Isotope: Transactional Isolation for Block Storage

Ji-Yong Shin
Cornell University

In collaboration with
Mahesh Balakrishnan (Yale), Tudor Marian (Google), and
Hakim Weatherspoon (Cornell)
Multicore and Concurrency

• Concurrent access to storage is the norm

• For safe data access, concurrency control is a must
Concurrency Control in Storage Stacks

• Most modern apps support concurrency control
  – App-specific implementation
  – Typically, locking

Concurrency Control (+ Atomicity/Durability) Is Difficult

Transactional Block Store (Isolation + Atomicity + Durability)
Why Transactional Block Store?

• Simpler applications
  – One common implementation for isolation (and atomicity/durability)
  – TX APIs decouple policy/mechanism
  – TX over application-level constructs (e.g. file, directories, key-value pairs)
  – TX across different applications (e.g. read from file and write to KV store)
End-To-End Argument?

Application specific functions should be in end-hosts
– Transactional isolation is general
Pushed down function should not incur unnecessary overheads
– Isolation can be implemented efficiently

Many block-level functions, e.g. atomicity, block layer indirection, are already implemented

TX using optimistic concurrency control yields low overhead
How do we design a transactional block store?

Isotope

Is a transactional block store useful?

IsoBT, IsoHT, IsoFS, and ImgStore
Rest of the Talk

• Isotope
  – Overview
  – Design and APIs
  – Applications

• Performance Evaluation

• Conclusion
Isotope

• The first block store to support TX isolation
  – MARS and TxFlash only supported TX atomicity

• Multi-version optimistic concurrency control
  – Keeps multiple versions of block data
  – Speculatively executes TX until commit time

• One of two semantics supported
  – Strict serializability
  – Snapshot isolation

• Simple APIs
  – BeginTX/EndTX/AbortTX and more
```plaintext
BeginTX();
foo=Read(0);
Write(1,boo);
Write(3,baz);
EndTX();
```
Deciding Transactions

- Strict serializability based
  - Checks for read/write conflicts

```
BeginTX(); // @ T53
foo=Read(33);
Write(25, bar);
Write(33, baz);
EndTX();   // @ T55
```
Isotope Challenges and Additional APIs

1. Application must be stateless (no caches)
   - `PleaseCache()`: caches a data block in internal memory cache

2. Mismatching data access granularity (application vs block)
   - `MarkDirty()`: indicates subblock level data access

---

**False Conflict**

TX A
Write (0, foo); // modified 1st bit

TX B
Write (0, bar); // modified last bit

Filesystem metadata block
Implementation

• Built as device mapper in Linux kernel
  – Logical block device similar to software RAID or LVM
  – Can run on any block device (Disk, SSD, etc.)

• Log implemented based on Gecko
  – Chain logging design
    (Logs to multiple drives in round robin)

• APIs supported using IOCTL calls
  – BeginTX/EndTX/AbortTX
  – MarkDirty/PleaseCache
  – ReleaseTX/TakeoverTX
Isotope Applications

• IsoBT and IsoHT
  – C++ library key-value stores
  – Based on persistent B-tree and hashtable
  – ACID Put, Get, Delete, etc.

• IsoFS
  – FUSE based transactional filesystem
  – Executes arbitrary filesystem ops (read, write, rename, etc.) ACID’ly
  – PleaseCache to handle metadata
Ease of Programming

- Lines of code

<table>
<thead>
<tr>
<th>Application</th>
<th>Naïve Lock-Based Isolation</th>
<th>Isotope TX APIs (lines modified)</th>
<th>Isotope Optional APIs (lines added)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsoHT</td>
<td>591</td>
<td>591 (15)</td>
<td>617 (26)</td>
</tr>
<tr>
<td>IsoBT</td>
<td>1,229</td>
<td>1,229 (12)</td>
<td>1,246 (17)</td>
</tr>
<tr>
<td>IsoFS</td>
<td>997</td>
<td>997 (19)</td>
<td>1,022 (25)</td>
</tr>
</tbody>
</table>

- Simple replacement of locks to BeginTX/EndTX/AbortTX
- Only few lines of code to add optimizations

Very easy to build transactional applications using Isotope APIs
Composing Applications

• ImgStore
  – Transactional storage with two subsystems
  – IsoBT for metadata and IsoHT for images

• Case
  1. Library
Composing Applications

- **ImgStore**
  - Transactional storage with two subsystems
  - IsoBT for metadata and IsoHT for images

- **Case**
  1. Library
  2. Process

Returns a transaction handle

Continues on a transaction given the handle

Thread Id: X

TX Handles through IPC

2 processes with threads
Composing Applications

• ImgStore
  – Transactional storage with two subsystems
  – IsoBT for metadata and IsoHT for images

• Case
  1. Library
  2. Process
  3. Thread pools

  1. ImgStore was only 150 LoC
  2. Easy to build large apps whose TX cross boundaries
Performance Evaluation

1. Micro benchmark
   – Base performance of Isotope?

2. Key-value stores
   – Performance of applications built over Isotope?

3. Filesystems
   – Performance of new and existing filesystems?

4. ImgStore Composition
   – Performance under different composition?
Micro Benchmark
(Base Performance of Isotope)

- Random 3-4KB-reads-3-4KB-writes TX’es from 64 threads
- Increasing address space (decreasing Tx conflicts)
- Ran on 3-SSD chain

1. Aborts are cheap
2. Subblock TX mechanism has negligible overhead
Key-Value Stores

• LevelDB: on RAID0 volume, Sync/Async mode
• Increasing number of threads on 2 SSDs
• 8KB data using YCSB workload-a

Isotope-based applications perform comparable to existing applications and guarantee strong semantics
Filesystems

- Ext2 and Ext3 on top of Isotope on SSDs
  - Logging benefit
  - All I/Os as singleton transactions

- IOZone benchmark write/rewrite phase with 8 threads

1. IsoFS performs comparable to ext2/3
2. ext2/3 saturates SSD with no slowdown
**ImgStore Compositions**

- Different compositions of ImgStore
- YCSB Workload-a
  - 16KB image to/from IsoHT and metadata to/from IsoBT in a TX

1. Small ReleaseTX/TakeoverTX overhead (lib vs thread)
2. Cross process overhead comes from IPC
Conclusion

• First block storage with TX isolation
  – Simple API: BeginTX, EndTX, AbortTX
  – Low overhead design
    (nearly free abort and MVCC)
  – Optimizations for fine grained TX and caching

• Facilitates TX application design
  – 1K LoC transactional KV-stores and filesystem
  – Easy support for composition of TX applications

• Right time to consider pushing Isolation down the I/O stack
Thank you

Questions?