Zero-Overhead NVM Crash Resilience

Faisal Nawab[*,†], Dhruva Chakrabarti[†], Terence Kelly[†], Charles B. Morrey III[†]
[*]*CS Dept., UC Santa Barbara,* [†]*HP Labs, Palo Alto, CA*

Introduction

Byte-addressable non-volatile memory (NVM) allows fine-grained in-place update of durable data. Failures can corrupt application data. Realizing the full value of NVM requires mechanisms to preserve application data integrity in the presence of failures.

NVM transaction mechanisms [1, 2, 7] prevent failures during updates from corrupting data, but such mechanisms carry substantial performance overheads. Our new alternative guarantees consistent recovery of application data following failure and has *zero* overhead during failure-free operation. Below we outline our new approach, and evaluate its effectiveness. Our tech report provides more detail [5].

NVM Transaction Overheads

NVM transaction overhead largely stems from forcing data from volatile CPU caches to NVM (e.g., via cache line flushes). We can eliminate the need to force data into NVM by borrowing an insight from whole-system persistence: *flush-on-failure* can replace flush-as-you-go. We need not insist that data *has reached* NVM during failure-free operation if instead we are assured that the data *will reach* NVM in the event of failure [5].

Tolerating power outages requires sufficient standby power for orderly system shutdown. Fortunately, The time and energy required to flush CPU caches to NVM is orders of magnitude smaller than would be required to dump volatile DRAM to block storage. Narayanan & Hodson report that a typical computer's power supply contains sufficient residual energy for this task [4]. Tolerating OS kernel panics requires the kernel panic handler to flush all CPU caches to NVM before halting the system. Tolerating process crashes can be done by file-backed memory mappings. Cached modifications will eventually find their way down through the CPU cache and page cache to the backing file

Zero-Overhead Atomic Updates

Our main contribution is a crash resilience technique that avoids all of the performance overheads of existing NVM transaction mechanisms because it performs no logging. Our technique combines flush-on-failure with a class of *multi-threaded isolation* mechanisms into a *consistent recovery* mechanism.

Our technique is to combine flush-on-failure with non-blocking algorithms: Consistent data recovery will always succeed following the abrupt termination of a program that manipulates NVM via non-blocking algorithms on a system with flush-on-failure support. A non-blocking algorithm ensures that an observer will see a "sane" state of memory and can make useful progress. Thus, recovery code has a consistent view of application data and can resume correct execution.

References

[1] D. Chakrabarti et al. Atlas: Leveraging Locks for NVM consistency. In *OOPSLA*, 2014.

[2] J. Coburn et al. NV-Heaps: Making persistent objects fast & safe with NVM. In *ASPLOS*, 2011.

[3] K. Fraser and T. Harris. Concurrent Programming without locks. *ACM TOCS*, 25(2), May 2007.

[4] D. Narayanan and O. Hodson. Whole-System persistence. In *ASPLOS*, 2012.

[5] F. Nawab et al. Procrastination Beats Prevention. Technical Report HPL-2014-70, HP Labs, 2014. Submitted to EDBT.

[6] S. Pelley et al. Memory Persistency. In *ISCA*, 2014.

[7] H. Volos et al. Mnemosyne: Lightweight persistent memory. In *ASPLOS*, 2011.

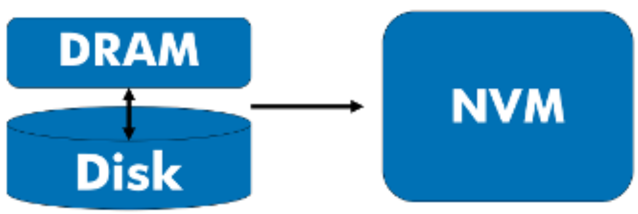
Zero-Overhead NVM Crash Resilience

Faisal Nawab, Dhruva Chakrabarti, Terence P. Kelly, Charles B. Morrey III

Abstract

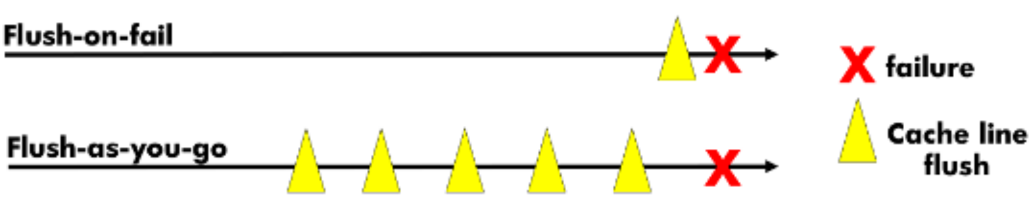
Byte-addressable non-volatile memory (NVM) allows fine-grained in-place update of durable data. NVM transaction mechanisms prevent failures during updates from corrupting data, but such mechanisms carry substantial performance overheads. Our new alternative for high-performance multi-threaded software guarantees consistent recovery of application data following failure and has zero overhead during failure-free operation. Our approach preserves application data integrity in crash-injection experiments and its performance rivals state-of-the-art NVM transactions.

Non-Volatile Memory (NVM)



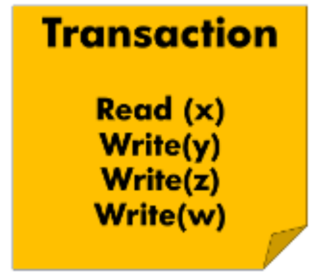
- byte-addressable
- Non-volatile
- Access latency
- Bandwidth
- Scaling

Flush-on-fail vs. flush-as-you-go



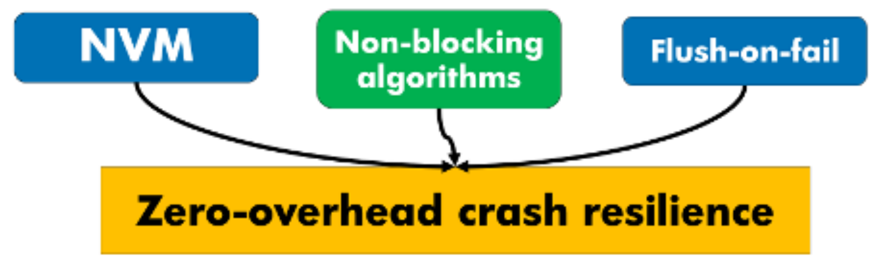
NVM transactions overhead largely stems from cache line flushing

NVM transactions overhead



- A failure while a transaction is executing will leave the data in an inconsistent state even if flush-on-fail is used.
- Undo logging is used to rollback unfinished transactions.
- **Logging is a major source of overhead after eliminating as-you-go cache line flushing**

Zero-overhead crash resilience



Performance (million iter/sec)

computer	No Atlas	Log only	Log+flush	Non-blocking
ENVY Phoenix 800 Desktop	3.66	2.36	1.58	2.54
DL580 Gen8 Server	2.13	1.50	1.06	2.00

References

[1] Chakrabarti, Dhruva R., Hans-J. Boehm, and Kumud Bhandari. "Atlas: Leveraging Locks for Non-volatile Memory Consistency." OOPSLA (2014).

[2] Faisal Nawab, Dhruva Chakrabarti, Terence P. Kelly, Charles B. Morrey III. "Procrastination Beats Prevention: Timely Sufficient Persistence for Efficient Crash Resilience", EDBT (2015).