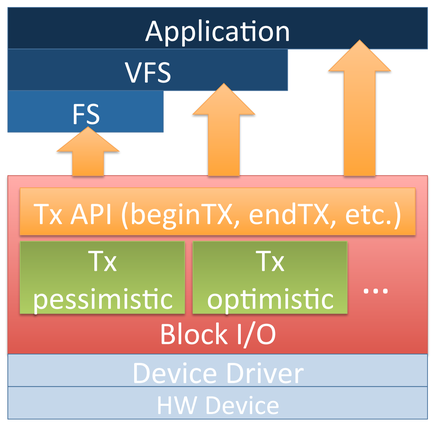
Isotope: Transactional Isolation for Block Storage

Ji-Yong Shin[1], Mahesh Balakrishnan[2], Tudor Marian[3], and Hakim Weatherspoon[1]
[1]Cornell University , [2]Yale University, [3]Google

Isotope: A Case for Block I/O Level Tx Isolation

- Traditionally I/O stacks are top heavy
  - ✓ Filesystem and above have rich functionalities
  - ✓ Block I/O and below are simple/non-intelligent
  - ✓ Time for new abstractions!
- Tx isolation is general
  - ✓ Multicore and cloud make concurrency a norm
- Tx isolation is difficult
  - ✓ Having one high quality implementation helps



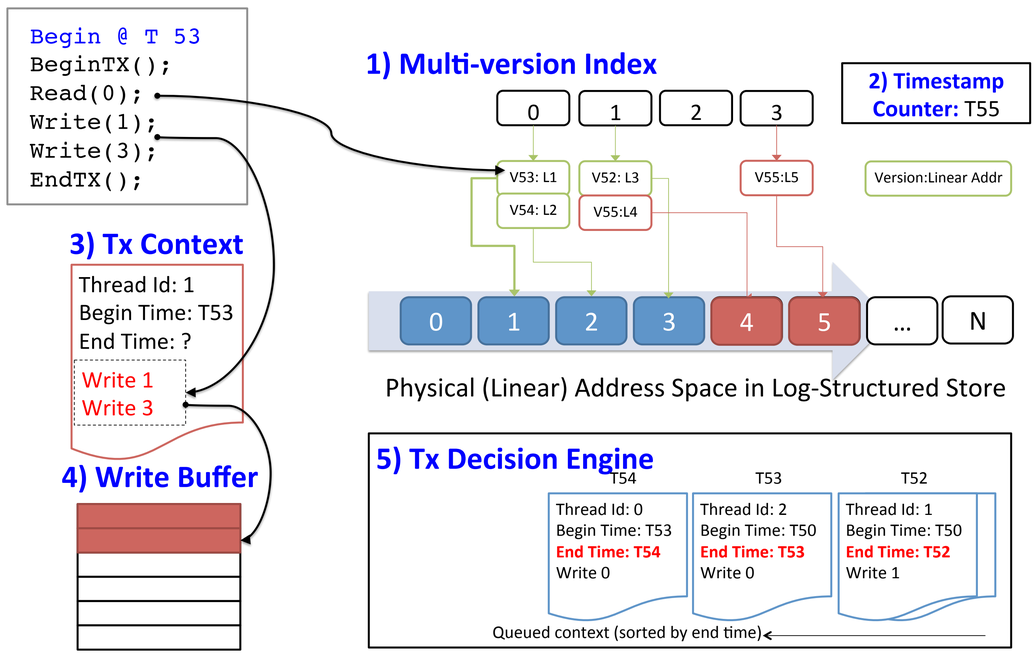
- Tx isolation in block layer enables easy and clean software design
  - ✓ Lowest common software layer that can directly support higher layers
  - ✓ New abstractions enable policy/mechanism separation
- Tx isolation can be implemented with negligible overhead
  - ✓ Optimistic concurrency control using advanced CPUs and abundant Memory

Isotope APIs

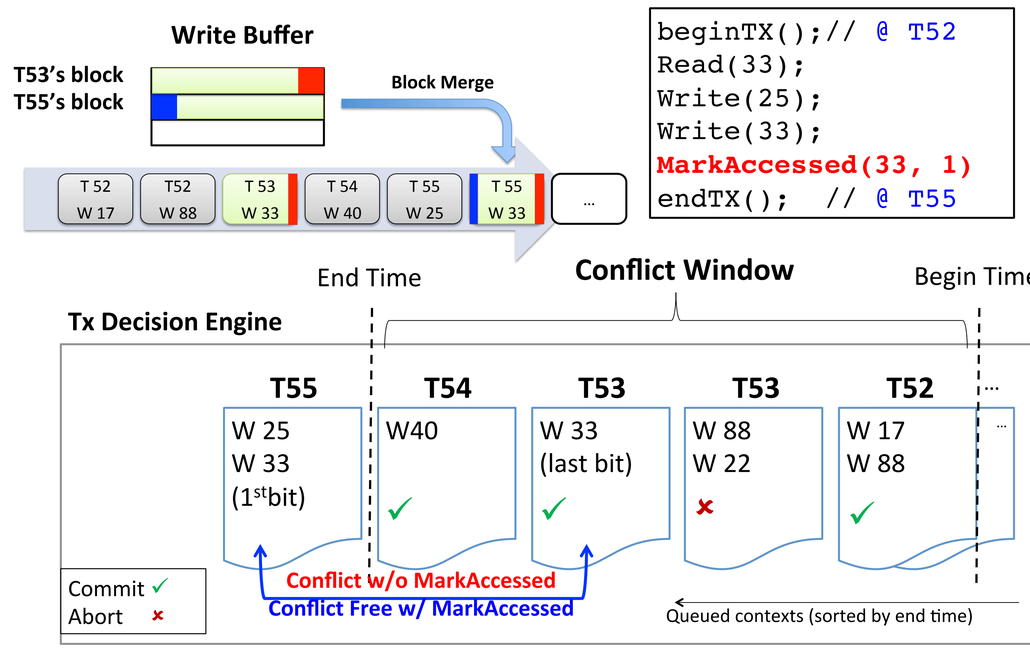
- BeginTx()
  - ✓ Creates a Tx context
  - ✓ Treats I/Os before EndTx as a Tx
  - ✓ Every write handled in memory
- AbortTx()
  - ✓ Terminates a Tx
- MarkAccessed
  - ✓ Marks subblock accesses
- PleaseCache
  - ✓ Enables in-memory read
  - ✓ E.g. for filesystem metadata
- EndTx()
  - ✓ Checks transaction conflicts
  - ✓ Persists updates on success
  - ✓ Aborts on failure
  - ✓ Returns success/failure

```
// Example Code
txbegin:
BeginTX();
If (!read(blknum1, buf1)) {
            AbortTX();
            return EIO;
}
If (!write(blknum2, buf2)) {
            AbortTX();
            return EIO;
}
If (!EndTX()) goto txbegin;
```

Isotope Design



Deciding Tx'es and Subblock Writes



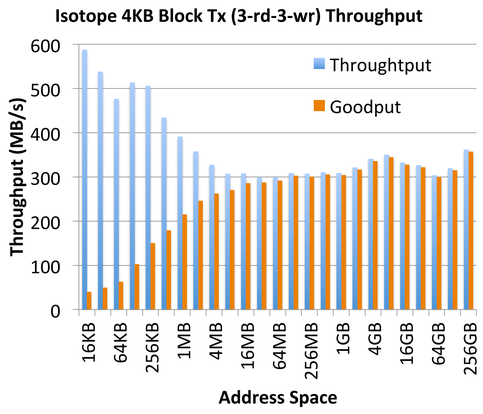
Implementation

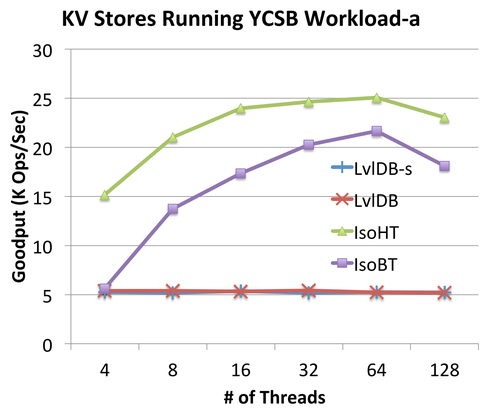
- Block I/O layer kernel module (device mapper)
  - ✓ Similar to LVM and software RAID
  - ✓ Can run on any block devices (Disk, SSD, etc.)
- Based on Gecko, a chain logging storage
  - ✓ Log-structured design with chaining block devices
  - ✓ Garbage collection is isolated from writes
  - ✓ SSD and memory cache included

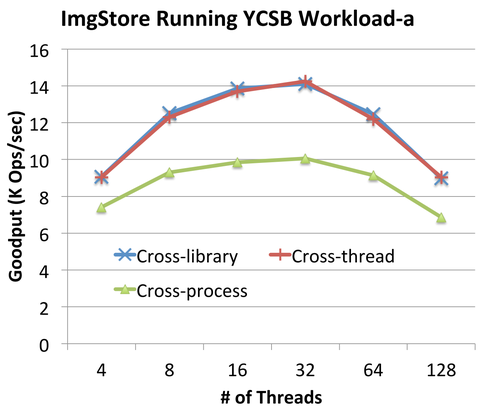
Applications

- IsoBT and IsoHT
  - ✓ Persistent B-tree and hashtable based key-value stores
  - ✓ Uses LevelDB APIs
- IsoFS
  - ✓ Transactional file system on FUSE
  - ✓ PleaseCache() for metadata
- ImgStore using IsoBT and IsoHT
  - ✓ IsoBT for metadata and IsoHT for data
  - ✓ ReleaseTX/TakeoverTX to continue transactions
  - ✓ Three compositions to handle transactions across, libraries, threads, and processes
- Easy to build transactional applications with Isotope APIs
  - ✓ 1K LoC for IsoFS and 150 LoC for ImgStore

Performance







- Low overhead aborts due to in-memory operations
- Saturates SSD's read/write throughput
- Applications on Isotope perform comparable to existing applications (LevelDB)
- RelaseTX/TakeoverTX overhead is negligible
- IPC overhead exists for cross-process Tx'es

Conclusion

- First system with block level Tx isolation
  - ✓ Other systems so far only supported Tx atomicity
- Simple APIs with optimistic concurrency control
  - ✓ BeginTx/EndTx/AbortTx
  - ✓ Multiversion concurrency control based
- Facilitates Tx application design
  - ✓ IsoBT, IsoHT, IsoFS and ImgStore
- Performs with low overhead
  - ✓ saturates block device bandwidth

*Work funded by NSF and DARPA