Re-architecting the Systems Stack for Byte-Addressable NVRAM

Ethan L. Miller
Center for Research in Storage Systems
Integrating byte-addressable NVM into the systems stack

- Byte-addressable NVM technologies will become commonplace
  - Memory-style DIMMs
  - Separate “replaceable” modules

- How is the operating system going to best leverage this new memory technology?
  - Treat it like a block device?
  - Treat it like DRAM that persists across power failure?

- This is a longer-term project!
  - Requires exploring and modifying OS structure
  - May require architecture changes
    - Possible without them, but not as efficient
Impact of new NVM technologies

- New technologies will (initially) be more expensive
  - Combine byte-addressable NVM (BNVM) with flash?
  - Use caching / write buffer strategies to leverage smaller amounts of BNVM?

- Likely approach: build devices that leverage multiple technologies for
  - Performance and flexibility of byte-addressable NVM
  - Cost advantages of flash?

- Big question: how will the storage / systems stack use such devices?

- This research has potential for high impact!
Our vision: single-instance store

- Eliminate the transformation from memory to storage
- Use a single address space across many systems
- Base the store on objects
  - Flexible naming
  - Object sharing
  - Mobility

Many issues to address

- Basic system design
- Consistency / invariant preservation: what happens when a system fails mid-update?
- How can memory associate multiple related updates?
- How do programs find the data (objects) they need: naming?
Object-based NVM everywhere

- Data remains an object everywhere
  - Pointers are <object, offset>
  - Similar to MULTICS segments
- Objects named by GUIDs
  - OS translates "human-readable" name into GUID
- Object may be fetched from any store
Memory for a single object

- **Object is paged**
  - Similar to MULTICS

- **Individual object identified by GUID**
  - Unique ID: never reused (ever!)
  - Large ID space (128–256 bits)

- **Keep pointers small**
  - `<local_id, offset>`
  - Translate `local_id` to GUID using per-object table

- **Non-local GUID can be resolved at**
  - Creation time: always refers to the same object
  - Access time: might involve translation through a naming service
    - Replace GUID in table with a name

![Diagram](image)
Managing pointers

Translation poses design issues
• Pointer must be translated if moved to a new object
• System needs to know which object a pointer came from

Pointer translation
• Add destination GUID to the table (if needed)
• Translate local_id from first object’s view to second object’s view
• This is fast, and can be done automatically with system help
System-level issues

❖ Consistency & atomicity
  • Need to combine multiple writes to an object into atomic operations
  • Leverage new Intel cache line flush instructions
  • Use copy-on-write?
  • Specialized data structures?
  • What happens to multi-object updates?
  • How is consistency handled if an object exists in multiple stores?

❖ Garbage collection of objects is a potential issue
  • Every new process generates one or more objects
  • Many objects are evanescent: need efficient ways to specify persistence level
Object naming

❖ Objects are “named” by GUIDs
  • Not very user friendly!
  • Need an approach to map human-readable names to GUIDs

❖ Solution: metadata (naming) service
  • Converts strings to GUIDs

❖ Might be multiple options for naming services!
  • Hierarchical POSIX-like naming system
  • Database-like naming system
  • URLs / URIs?

❖ Naming services can also help locate objects
  • Objects might be stored on local BNVM
  • Objects might be further away: naming service can identify locations
Other issues

❖ Security and access control
  • Need to be able to restrict access to objects
  • Internal mechanism should be straightforward
  • What does the user interface look like?

❖ User API for our approach
  • Greatly simplified: may resemble mmap-type calls
  • Need a POSIX shim for backwards compatibility
    • Fortunately, this isn’t too difficult!
    • read(), write(), etc. can be done by operations on objects
    • POSIX doesn’t allow for cross-object pointers…
We have a home-grown operating system!
• Written largely by Daniel Bittman, current undergrad (and soon-to-be grad student)
• Modify the operating system to support simulated BNVM
  • Unless one or more sponsors can get us real BNVM…
• Use existing x86 and/or ARM CPUs to simulate true segmentation
  • Propose new (reused old) primitives to make this easier

Develop object-based storage device using BNVM and flash
• Work with Jishen Zhao on this

Develop interfaces between
• System stack and object-based BNVM-based storage
• System stack / OS and user

Multiple researchers working over several years!
• Several dissertations' worth of new ideas
• High potential for very high impact!
Questions & discussion