Managing Non-Volatile Memory in Database Systems

A review by
Apaar Shanker

DATA ANALYTICS
USING DEEP LEARNING
GT CS 8803 // FALL 2018 //

CREATING THE NEXT®

Paper under review

Managing Non-Volatile Memory in Database Systems

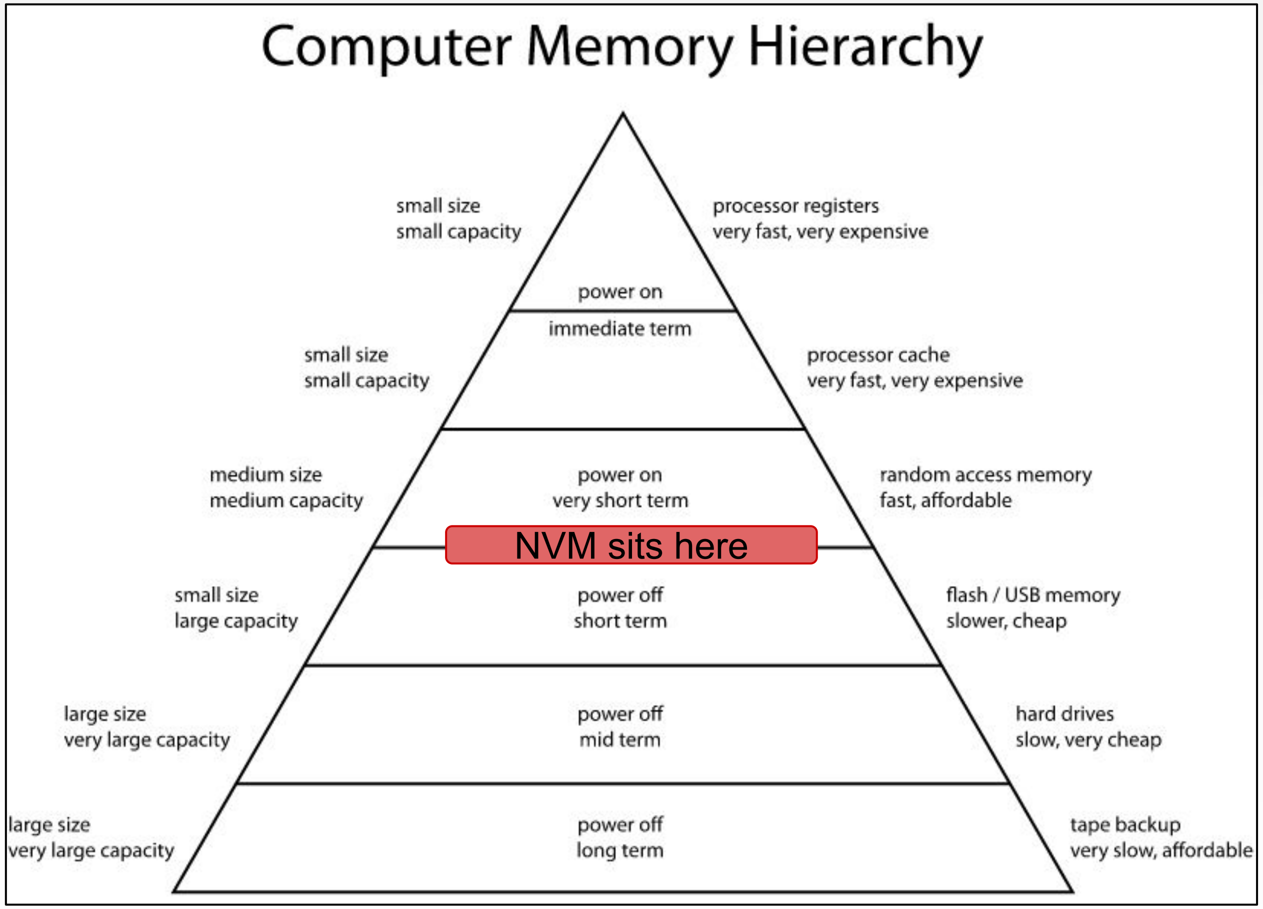
Authors: Alexander van Renen[1], ViKtor Leis, Alfons Kemper[1], Thomas Neumann[1], Takushi Hashida[2], Kazuichi Oe[2], Yoshiyasu Doi[2], Lilan Harada[2], Mitsuru Sato[2]

[1]Technische Universität München, [2]Fujitsu Laboratories

Publication: SIGMOD ‘18

doi:https://doi.org/10.1145/3183713.3196897

Salient Aspects of the Computer Memory Hierarchy



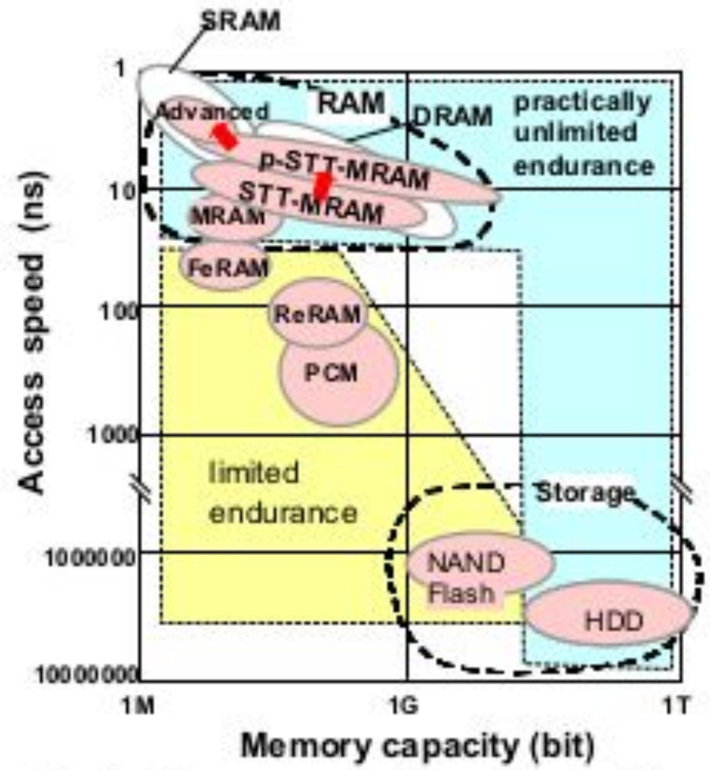


Fig.1. Memory capacity and speed for various nonvolatile memories recently developed.

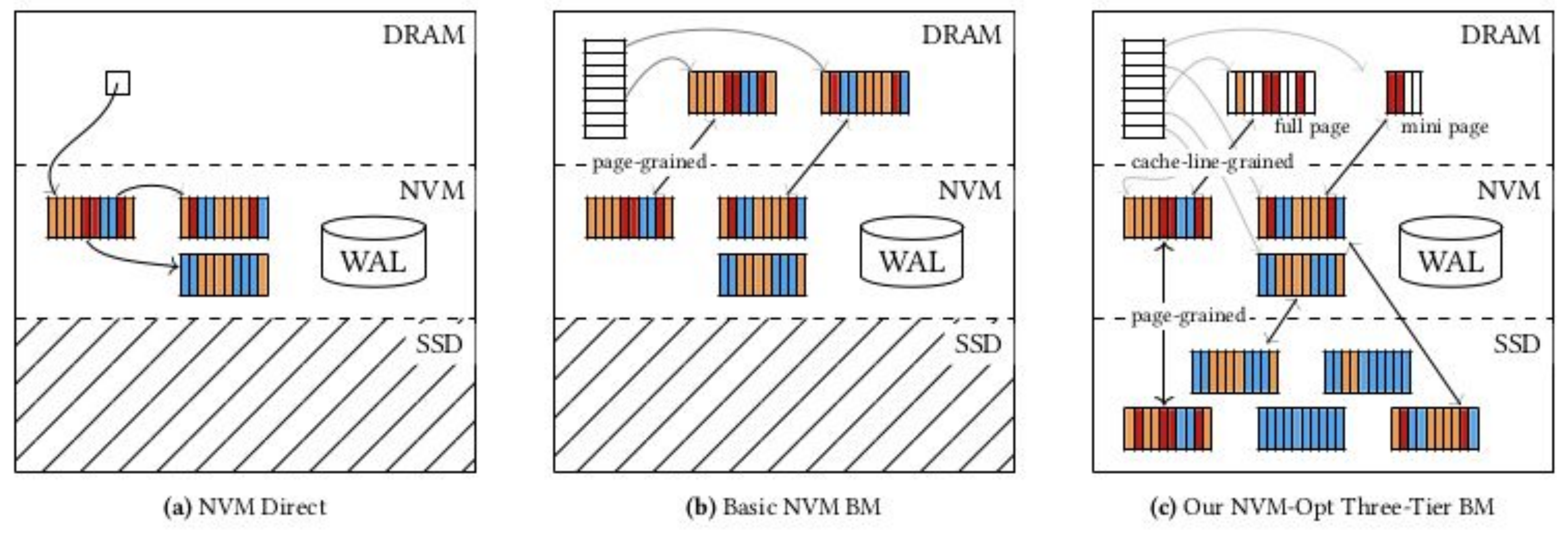
https://en.wikipedia.org/wiki/Memory_hierarchy

DOI: 10.1109/ASPDAC.2014.6742851, Fujita et al. 2014

Objective of the Paper

This paper evaluates the current art and demonstrate a new approach for integrating NVM into the storage layer of database systems.

Non Volatile Memory Based Architectures

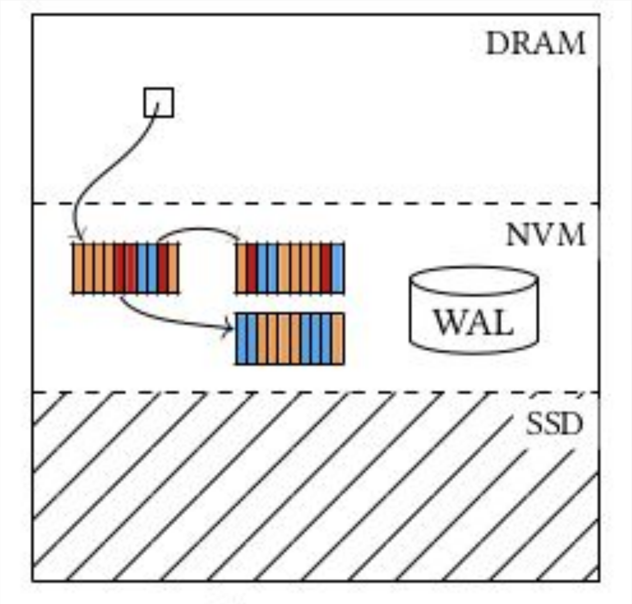


(a) NVM Direct (b) Basic NVM BM (c) Our NVM-Opt Three-Tier BM

B.M : Buffer Manager

ref 1.) Alexander Van Renen et al. 2018

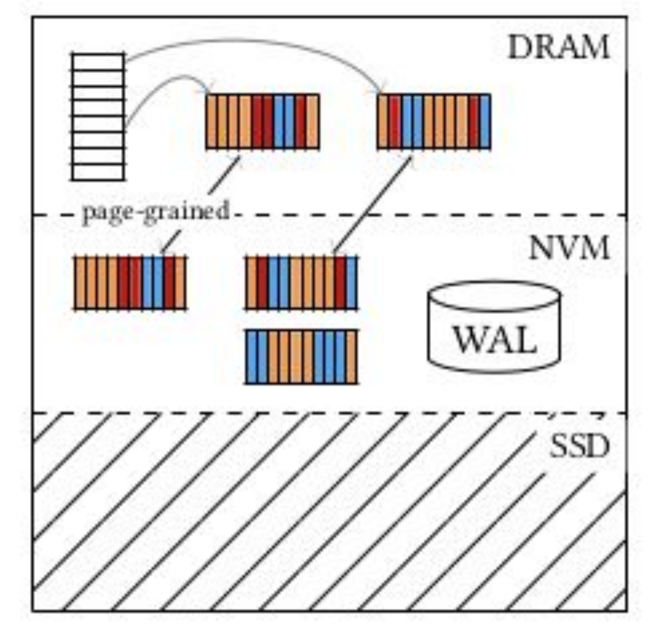
NVM Direct



(a) NVM Direct

- NVM Direct systems were investigated by Arulraj et al.
- Levarages byte addressability of NVM
- Features
  - The design keeps all data in NVM
  - DRAM is only used for temporary data and to keep a reference to NVM data
- Advantages
  - minimalist log (containing only in-flight operations) ensures recovery is very efficient
  - read operations are very simple because a tuple can be directly requested from the NVM.
- Downsides
  - Higher latency of NVM compared to DRAM leads to difficulties in achieving a very high transaction throughputs
  - Doing I/O on NVM directly wears out limited NVM endurance, leading to hardware failures
  - Difficulty in programming database engines for NVM as any modification to is potentially persisted, and can lead to concurrency related problems.

Basic NVM Buffer Manager



(b) Basic NVM BM

- Kimura et al. proposed using a database managed DRAM as a cache in front of NVM

- Similar to the commonly used notion of a buffer manager between a volatile memory (RAM) and SSD

- Features
  - All pages stores on the persistent layer (NVM)
  - DRAM acts as a software managed buffer/cache layer.
  - Transactions operate by accessing pages after loading them onto the buffer pool in DRAM

- Advantages
  - DRAM comparable latency for accessing data in the buffer pool
  - limits read/ write operation on NVM increasing hardware endurance

- Downsides
  - accessible a tuple not present in the buffered pages, requires loading an entire page onto DRA, failing to leverage byte addressability

  - System is optimized for workloads fitting into DRAM only - and does not scale to workloads on larger datasets which require accessing NVM resident data frequently as well.

Key Techniques in Current Approach

- Cache-Line-Grained Pages
- Mini Pages
- Pointer Swizzling

Cache-Line-Grained Pages

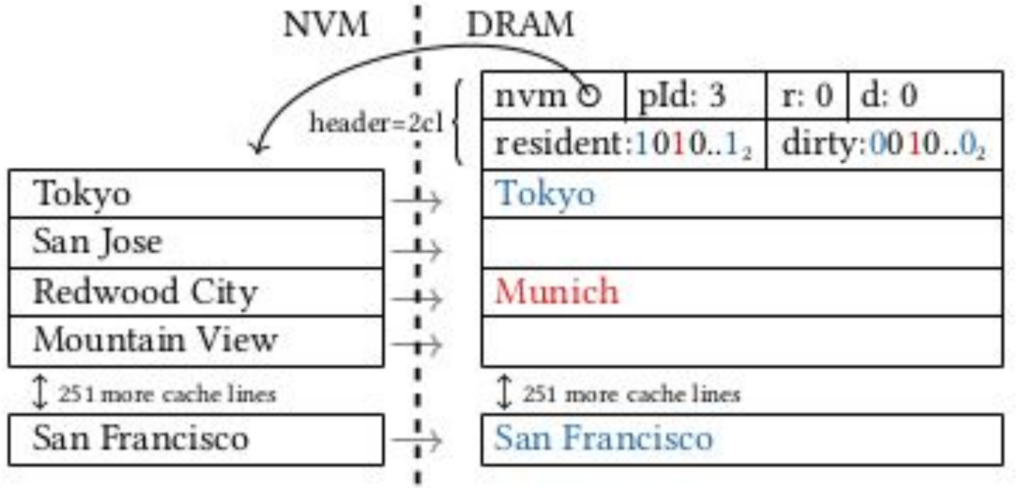


Figure 3: Cache-Line-Grained Pages – The bit masks indicate which cache lines are resident and which are dirty.

- Low nvm latency allows extraction of specific cache-lines rather than entire pages.
- Allows targeted extraction of “hot” data objects from otherwise cold page.
- Buffer manager allocates a page in DRAM without loading data from NVM
- Upon specific transaction request - buffer manager retrieves corresponding cache lines of the page.

- Drawbacks
  - cache-line-grained access is more difficult to program compared to more traditional page-based approach.
- A hybrid approach is adopted where only specific operations such as insert, look-up, delete; that get most benefit from cache-line-grained access are implemented as such.

Mini Pages

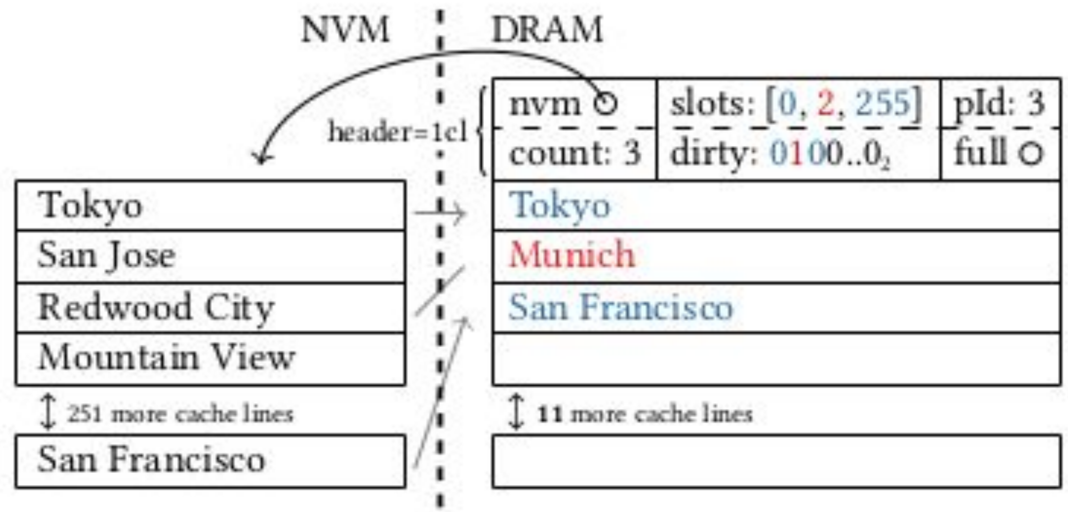
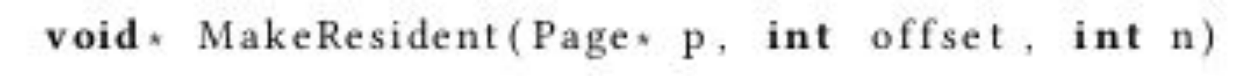


Figure 4: Mini Pages – The `slots` array indicates which cache lines are loaded (max 16). If promoted, `full` points to the full page.

- Allocating space for a full page, even when only few tuples are required, wastes valuable DRAM space
- Solution: A mini page that can store upto 16 cache lines
- An additional "slots" array stores the line id for an item in the original page
- In order to resolve the issue of offset, following function prototype is used.



```
void* MakeResident(Page* p, int offset, int n)
```

When a mini page does not have enough memory to serve a request, it is promoted to a full page.

Pointer Swizzling

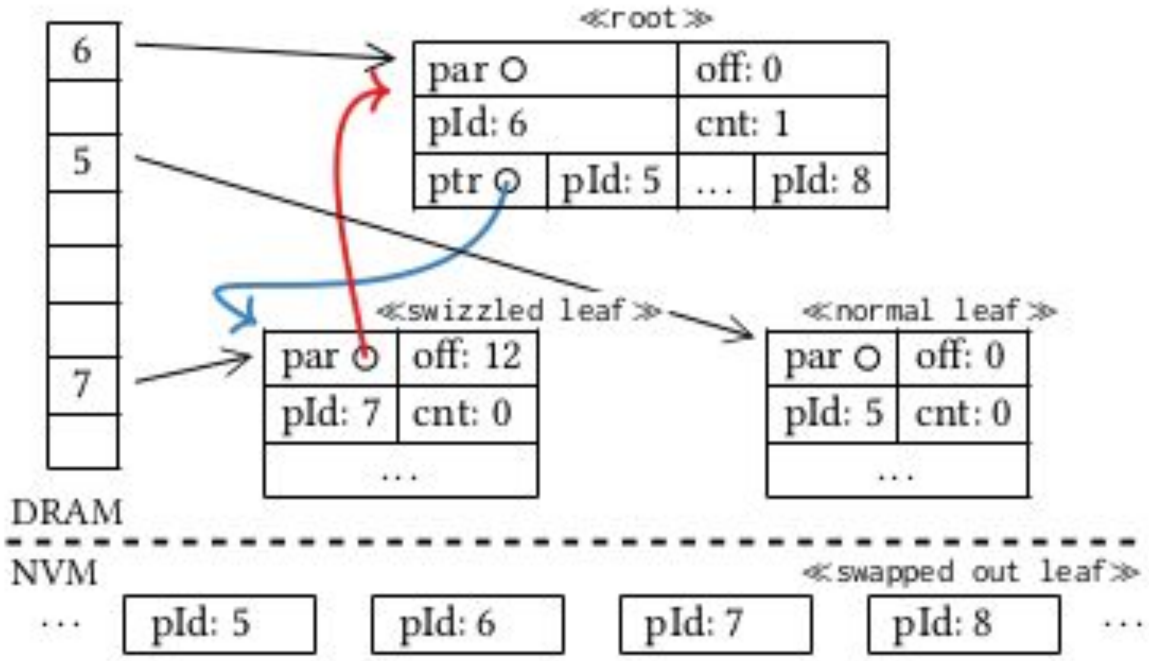


Figure 5: Pointer Swizzling – A B-tree with a root (pId: 6) and three child pages: A swizzled page (pId: 7), a normal DRAM page (pId: 5) and a page currently not in DRAM (pId: 8).

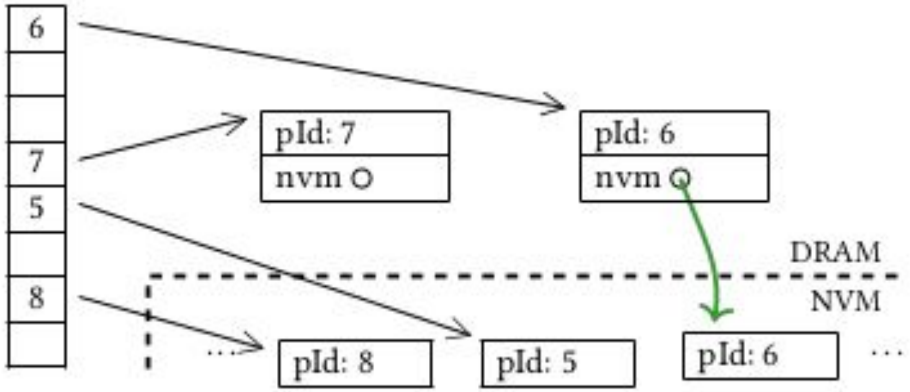


Figure 7: Single-Table Mapping – Using one hash table for DRAM and NVM-resident pages eliminates most overhead for managing the SSD layer. The hash table entries are identified by their location in memory (DRAM or NVM).

Design Outline

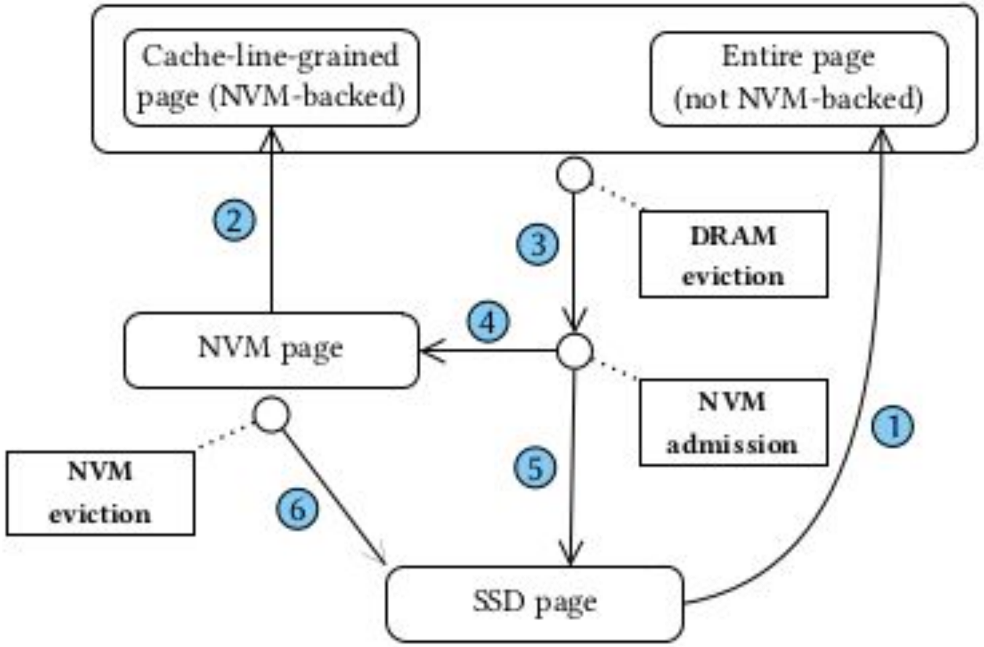


Figure 6: Page Life Cycle – There are five possible page transitions and the three critical decisions (DRAM eviction, NVM admission, and NVM eviction).

- A 3-tier buffer management is implemented, which incorporates ssd as well, apart from DRAM and NVM.
- Addition of SSD - while not improving latency is important for management of large datasets.
- In current set-up the very cold data is stored in SSD.
- Initially, all new-pages start on SSD. On transaction request page is first directly loaded to DRAM and then relegated to NVM or SSD based on decisions.
  - DRAM eviction
  - NVM admission
  - NVM eviction
    - clock algorithm

Performance Evaluation

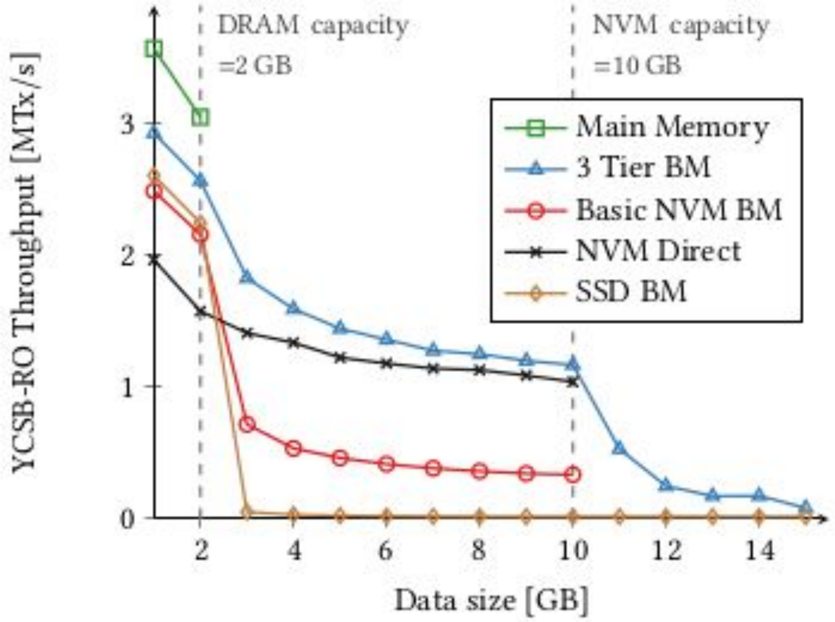


Figure 8: YCSB-RO – Performance for varying data sizes on read-only YCSB workload. The capacity of DRAM, NVM, and SSD is set to 2 GB, 10 GB, and 50 GB, respectively.

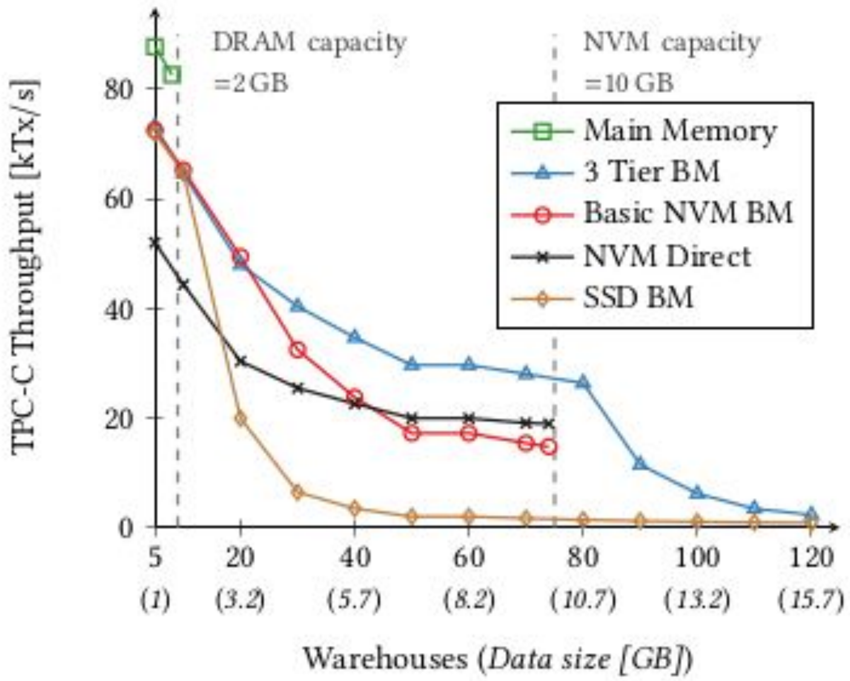


Figure 9: TPC-C – Performance in TPC-C for an increasing number of warehouses. The capacity of DRAM, NVM, and SSD is set to 2 GB, 10 GB, and 50 GB, respectively.

- ❖ YCSB is a key-value store benchmark framework
- ❖ Only point look up operations considered

- ❖ TPC-C is considered the industry standard for benchmarking transactional database systems.
- ❖ It is an insert-heavy workload that emulates a wholesale supplier.

Performance Evaluation across Architectures

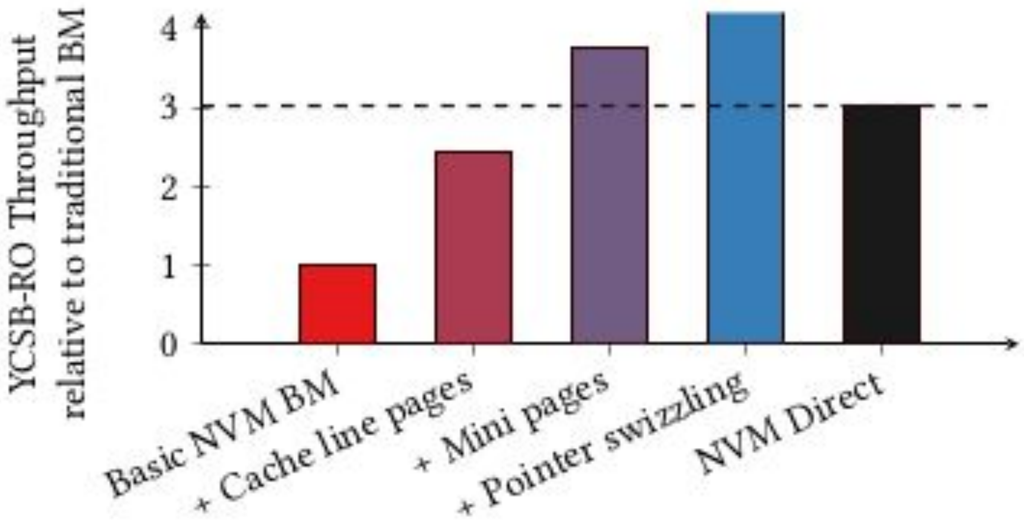


Figure 10: Performance Drill Down – Effect of proposed optimizations relative to a traditional buffer manager on NVM (YCSB-RO with 10 GB of data, read only, 2 GB DRAM, and 10 GB NVM).

Evaluation w.r.t NVM hardware characteristics

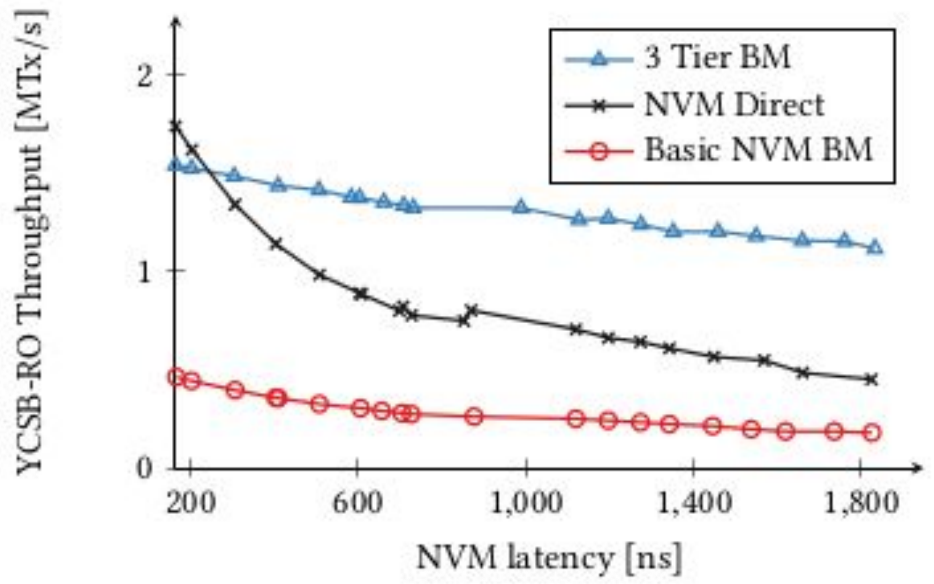


Figure 12: NVM Latency – The impact of varying NVM latencies on the YCSB-RO performance (YCSB with 10 GB of data, read only, 2 GB DRAM, and 10 GB NVM).

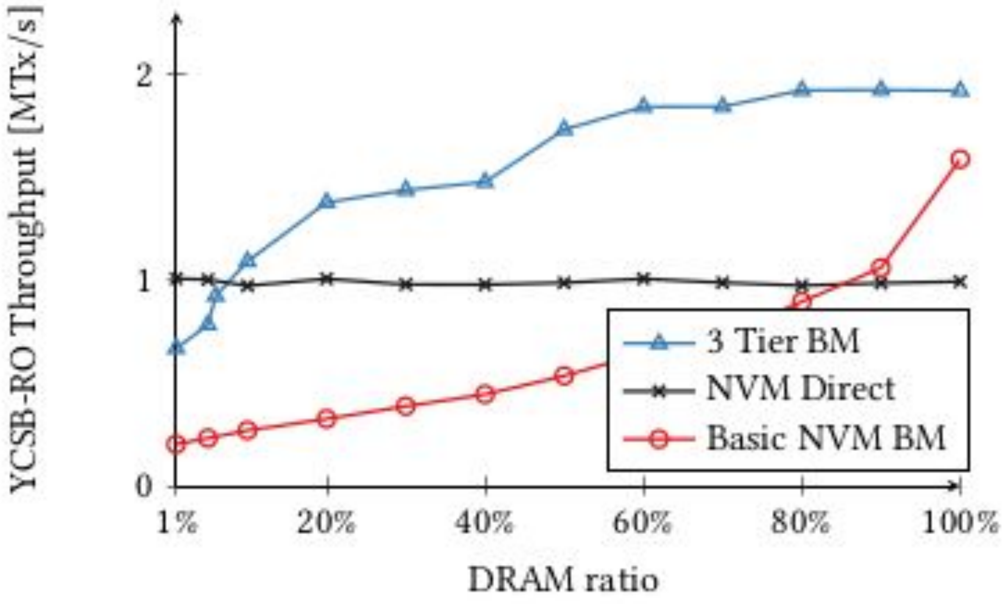


Figure 13: DRAM Buffer Size – YCSB-RO performance for varying amounts of DRAM and a fixed NVM capacity (YCSB with 10 GB of data, read only and 10 GB NVM).

Comments

- Pointer swizzling could compromise data integrity through malicious or unwitting actors
- OS level optimizations not considered.
- Tradeoff between performance improvement and usability? - are these only one time programmer costs?
- What are the other metrics for performance other than throughput? Any economic metrics out there?

References

[1] van Renen A, Leis V et al (2018) Managing non-volatile memory in database systems. SIGMOD ’18, pp 1541–1555

[2] Götze P, van Renen A (2018) Data management on non-volatile memory: A perspective, Datenbank Spektrum (2018) 18:171–182