Tair-PMem: A Fully Durable Non-Volatile Memory Database

Caixin Gong, Chengjin Tian, Zhengheng Wang, Sheng Wang, Xiyu Wang, Qiulei Fu, Wu Qin, Long Qian, Rui Chen, Jiang Qi, Ruo Wang, Guoyun Zhu, Chenghu Yang, Wei Zhang, Feifei Li

Alibaba Group
Redis Advantages & Disadvantages

The Most Popular Memory Database

Redis

Abundant models

High Performance

Goals

More Valuable Information

Larger Cluster

Difficult

To

Achieve

Expensive

Volatile
Opportunities and Challenges

Enterprise Features
- Abundant models
- High Performance
- Full durability
- more economical

Challenge 1, performance degradation
- Longer access Latency (3x)
- Much lower Bandwidth (10x)

Challenge 2, Redis compatibility

Challenge 3, NVM programming complexity

Intel Optane PM

Latency:
- DRAM: 0.1us
- PM (Persistent Memory): 100 us
- SSD: 100 us
- HDD: 100 us

Cost:
- DRAM: $16/GB
- PM (Persistent Memory): $1/GB
- SSD: $1/GB
- HDD: $1/GB
Outline

• Core Design Decisions

• The Database Architecture

• Evaluations
Decision 1: Hybrid Memory

- **For performance**
  - Keep index (small in size) in DRAM.
  - A small part of index may be stored in NVM.
  - Most KV read takes only one NVM access.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>size</th>
<th>persistent</th>
<th>hot</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>User data</td>
<td>large</td>
<td>yes</td>
<td>-</td>
<td>NVM</td>
</tr>
<tr>
<td>MetaData of Allocator</td>
<td>small</td>
<td>no</td>
<td>yes</td>
<td>DRAM</td>
</tr>
<tr>
<td>Indexes</td>
<td>large</td>
<td>no</td>
<td>-</td>
<td>DRAM/NVM</td>
</tr>
<tr>
<td>Runtime variables</td>
<td>small</td>
<td>no</td>
<td>-</td>
<td>DRAM/NVM</td>
</tr>
</tbody>
</table>

The characteristics of different data.

The hybrid memory structure.
Decision 2: Log as Data

• What data should be persistent for durability, and How to organize them?
  ⭐ For Performance: Log plays the role of user data, which makes user data only written once.

• How to recover
  ➢ Redo the log to reconstruct indexes.
Decision 3: No Changes to Read Operation

• For easy programming
  ➢ User KVs encoded in *Log & data pool* keep the original format.
  ⭐Index need not be re-implemented, so as read operations.
Decision 4: Programming Toolkit

- A toolkit to hide the complexity of NVM programming
  - An allocator to manage both DRAM and NVM;
  - A component (the *Log & Data Pool*) to store all the persistent data;
  - high performance.

⭐ Easy Programming

The structure of toolkit
Outline

• Core Design Decisions

• The Database Architecture

• Evaluations
Architecture

- **Toolkit**
  - Allocator; Log & Data pool
- **Database Core**
  - Support abundant models for compatibility
  - Database components
Toolkit

• Allocator
  - Manages both DRAM and NVM, and produces `malloc/free` style APIs.
  - Metadata is volatile
  - An allocation can be recovered.

• Log & Data Pool
  - Stores all persistent data, which is organized by an atomic persistent list.
  - Supports persistent and atomic append and delete.
  - Supports recovery.
Database – Data Encode

• Abundant model and indices

• The Encode Method
  ➢ Abstracted to KV/KKVs.
  ➢ The key/value can be pointed by index as the original Redis.
  ➢ The implementation of read operations remains intact.
Database – User Write Operations

• Write operations generate an entry to serve as a redo log.
  ➢ Both Insert and Update operations create a user data entry.
  ➢ Deletion generates a tombstone entry.
  ➢ Take update as an example

• Disaster Recovery
  ➢ Sequentially redoes the log to reconstruct indices.
GC and Checkpoint

• Entry deletion is done by the background GC thread.
  • The deletion order should be right.

• When taking a snapshot/checkpoint
  ➢ The GC thread protects the entries to be deleted.
  ➢ Other procedures of checkpoint are the same as the original Redis'.
Programming Skills

• Breaking Large Values into Shards for COW

• Single Tombstone Entry When Possible

• Prefetching

• Pin frequently accessed index in DRAM
Outline

• Core Design Decisions

• The Database Architecture

• Evaluations
Throughputs

Tair-PMem is better, compared to fully durable (FD) Redis,
Tair-PMem is comparable, compared to partially durable (PD) Redis,
Tair-PMem is always better, compared to TieredMemDB.
99 Percentile Latencies

The 99 percentile latencies of string model.

The 99 percentile latencies of hash model.

Much better 99 percentile latency due to no AOF writing.
Maximum Latencies

The maximum latencies of string model.

The maximum latencies of hash model.

Much more stable because of no AOF rewriting which incurs fork system call.
Conclusions

Abundant models

Volatile

Redis

High Performance

Unstable latency

Expensive

Enterprise Features

Full durability

more economical

Latency stability

Tair-PMem

Abundant models

High Performance
For More Information

Tair-Pmem service on Alibaba Cloud