical address has been written in the current active epoch and its corresponding hardware address is the main hardware address in HOME. There is actually no address translation because the main hardware address equals to the physical address. However, the entry resides until the end of the current execution phase only to coalesce further writes on the physical address. After the execution phase ends, this entry will be freed.

Example. To illustrate the dynamic transition between these states, we compose an example in Figure 3. We first overlook steps during checkpointing (Step A and B) and simply assume they do not happen. In the example, we show three consecutive epochs, numbered from 0 to 2. Memory writes from Epoch k result in the version v_k of data, where $k = 0, 1, 2$.

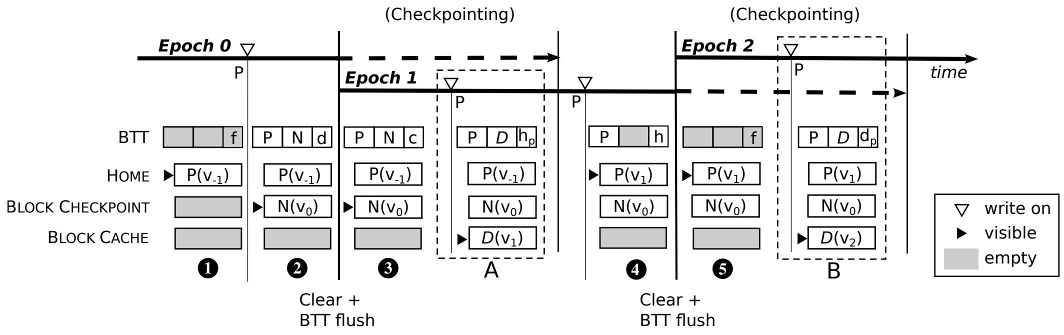

Fig. 3. An example of memory writes to the physical address P across multiple epochs.

❶ Initially, we assume that data at the physical address P is located at its main hardware address in HOME, so it has no corresponding entry in BTT, and occupies no location in BLOCK CHECKPOINT or BLOCK CACHE. The data at P is regarded as of the last checkpoint version (denoted as v_{-1}, i.e., one version before v_0).

❷ In executing Epoch 0, when a memory write on the physical address P arrives, we cannot overwrite the checkpoint data v_{-1} in HOME. Hence a new mapping P-N is added to BTT to redirect the working copy v_0 to a hardware address N in BLOCK CHECKPOINT. We mark this entry as `dirty` (Transition ① in Figure 2) to denote that the physical address has been written in Epoch 0. Any further write on P during the execution phase of Epoch 0 can be coalesced at N.

❸ After the execution phase of Epoch 0, the above `dirty` entry is flushed to BTT/PTT/CPU BACKUP. The state of the entry in BTT becomes `clean` (Transition ②) to indicate that this mapping has been persistent and there are no writes on the physical address yet in the active Epoch 1.

❹ After Epoch 0 is completely checkpointed, we no longer need v_{-1} in the HOME region. Therefore, a new memory write on the physical address P can be remapped to HOME. At the same time, the corresponding BTT entry state becomes `hidden` (Transition ③). This state denotes that the main hardware address in HOME contains the working copy v_1 and any writes on P during the execution phase of Epoch 1 can update the data at the main hardware address. In case of a crash in this step, ThyNVM would roll back to v_0 stored in BLOCK CHECKPOINT.

❺ When Epoch 2 begins, the BTT flush operation clears the `hidden` state and this entry is freed to indicate that the main hardware address in HOME holds the last checkpoint v_1 of the physical address P (Transition ④).

3.2. States in Checkpointing

The second group of states works during checkpointing phases. It includes two states, `pre-hidden` and `pre-dirty`, which handle the following two situations respectively.

— `Pre-hidden`: This state handles the situation when a memory write is received on a `clean` physical address during a checkpointing phase. We cannot simply follow Transition ③ in Figure 2 as during execution, because the data in both HOME and BLOCK CHECKPOINT has to be preserved during checkpointing. Take Step A in Figure 3 for example. Since we are in Epoch 1, data v_0 in BLOCK CHECKPOINT is immutable because it is part of the last checkpoint C_{last} , and so is data v_{-1} in HOME because a full C_{last} is not completed yet - the system would have to roll back to v_{-1} (C_{penult}) if a crash interrupted the checkpointing phase. Therefore, we have to temporarily put the working copy W_{active} (v_1) in another venue, BLOCK CACHE, and accordingly mark the special state as `pre-hidden` (Transition ⑥). `Pre-hidden` acts as a stepping stone between `clean` and `hidden`. It will live until the end of the execution phase of the current active epoch, then, if not transited to `hidden` by a memory write hit (resulting in the state of Step ❹ in Figure 3), it is explicitly cleared to `hidden` (Transition ⑦). In either case, as the checkpoint of Epoch 0 (C_{last}) is done, v_{-1} (C_{penult}) is safely overwritten by W_{active} and accordingly the location in BLOCK CACHE is revoked.

— `Pre-dirty`: The state handles the situation when a memory write is received on a `free` physical address during a checkpointing phase. For similar reasoning with `pre-hidden`, we cannot directly follow Transition ① as in execution. Take Step B in Figure 3 for example. When the checkpoint of Epoch 1 (C_{last}) is being created, not only should we preserve v_1, but also v_0 (C_{penult}) because C_{penult} is the only complete consistent checkpoint we have. Therefore we cannot overwrite HOME (v_1) or BLOCK CHECKPOINT (v_0). Instead we have to put data v_2 in BLOCK CACHE, and add a new `pre-dirty` entry to BTT to record this address mapping (Transition ⑧). `Pre-dirty` is either transited to `dirty` by a memory write hit in the following execution phase, or cleared to `dirty` at the end of that phase (Transition ⑨).

Since `hidden` and `dirty` states are cleared before BTT is flushed to NVM at the beginning of each checkpointing phase, they are never encountered by memory writes during checkpointing. Hence there are no other situations to handle.

3.3. State for Dual-Scheme Cooperation

There is one last state for cooperation between the dual schemes. When the page writeback scheme checkpoints dirty pages, individual memory writes to these pages are redirected by the block remapping scheme to BLOCK CACHE. We do not mix these BTT entries with the above states/transitions, so the entries are denoted as `loan`. `Loan` entries eventually move data back to original pages after the checkpointing is finished and pages are open to direct modification again, following Transition ⑩ in Figure 2.

Finally, PTT uses states `free`, `clean`, `dirty` and `hidden` to handle page writebacks from PAGE CACHE during the checkpointing phase of the page writeback scheme. The states are reused in such a way that, when performing state transitions, all such writebacks are regarded as if it is the *execution* phase in Figure 2. Consequently, we do not involve `pre-dirty`, `pre-hidden` or `loan` in PTT. Of course, the page writeback scheme stores checkpoint data in PAGE CHECKPOINT instead of BLOCK CHECKPOINT as in Section 3.1.

4. ENTRY EVICTION AND SHRINKING

If there is no `free` entry in BTT to accommodate a coming memory write whose physical address does *not* currently exist in BTT, we have to evict a `hidden` or `clean` entry to make a `free` entry for the memory write.

`Hidden` entries are anyway to be evicted at the end of the current execution phase, so they can be safely removed without introducing extra costs.

`Clean` entries can be safely evicted (Transition ⑤ in Figure 2) because they do not track new updates in the current active epoch (otherwise either Transition ③ or Transition ⑥ would have happened). However, the data block referenced by an evicted `clean` entry (located in BLOCK CHECKPOINT) has to be moved back to its main hardware address, so that after eviction, the data block can be accessed without address translation by the BTT entry. But we are not always allowed to overwrite the original data block at the main hardware address, which is of the version C_{penult} (see the state of Step ❸ in Figure 3 for example). If the eviction happens during checkpointing, we cannot overwrite the data block of C_{penult} at the main hardware address because C_{last} is not fully completed, so we have to exchange the data block with the one in BLOCK CHECKPOINT, preserving both data blocks. If the eviction happens out of a checkpointing phase, we can safely discard C_{penult} and simply overwrite on the main hardware address. Overall the eviction operation is costly, in terms of both memory bandwidth utilization and latency.

Fortunately, there is a favorable phenomenon, which we call *entry shrinking*, in our scheme: if a physical address is written in consecutive epochs, the BTT entry it occupies is naturally freed (e.g., from Step ❶ to Step ❺ in Figure 3). Therefore, temporal locality across epochs can reduce the number of entries in use and consequently the chance of entry eviction.

ACKNOWLEDGMENTS

The author would like to thank Prof. Samira Khan at University of Virginia and Prof. Jishen Zhao at University of California, Santa Cruz for refining the writing of this document.

REFERENCES

Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu. 2015. ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems. In *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-48)*. ACM, New York, NY, USA. DOI:http://dx.doi.org/10.1145/2830772.2830802