Mnemosyne: Lightweight Persistent Memory

Haris Volos, Andres Jaan Tack, and Michael M. Swift

Motivation

Technology trends: Storage-Class Memory (SCM) blurs the distinction between
- Memory (fast, expensive, volatile), and
- Storage (slow, cheap, non-volatile)

Application trends: Storage-latency drives modern applications and services
- Web applications: Facebook, Amazon,
- Desktop applications: Firefox
- Distributed systems: Paxos
- Other: High-frequency trading

Goal: Enable flexible low-latency storage.

Challenges
- Expose storage-class memory to programmers for direct use.
- Support consistent modifications in the presence of failures.
- Design a system compatible with existing commodity processors.

Mnemosyne provides a new abstraction, persistent memory, that programmers can use to get direct access to durable storage.

Compared to past approaches to persistence [1,2,3], Mnemosyne enables fine-grain and low-latency updates to durable storage.

Storage-Class Memory

Features
- Byte-addressable
- Non-volatile
- Short access time (DRAM-like)

Technologies
- Phase Change Memory
- Spin Torque Transfer RAM
- Memristors

Performance Comparison

<table>
<thead>
<tr>
<th>Technology</th>
<th>Read</th>
<th>Write</th>
<th>Endurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>60 ns</td>
<td>60 ns</td>
<td>&gt;10^4</td>
</tr>
<tr>
<td>Spin Torque Transfer RAM</td>
<td>6 ns</td>
<td>13 ns</td>
<td>10^5</td>
</tr>
<tr>
<td>Phase Change Memory</td>
<td>100-300 ns</td>
<td>150 – 600 ns</td>
<td>10^7 – 10^12</td>
</tr>
<tr>
<td>NAND Flash</td>
<td>85 μs</td>
<td>200 – 500 μs</td>
<td>10^7 – 10^9</td>
</tr>
</tbody>
</table>

Programing Model

Persistent Memory Regions are segments of virtual memory stored in storage-class memory and backed by a disk file. Regions can be created statically to hold variables labeled with the keyword persistent, or allocated dynamically. The system also provides a persistent heap for allocating small blocks of memory.

Consistent Updates support modifying persistent data without risking correctness after a failure. The system provides four common mechanisms for consistently updating data with varying flexibility.

1) Single-variable
2) Append
3) Shadow
4) In-place

Legend:
- Existing data
- Replica
- New data
- Ordering constraint

Durable Memory Transactions support in-place updates. A programmer annotates a block of code with the code persistent and the compiler produces code that passes all memory references to a transaction system. The system ensures all modifications are atomic and durable but provides no concurrency control.

Example: Persistent Hashable

```c
persistent hash = NULL;
main () {
  if (!hash) {
    pmalloc(sizeof('bucket'), &hash);
  }

  update_hash(key, value) {
    lock(mutex);
    atomic pmalloc(sizeof('bucket'), &bucket);
    bucket->key = key;
    bucket->value = value;
    insert(hash, bucket);
    lock(mutex);
  }
}
```

**Performance Evaluation**

Platforms
- Mnemosyne: persistence primitives/regions + DRAM + fixed delay (150ns) after each store
- PCM-disk: ext2 file system + RAM-disk + a fixed delay when writing a block

Application Performance

<table>
<thead>
<tr>
<th>Application</th>
<th>Workload</th>
<th>Backend</th>
<th>Updates/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>TokyoCabinet</td>
<td>512-byte ins/del queries</td>
<td>PCM-disk, Memnosyne</td>
<td>8,361</td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>SLADM LDIF template</td>
<td>PCM-disk, Memnosyne</td>
<td>4,545</td>
</tr>
</tbody>
</table>

Hashtable Performance

Compare performance of a simple hash table implemented using Mnemosyne against Berkeley DB’s hash table.

**References**