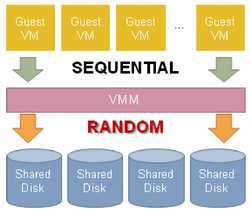
Gecko: A Contention-Oblivious Design for Cloud Storage

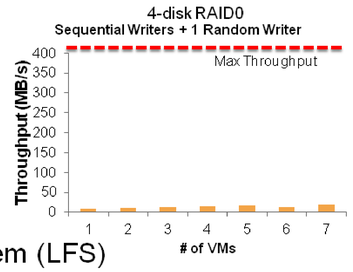
Ji-Yong Shin[1], Mahesh Balakrishnan[2], Tudor Marian[3], Lakshmi Ganesh[4], and Hakim Weatherspoon[1]

[1]Cornell University, [2]Microsoft Research, [3]Google, [4]University of Texas at Austin

Motivation

- Cloud/Virtualization accelerates consolidation of servers
  - Numbers of CPU cores and VMs increase per server
  - Storage is typically poorly virtualized

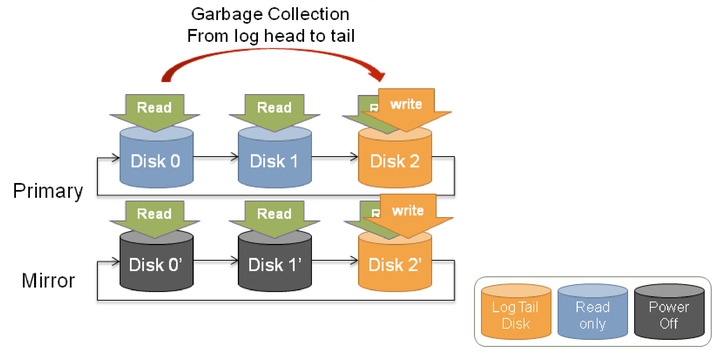




- Log-structured file system (LFS)
  - Solves write-write contention
  - Garbage collection is the Achilles' Heel
  - First class reads can interfere with writes

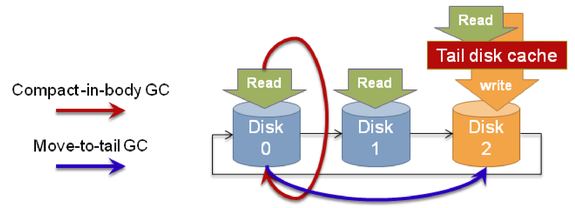
Overview of Gecko

- Chain logging design
  - Solves write-write contention by logging
  - Minimizes GC-write contention by separating log tail and head
  - High performance and power saving via mirroring or striping
  - Avoids read-write contention using cache



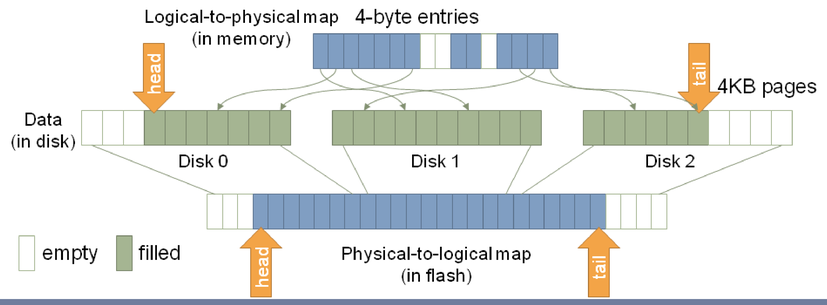
Caching and Garbage Collection (GC)

- Flash-based tail cache for read-write contention
  - Based on cheap MLC flash
  - Caches log-tail-drive only (30+GB flash for 500GB disk)
  - Blocks 86% of first class reads of real workloads
  - Mitigates read-write contention (Similar to LFS assumptions)
- Two types of GC
  - Move-to-tail GC (+ Simple metadata, - Shares write bandwidth)
  - Compact-in-body GC (+ No tail interference, - Complicated metadata)



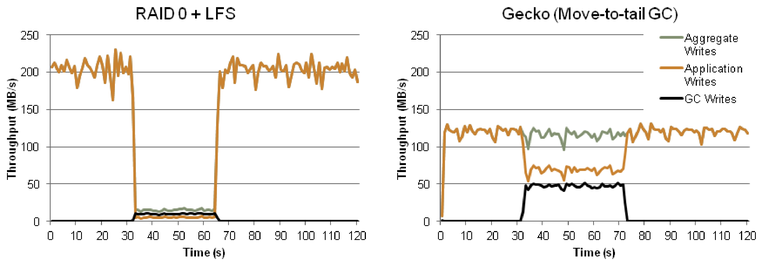
Metadata and Persistence

- In memory logical-to-physical map
  - 4-byte entries per page
  - 8GB for 8TB storage
- In flash physical-to-logical map
  - Maintains persistence
  - Flushed to flash every 1024 page writes

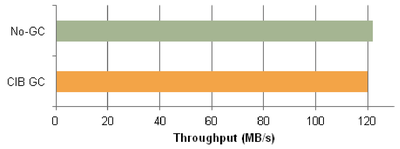


Evaluation

- 2-Disk RAID0+LFS vs Gecko under GC



- Compact-in-body GC



Summary

- Log-structured designs
  - Oblivious to write-write contention
  - Sensitive to GC-write and read-write contentions
- Gecko fixes GC-write and read-write contentions
  - Separation of log tail and head using chain logging
  - Use of tail disk cache
- Flash re-enables log-structured designs
  - As a read cache
  - As a persistent metadata store