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)

<table>
<thead>
<tr>
<th>Storage</th>
<th>Latency</th>
<th>Read Gran.</th>
<th>Write Gran.</th>
<th>Read Thr.</th>
<th>Write Thr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>≈ 100 ns</td>
<td>64 B</td>
<td>64 B</td>
<td>90 GB/s</td>
<td>50 GB/s</td>
</tr>
<tr>
<td>NVRAM</td>
<td>1,000 ns</td>
<td>64 B</td>
<td>30 GB/s</td>
<td>15 GB/s</td>
<td></td>
</tr>
<tr>
<td>NVMe SSD</td>
<td>≈ 10,000 ns</td>
<td>4 KiB</td>
<td>1-8 MB</td>
<td>3.5 GB/s</td>
<td>2.5 GB/s</td>
</tr>
<tr>
<td>SATA SSD</td>
<td>≈ 10,000 ns</td>
<td>4 KiB</td>
<td>1-8 MB</td>
<td>550 MB/s</td>
<td>500 MB/s</td>
</tr>
<tr>
<td>HDD</td>
<td>≈ 100,000,000 ns</td>
<td>4 KiB</td>
<td>250 MB/s</td>
<td>200 MB/s</td>
<td></td>
</tr>
<tr>
<td>SMR SSD</td>
<td>≈ 100,000,000 ns</td>
<td>4 KiB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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

SMASH Architecture
- Based on $\beta$-trees
  - Modifications are buffered
  - $\varepsilon$ 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

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

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 $\beta$-Tree
- Initial placement currently by user on key/object level
  - Defaulted if not specified
  - Node at least as fast as fastest sub-tree
- 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.)

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

NVRAM Experiments
- 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