Twizzler: A Data-Centric OS for Persistent Memory

Daniel Bittman

Peter Alvaro  Pankaj Mehra  Darrell Long  Ethan Miller

Center for Research in Storage Systems
University of California, Santa Cruz
Hardware Trends

sys_read

~100-300 ns  ~1 us  ~1-10 ms
Growing, becoming persistent  Outdated interface  Cannot compute on directly

Persistent data should be operated on directly and like memory
Persistent data should be operated on *directly* and *like memory*
Existing approaches?

**POSIX**
- Explicit persistence and data access
- Multiple forms of data
- Kernel involvement
- mmap helps, but does not solve the virtual memory problem

**PMDK**
- No OS support
- Data sharing is hard
- Slow pointers

**Twizzler’s goals**
- Little kernel involvement
- Pervasive support (security, sharing)
- Low overhead persistent pointers
Design Overview

- application
- musl* libc
- twix
- twix
- Twizzler kernel

* modified musl to change linux syscalls into function calls

Data object:
- direct access (memory-style)
- POSIX access (read/write)
- metadata & FOT management
- create, delete, etc.
- physical mapping

libtwz
- Linux syscall emulation
- view management, pointer translation, consistency primitives
- userspace
- kernelspace

Object & thread management, trusted computing base
Persistent Pointers

Pointers may be *cross-object*: referring to data within a different object

- **Object ID or Name**
- **Name Resolver**
- **Flags**

FOT entry of >0 means “cross-object”—points to a different object.
Implications for Data and Sharing

Objects are self-contained

Persistent pointers are based on *identity* not *location*

Persistent pointers can be operated on generically

**Objects can be easily shared**

Pointers in Twizzler

Pointers in PMDK

64-bit IDs require global coordination or collision management

Coordination free sharing
Consistency and Security

Cryptographically signed capabilities for access control

The kernel cannot create capabilities, but it can (must) verify them.

All enforcement must be done by hardware.
Implementation

More details available at twizzler.io
Evaluation Goals

Programmability, not performance (though, performance where we can get it)
Case Study: KVS

Lookup returns direct pointers

250 lines of simple C code is *all you need*
Evaluation

Dell R640 Servers with Intel Optane DC

Ported SQLite to Twizzler and to PMDK

Compared to SQLite “native” and SQLite “LMDB” (mmap)
Performance: SQLite

![Graph showing transaction rate normalized for different workloads](image-url)
Performance: SQLite

![Chart showing performance of SQLite operations]

- **Sort**
- **Mean**
- **Median**
- **Index**
- **Find**
- **Probe**

- **SQL-Native**
- **SQL-PMDK**
- **SQL-LMDB**
- **SQL-Twz**

Time (normalized)
Future Work: Distributed Twizzler

It’s a rendezvous problem

Explicit Relationships and the Object Graph
Conclusion

Operating systems must evolve to support persistent data programming models directly.

Cross-object pointers allow us to realize the power of UNIX in a data-centric model.

Twizzler provides benefits for both NVM and traditional systems.
Thank You! Questions?