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

SPP 2377 Kickoff

Heterogeneous Use Cases Meet Heterogeneous Hardware

- Scientific research is data-intensive
- Database management systems and file systems have to handle heterogeneous storage
- Different performance characteristics (e.g., persistence, granularity, capacity, latency)

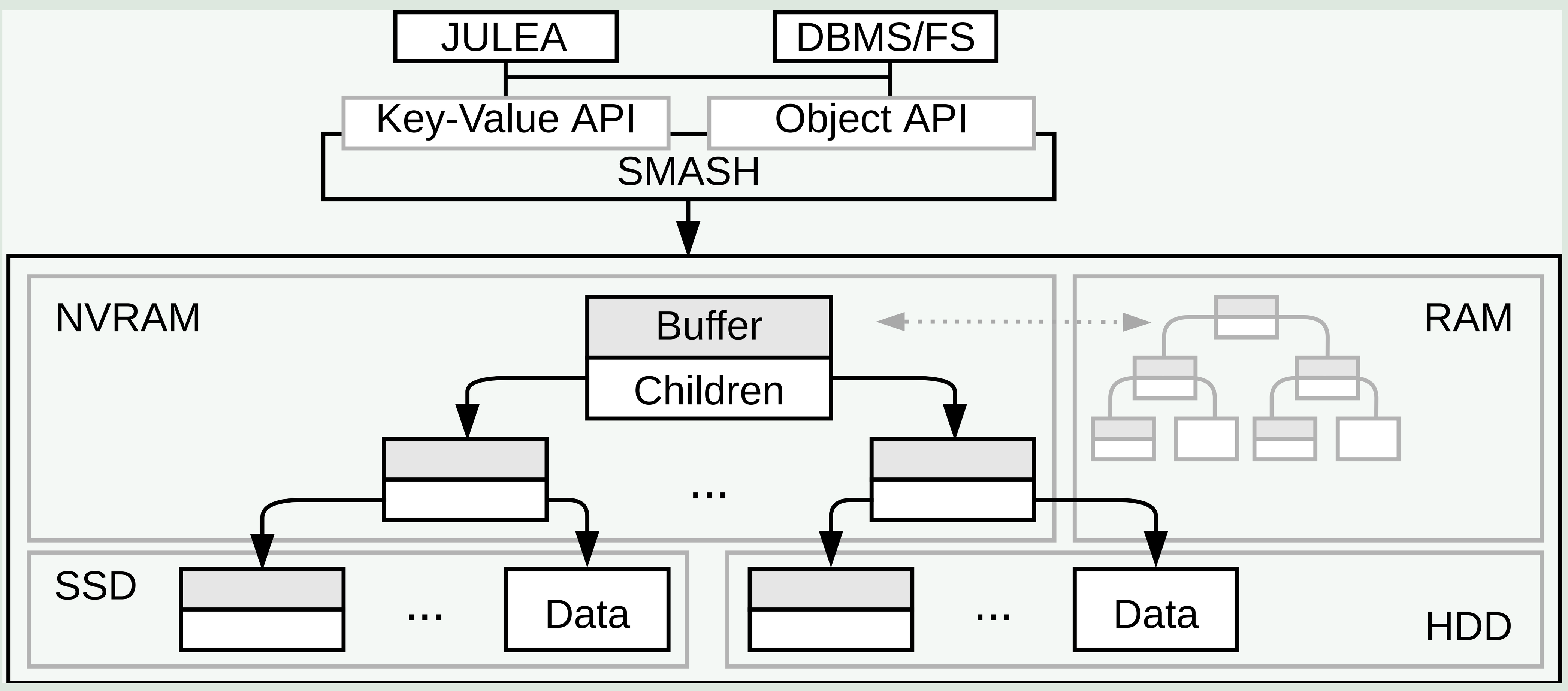
	Storage	Latency	Read Gran.	Write Gran.	Read Thr.	Write Thr.
volatile	DRAM	≈ 100 ns	64 B		≈ 90 GB/s	≈ 50 GB/s
non-volatile	NVRAM	≈ 1,000 ns	64 B		≈ 30 GB/s	≈ 15 GB/s
	NVMe SSD	≈ 10,000 ns	4 KiB	1–8 MiB†	≈ 3.5 GB/s	≈ 2.5 GB/s
	SATA SSD	≈ 100,000 ns	4 KiB	1–8 MiB†	≈ 550 MB/s	≈ 500 MB/s
	HDD	≈ 10,000,000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

- **Goal 1: Intelligent data placement and retrieval**
  - Make use of different technologies, keep characteristics in mind
- **Goal 2: Native data transformations**
  - Hardware-accelerated compression to decrease data volume
- **Goal 3: Use-case universality and reusability**
  - Building block and prototypical software library for many workloads

Storage Hierarchy

- Top tiers are volatile
  - Can lead to data losses or inconsistencies
- Challenge: Map accesses to appropriate tier
  - For example, random workflows better handled by NVRAM/NVMe
- Focus on DBMS and HPC use cases
  - Both handle huge volumes of data

SMASH Architecture



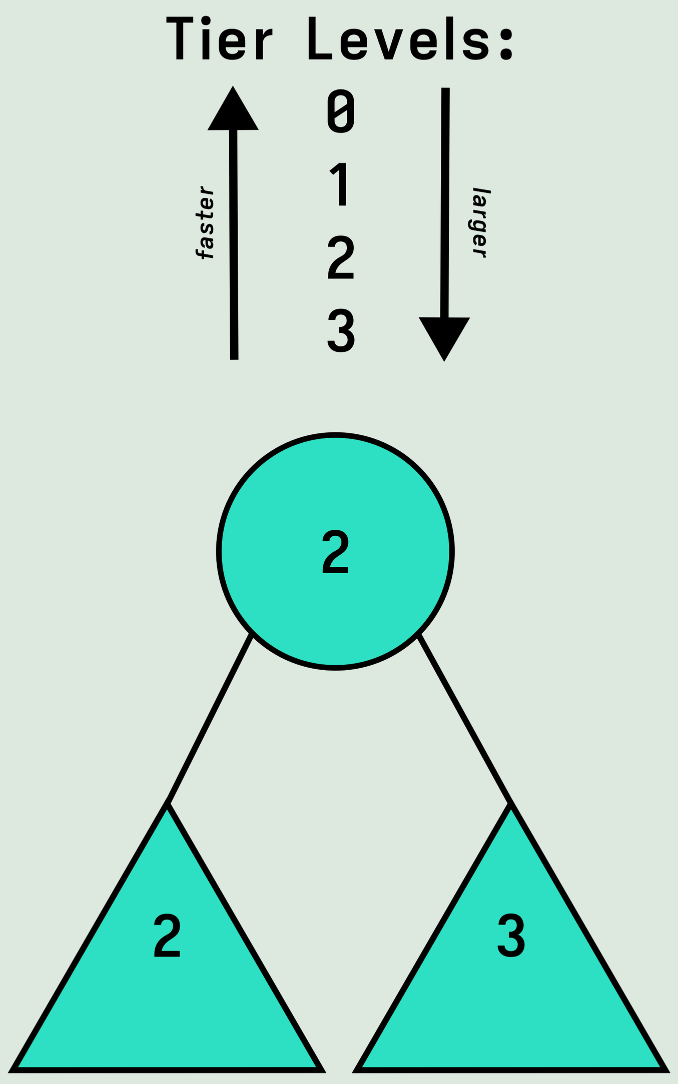
- Based on B$^\varepsilon$-trees
  - Modifications are buffered
  - ε determines size of buffer
  - Optimized for write accesses
- Access via key-value or object interface
  - Trees can be spread across different tiers
  - Different optimizations per tier

Work Packages

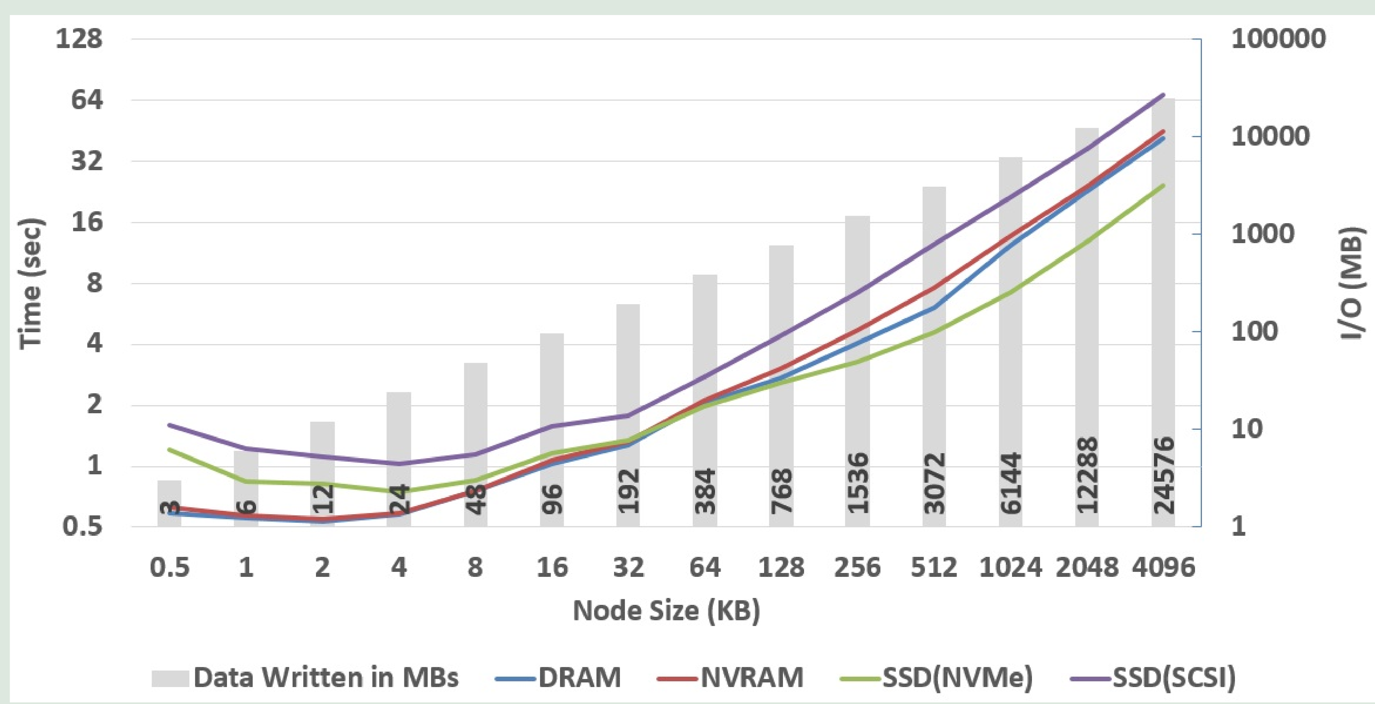
- **WP1: Common Storage Engine**
  - 1.1 Data structures and management
  - 1.2 Data placement and migration
  - 1.3 Application programming interface
- **WP2: Data Transformations**
  - 2.1 Data transformation algorithms
  - 2.2 External data transformation
  - 2.3 Hardware acceleration
- **WP3: Use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

Allocation and Migration Strategies on B$^\varepsilon$-Tree

- Initial placement currently by user on key/object level
  - Defaulted if not specified
  - Node at least as fast as fastest subtree
- Supports automatic migration
  - *Lazy* node migration ("zero-cost" on copy-on-write)
  - *Eager* key migration
- Plugin-like policy definition; for variety rich support
  - Traditional cache-inspired methods (LFU/MFU, ARC, etc.)
  - ML approach (Reinforcement Learning, STSNN, DNN, etc.)



NVRAM Experiments



Time taken by HAURA to write 6144 nodes

- Node sizes change the superiority of the devices
  - Larger node sizes favor SSD-NVMe
  - Smaller node sizes favor DRAM/NVRAM
  - → Heterogeneous node sizes among the tree as our next challenge

Contact

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

David Broneske
DZHW Hannover
broneske@dzhw.eu

Follow us:

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