SMASH: A Common Storage Engine for Modern Memory and Storage Hierarchies

SPP 2377 Annual Meeting 2023

Funded by DFG Deutsche Forschungsgemeinschaft German Research Foundation

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
INF
DZHW Deutsches Zentrum für Hochschul- und Wissenschaftsforschung

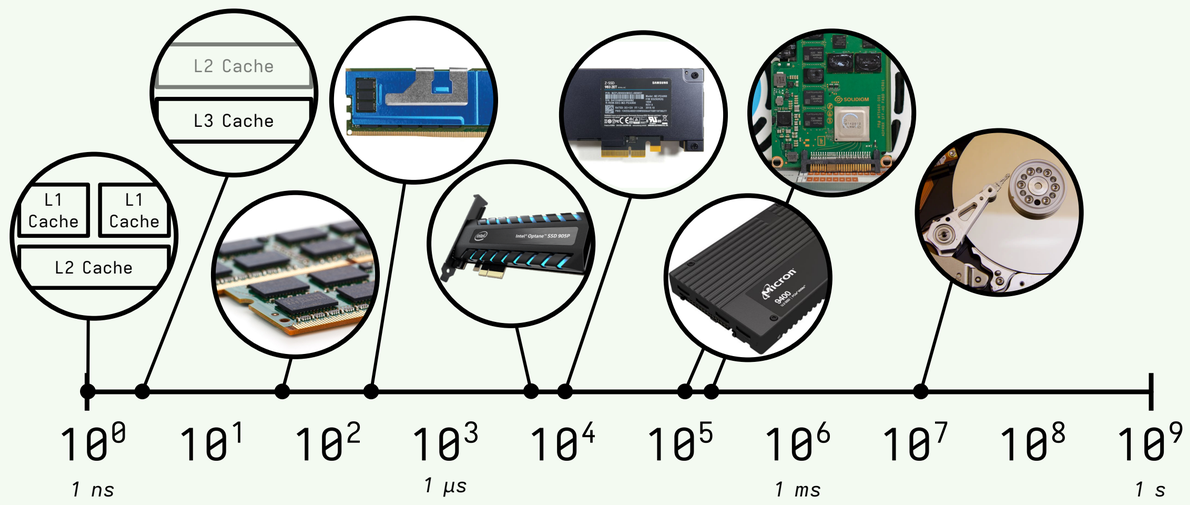
Heterogeneous Use Cases + Hardware

- Scientific research requires data-intensive analyses
- Workflows rely on varying performance characteristics
- Storage technology depending performance characteristics (e.g., persistence, granularity, capacity, latency)
- Database management systems and file systems have to handle heterogeneous storage which accommodate workflows

	Storage	Latency	Read Gran.	Write Gran.	Read Thr.	Write Thr.
volatile	DRAM	≈ 100 ns	64 B		≈ 90 GB/s	≈ 50 GB/s
non-volatile	NVRAM	≈ 1,000 ns	64 B		≈ 30 GB/s	≈ 15 GB/s
non-volatile	NVMe SSD	≈ 10,000 ns	4 KiB	1–8 MiB†	≈ 3.5 GB/s	≈ 2.5 GB/s
non-volatile	SATA SSD	≈ 100,000 ns	4 KiB	1–8 MiB†	≈ 550 MB/s	≈ 500 MB/s
non-volatile	HDD	≈ 10,000,000 ns	4 KiB		≈ 250 MB/s	
non-volatile	SMR HDD	≈ 100,000,000 ns	4 KiB		≈ 200 MB/s	

- **Goal 1: Intelligent data placement and retrieval**
  - Utilize different memory and storage technologies optimized to their characteristics
- **Goal 2: Native data transformations**
  - Hardware-accelerated compression to efficiently decrease data volume
- **Goal 3: Use-case universality and reusability**
  - Building block and prototypical software library for many workloads

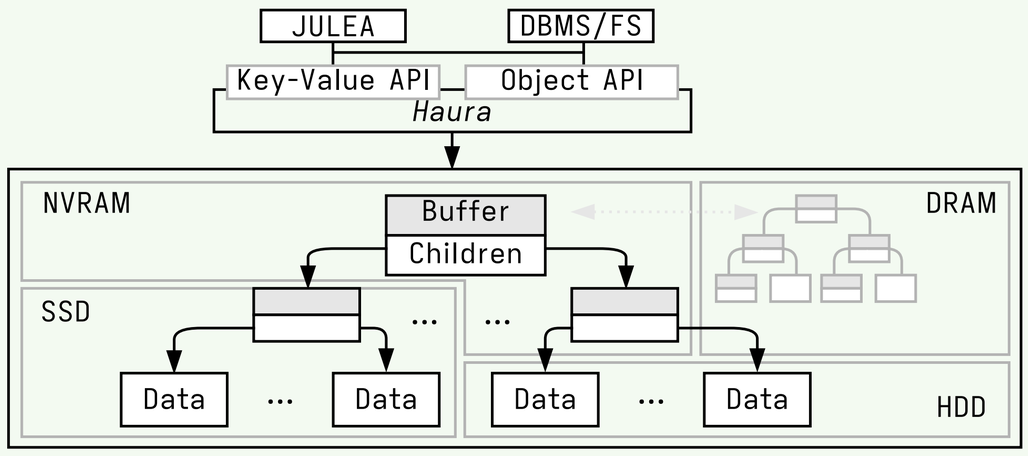
Storage Hierarchy



Access latency relative to magnitude in nanoseconds

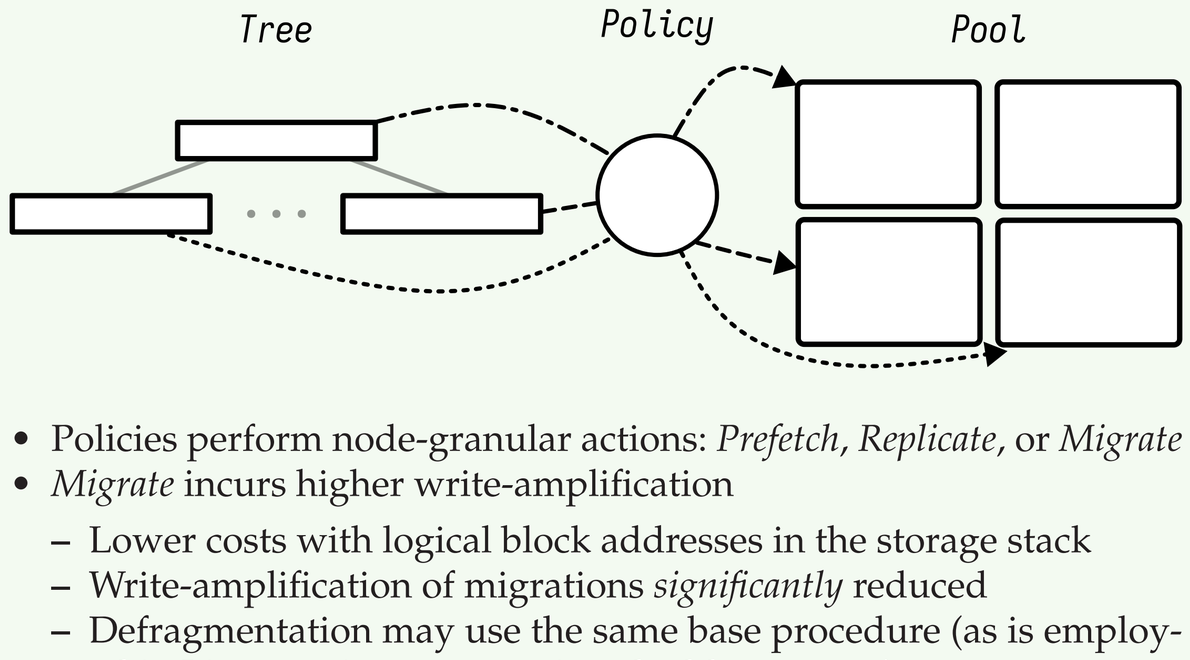
- Wide range of technologies and protocols in use; each exhibiting performance for certain data niches
- *Challenge*: Direct data efficiently to appropriate devices
  - Technologies offer optimal utilization in different access scenarios
  - For example: random workflows to NVRAM/NVMe SSD
- *Focus*: DBMS & HPC use cases
  - Both handle high volumes and traffic of data

SMASH Architecture



- *Hierarchical B^ε-trees*
  - Optimization per tier can be made within a single tree
  - Modifications are buffered in nodes with determinant ε
  - Write-optimized data structure for block devices
- API: *Key-Value* and *Object* interface
  - Supports most semantics required in DBMS and HPC domains

Placement Decisions



- Policies perform node-granular actions: *Prefetch*, *Replicate*, or *Migrate*
- *Migrate* incurs higher write-amplification
  - Lower costs with logical block addresses in the storage stack
  - Write-amplification of migrations *significantly* reduced
  - Defragmentation may use the same base procedure (as is employed in Linux main memory virtual address space)
  - NVRAM makes this feasible with small granularity and alleviates consistency concerns
- Messages propagate information to actors, their hot paths isolated
- *Exchangable Policies* allow for workflow-dependent adjustments
- *Policy Evaluation* in separate discrete-event simulation for fast feedback

Work Packages

- **WP1: Common Storage Engine** ◎
  - 1.1 Data structures and management ✓
  - 1.2 Data placement and migration ◎
  - 1.3 Application programming interface ✓
- **WP2: Data Transformations**
  - 2.1 Data transformation algorithms
  - 2.2 External data transformation
  - 2.3 Hardware acceleration
- **WP3: Use Cases and Evaluation** ◎
  - 3.1 HPC applications and workflows ◎
  - 3.2 Database applications and workflows ◎
  - 3.3 Robustness tests ◎

NVRAM-optimized Data Structures

- The engine leverages NVRAM to offer low-latency read and write operations
- Storage-optimized data structures are used for B^ε-tree nodes
- An NVRAM-optimized hash table is used to cache recent reads
- NVRAM's direct connection to memory bus allows for partial or more granular reads/writes on the nodes in NVRAM
- Any processing on the nodes on block storage requires them to be in DRAM
- The benefits of granular reads on the tree nodes in NVRAM are illustrated in Figure 1

Figure 1: The data in the nodes on NVRAM, DRAM, and SSD NVMe is accessed in different settings. It includes accessing the data partially, such as reading only `int`s and `substring`s (the sizes are mentioned on the horizontal axis). The cost for the nodes on SSD NVMe is high, as the whole node is required to be fetched into DRAM irrespective of the case.

Contact

Sajad Karim, Michael Kuhn, Gunter Saake, Johannes Wünsche
E-mail: {skarim,mkuhn,saake,jwuensch}@ovgu.de
University of Magdeburg

David Broneske
broneske@dzhw.eu
DZHW Hannover

Follow us

https://smash-spp2377.github.io

https://github.com/julea-io/haura