Towards Weakly Consistent Local Storage Systems

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

Server Trends

Year	2006	2016	Comparisons
Model (4U)	Dell PowerEdge 6850	Dell PowerEdge R930	
CPU [# of cores]	4 × 2 core Xeon [8]	4 × 24 core Xeon [96]	**12X**
Memory	64GB	6TB	**96X**
Network bandwidth	2 × 1GigE	2 × 1GigE 2 × 10GigE	**11X**
Storage	8 × SCSI/SAS HDD	24 × SAS HDD/SSD 10 x PCIe SSD	**# of devices: 4.2X** **Capacity: 175.3X** **Use of SSDs**

- Modern servers are as powerful as distributed systems in the past
  - ✓ CPU and storage devices are parallel, similar to distributed nodes
- Goal is to trade-off consistency and performance in a local store
  - ✓ Use of stale data in different storage devices for better performance

Distributed vs Modern Server

Distributed Systems	Modern Servers
Different versions of data exist in different servers due to network delays during replication	Different versions of data exist in different storage media due to logging, caching, copy-on-write, deduplication, etc.
Older versions are faster to access when the network overhead is low	Older versions are faster to access when they are on faster storage media

Reasons for different access speeds
- ✓RAM, SSD, HDD, hybrid-drives, etc.
- ✓Disk with arm contention or SSD under garbage collection
- ✓RAID under degraded mode

StaleStore

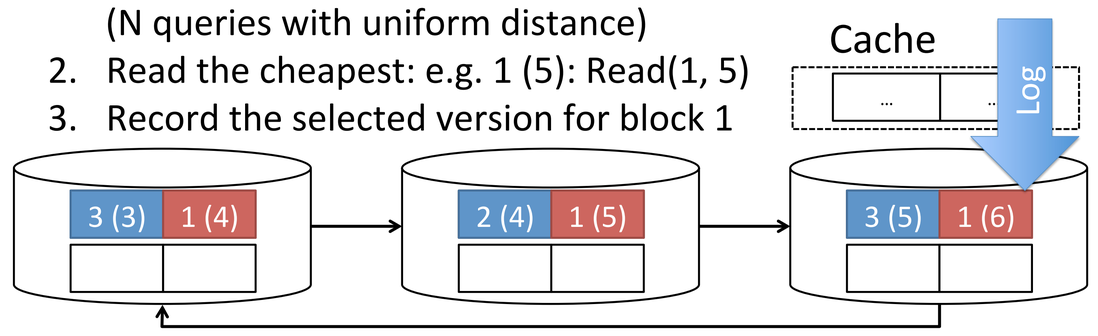
- Local storage systems in any form that can trade-off consistency and performance
  (e.g. KV-store, filesystem, block store, DB, etc.)

Requirements:
1. Maintain multiple versions of data
   - Should have interface to access older versions
2. Aware of consistency semantics
   - Bounded Staleness, monotonic-reads, read-my-writes, etc.
3. Can give cost estimates for accessing each version
   - Considerations for data locations and storage conditions

Yogurt: A Block Level StaleStore

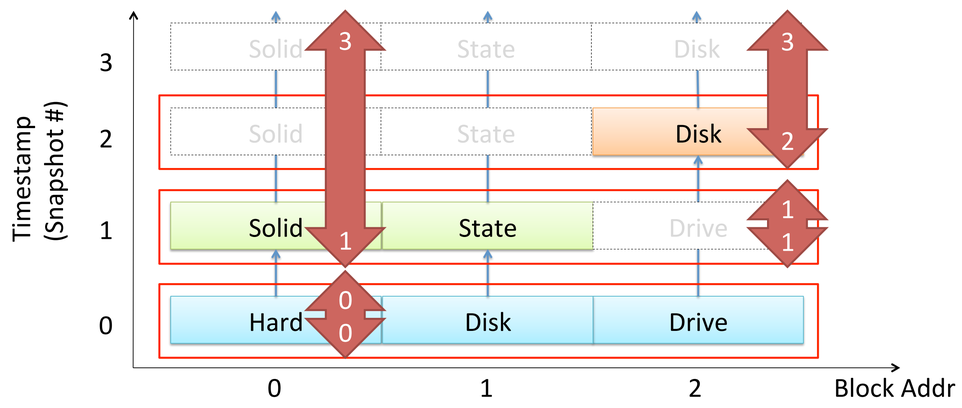
I/O	Write(blk, data, ver), Read(blk, ver)	Versioned writes to snapshots Versioned reads from snapshots
Cost	GetCost(blk, ver)	cache << disk, # of queued I/O (read << write)
Multi-block object access	GetVersonRange(blk, ver)	Returns a version range which a block is valid

Reading block 1 (monotonic-reads)
1. Issue GetCost() for block 1 between versions 3 and 6 (N queries with uniform distance)
2. Read the cheapest: e.g. 1 (5): Read(1, 5)
3. Record the selected version for block 1

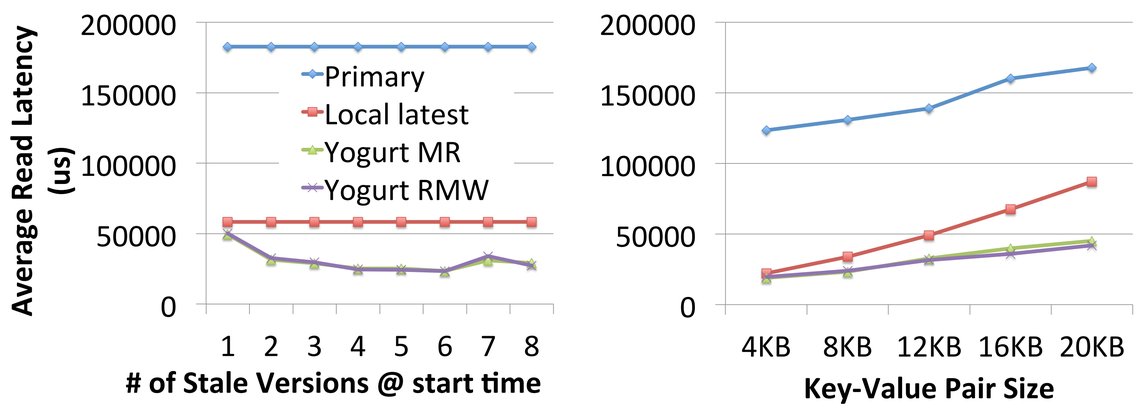


Multi-Block Object Access in Yogurt

- Key-value stores, filesystems can store an object over multiple blocks
- Read should be served from a persistent snapshot: GetVersionRange()

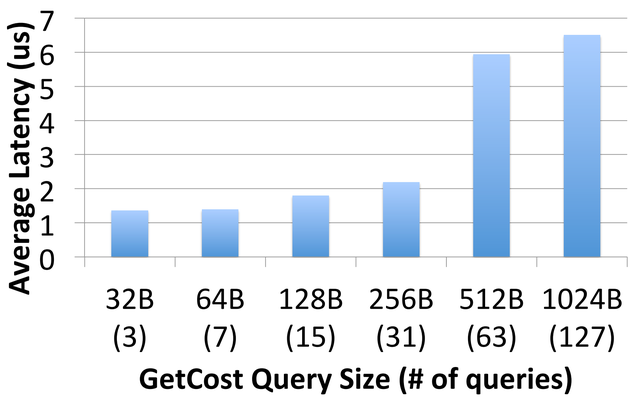


Performance: Accessing Blocks and K-V Pairs



- Primary/Backup setting
- Primary performs the worst due to network delays (100ms)
- Yogurt performs better than local latest by using the trade-off

GetCost Overhead



- Cost querying overhead is negligible compared to disk and SSD access latencies

Other Possible StaleStores

- Single disk log-structured store
- SSD flash translation layers
- Log-structured arrays
- Durable write caches that are fast for writes but slow for reads
- Deduplicated systems with read caches
- Fine-grained logging over a block-grained cache
- Systems storing differences from previous versions

Summary

- Modern servers are similar to distributed systems
- Local storage systems can adopt weak consistency
  - ✓ We define them as **StaleStores**
- Yogurt, a block level StaleStore
  - ✓ Effectively trades-off consistency and performance
  - ✓ Supports high level multi-block data constructs