Software Persistent Memory

FLORIDA INTERNATIONAL UNIVERSITY

Jorge Guerra, Leonardo Mármol, Daniel Campello, Carlos Crespo, Raju Rangaswami, Jinpeng Wei

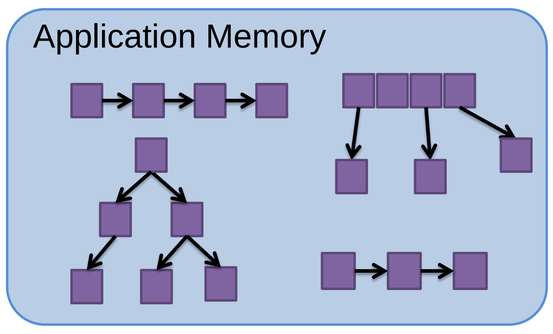
Persistence is ubiquitous!

Metadata Persistence in Systems Software	File Systems: layout management, journaling, versioning, snapshoting
Application Data Persistence	End-user applications, databases, key-value stores
Computational State Persistence	Scientific (HPC) applications, “Big Data” applications

What is needed for memory persistence?

There are multiple mechanisms of achieving persistence:

- Serialize and write to file
- Memory mapped I/O
- Databases



To achieve persistence using files:

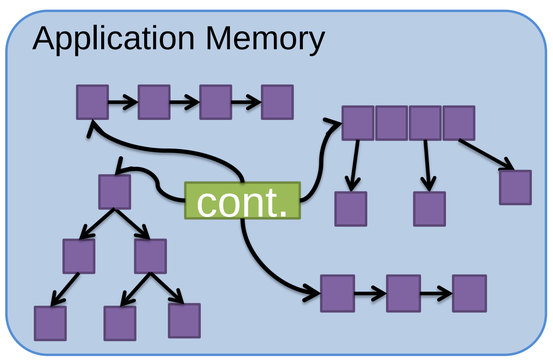
- Developer tracks data to be persisted
  - ✘ *Ensure all required data is tracked*
- Data is serialized and written to persistent media
  - ✘ *Can be complicated for arbitrary data structures*
- When restoring data structures must be recreated
  - ✘ *May require addition bookkeeping*

A different programming model

We propose Software Persistent Memory (*SoftPM*) a system which:

- ✓ Automatically tracks data dependencies
- ✓ Transfer data from memory to the persistent media using a transactional model
- ✓ Loads data back into memory

What’s the difference in programming models?



Using SoftPM is simple and requires only minimal changes to the code (only 9 new LOC for SQLite which is more than 70K LOC).

To achieve persistence using SoftPM:

- Developer creates a containers, which reference data to be persisted
  - ✓ *All memory reachable from the container is persistented*
- Memory pages are written to persistent media
  - ✓ *No data transformation required*
  - ✓ *Write I/O is optimized for the underlying media*
- Restoring data is a simple atomic operation

How does SoftPM work

We rely on a combination of static code C analysis and runtime instrumentation.

At compile time we add hints to:

- Record the location and size of memory allocations
- Register the location of all pointers **at initialization**
  - ✓ Explicit initialization: *l-value* of assignment operations
  - ✓ Implicit initialization: *memory copying/move* operations

Type Definition	Usage

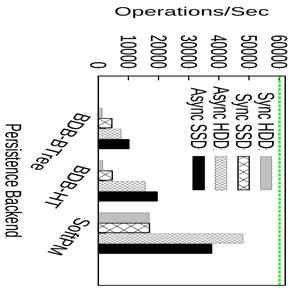
struct list {
  int val;
  struct list *next;
};
	struct list *l = malloc(n);
	mallocat(l, n);
	pointerat(&l);

Evaluation

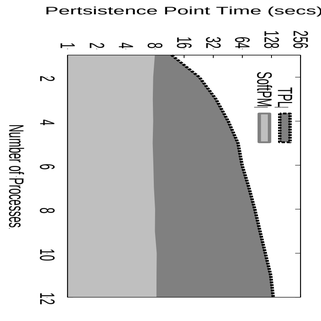
Performance

The workload adds, queries, and deletes 512 byte elements with 16 byte keys. The dashed line represents a memory only solution.

Performance of memcachedb



- ✓ Only minor code changes required
- ✓ With SoftPM we achieve up to 10X better performance



Scalability

- ✓ Guaranteed sequential writes even with multiple threads
- ✓ I/O time scales linearly

Data Structure /Application	Original LOC	LOC for Persistence	LOC to use SoftPM
Array	102	17	3
Linked List	188	24	3
RB Tree	285	22	3
Hash Table	396	21	3
SQLite	73042	6696	9
memcachedb	1894	205	40

Code Complexity

- ✓ Using SoftPM reduces the LOC required for persistence
- ✓ No need for application level I/O barriers

Conclusions and Future Directions

SoftPM provides:

- ✓ A memory abstraction to work with persistent memory
- ✓ High-Performance persistence I/O with existing technologies
- ✓ Conventional volatile interfaces for easy use and adoption

Future work:

- ✓ Support for versioning, record-replay, and execution branching
- ✓ Support for sharing containers across multiple processes
- ✓ Support for other programming languages (C++, Fortran)