MorphStore: A Local File System for Big Data with Utility-driven Replication and Load-adaptive Access Scheduling

Eric Villaseñor, Timothy Pritchett, Jagadeesh Dyaberi*, Vijay Pai, Mithuna Thottethodi

Purdue University

*Saavn, LLC

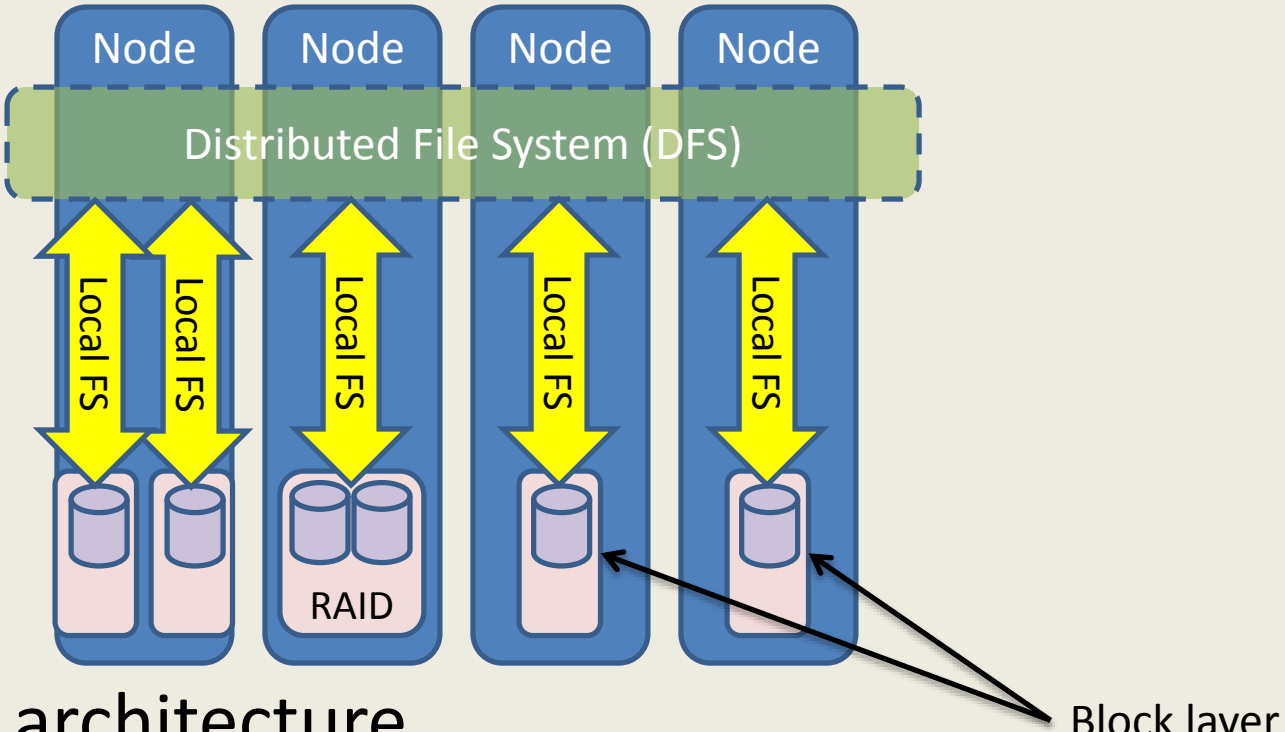
MSST 2014

Motivation

- Storage performance is important to overall performance
- Local large file (10MB-1GB+) handling is important
  - Big Data workloads on order of Petabytes

Goal: Improve local large file access

Context



- Common architecture
  - Global (distributed) file system (e.g., HDFS)
  - Node specific local file system

Opportunity

- Static replication choice
  - E.g., RAID-1 => static mirroring of all content
  - Unnecessary overheads
    - Wasted capacity (replication of low-popularity files)
    - Wasted bandwidth (replication of write-heavy files)
- Idea: **Allocate replication capacity to files with most "utility" (popular, read-mostly)**
- Static access scheduling techniques
  - Striping : maximize use of intra-request parallelism
    - Striping => striped access for all reads/writes
    - Ideal for low-load levels
  - Replication : maximize use of inter-request parallelism
    - Replication => steer reads to single replica/ write to all replicas
    - Ideal for high load-levels with read-mostly accesses
- Idea: **Adapt access scheduling to load level**

Contributions

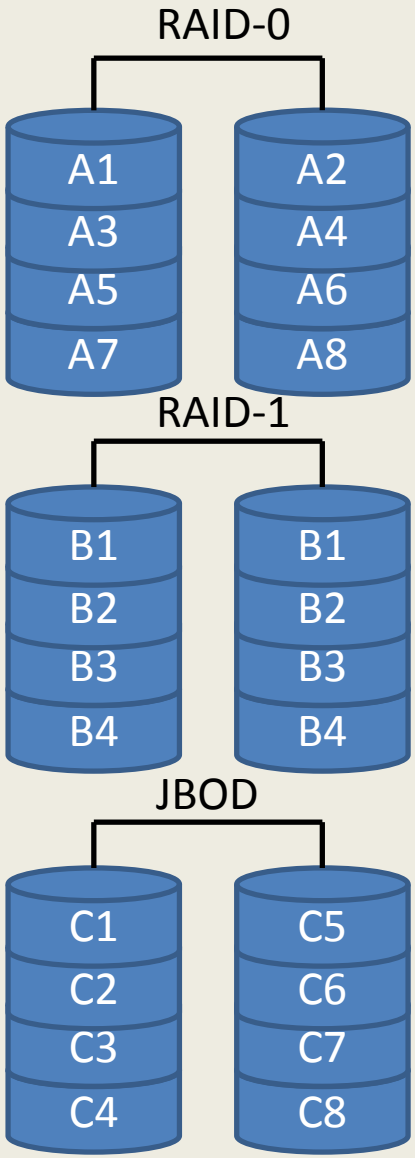
- MorphStore: New File System
- Utility-based replication strategy
  - Profile/expectation-guided
  - Selectively replicates data within a given capacity
- Load-adaptive access scheduling
  - Leverage replicas for both striping and steering
  - Performs closer to the best of static striping/replication strategies

Outline

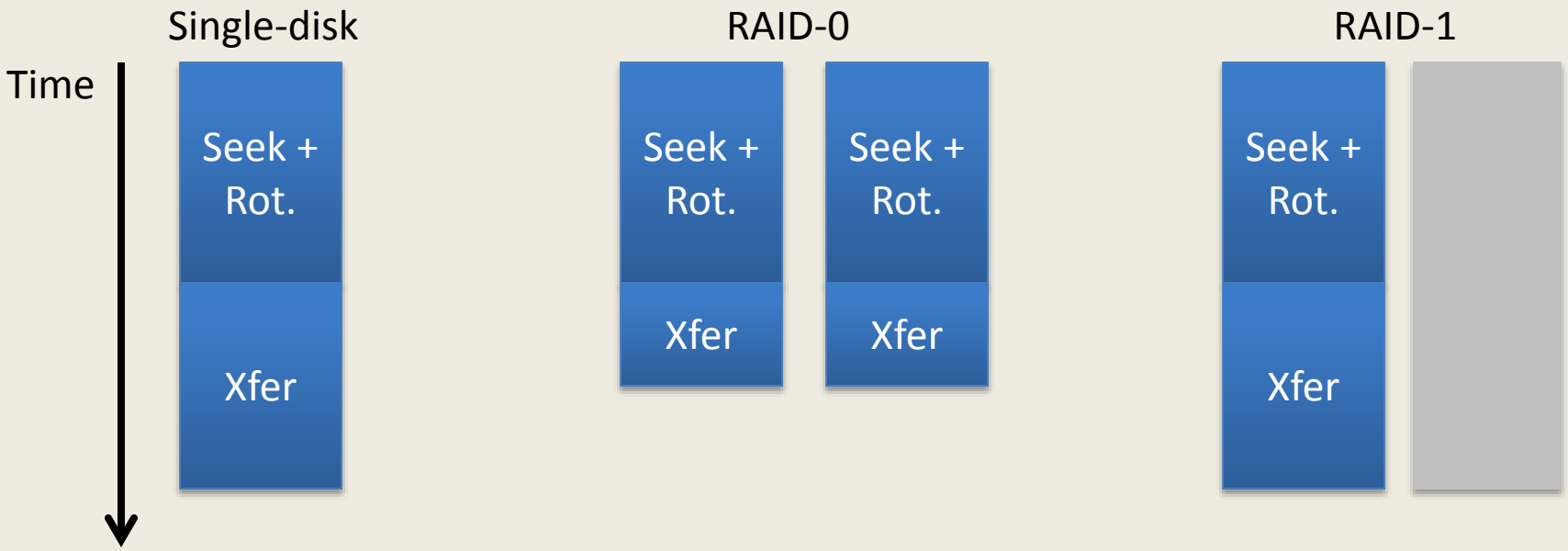
- **Background (RAID Comparison)**
- MorphStore
  - Utility-driven Replication
  - Load-adaptive Access Scheduling
- Implementation and Evaluation
- Results

Redundant Arrays of Inexpensive Disks

- RAID-0
  - Accesses **striped** across disks
  - Bandwidth at low loads
- RAID-1
  - Accesses **steered** to disks
  - Bandwidth at high loads
  - Reliability
- JBOD(Just a Bunch Of Disks)
  - Concatenation
  - No replication/striping



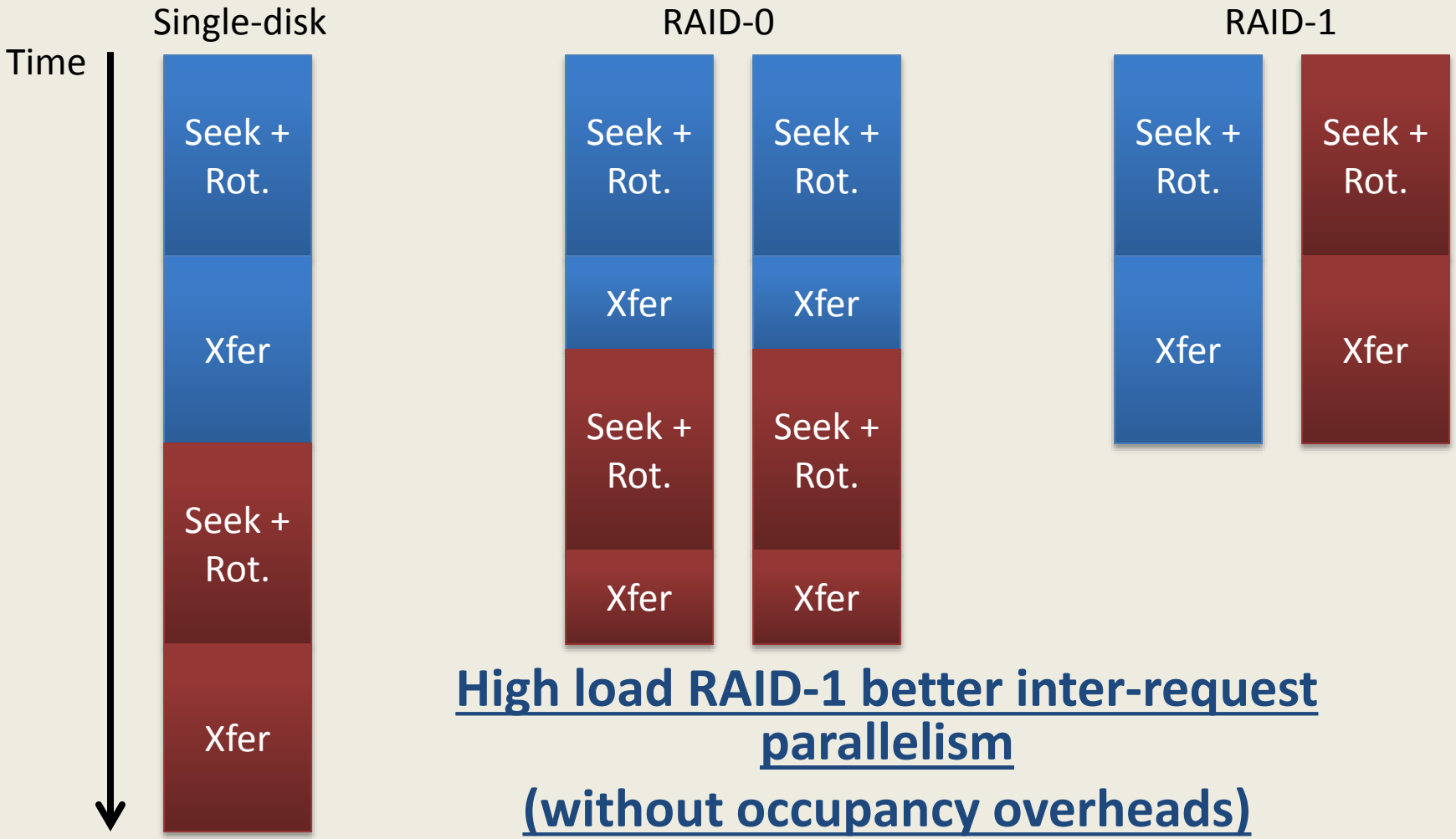
Low Load RAID Impact



Low load RAID-0 achieves better intra-request parallelism

(although with higher disk occupancy)

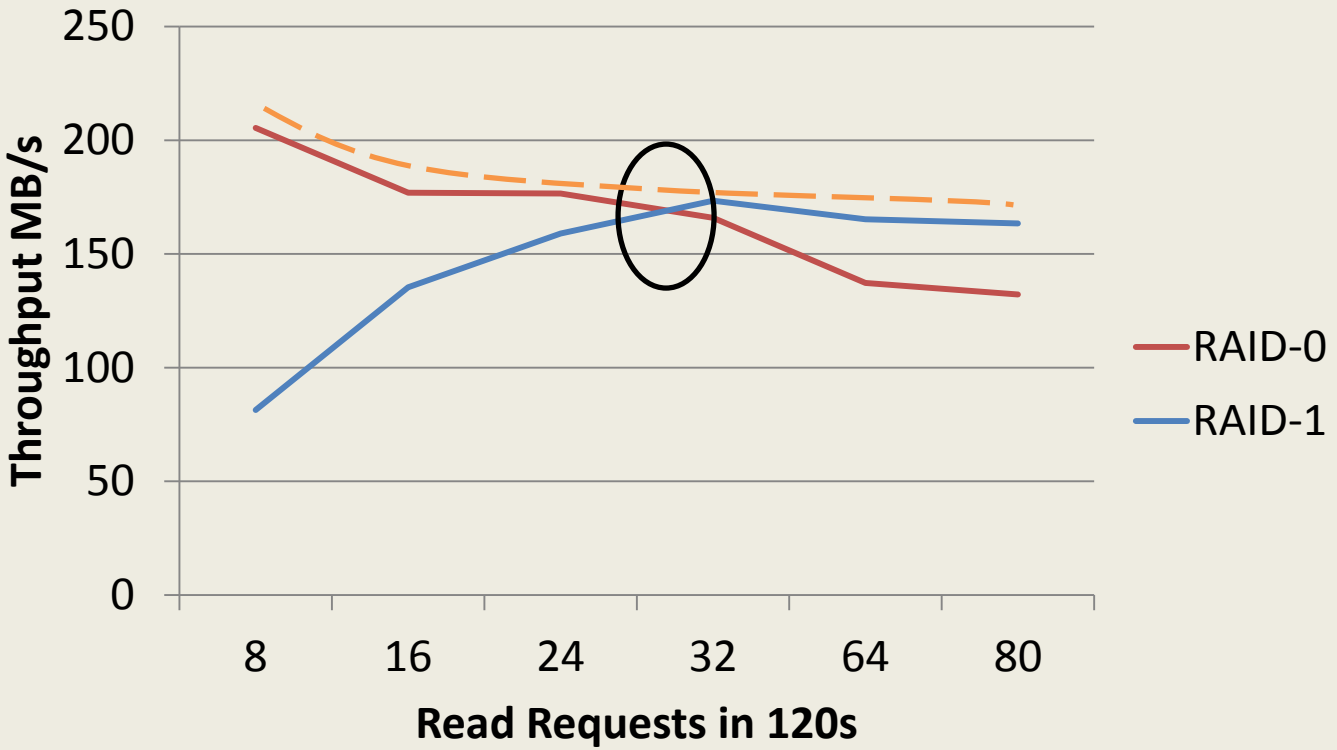
High Load RAID Impact



High load RAID-1 better inter-request parallelism

(without occupancy overheads)

RAID performance



- RAID-0: 4 way stripe over 4 disks (~50MB/s per disk)
- RAID-1: 4 way mirroring
- Sustained 2GB file accesses at varying (X-axis) load level

Outline

- Background (RAID Comparison)
- **MorphStore**
  - **Utility-driven Replication**
  - Load-adaptive Access Scheduling
- Implementation and Evaluation
- Results

Usage Model

- Monitor file usage
  - Reads/writes to files
  - Assumption: Profile data is predictive
  - In general, expectation can be from any source
- Periodically analyze to arrive at replication plan
  - Utility-driven replication
  - Input: Files and their access stats
  - Output: Number of replicas per file
    - Placement: Random, unique disk
- Replicate
  - Exploit diurnal cycles
  - Late night “performance tuning” maintenance

Utility-driven Replication

- Key Observation
  - **Static replication has high capacity/write overhead**
- RAID-1
  - High capacity cost (wasted for low popularity files)
  - High write cost (performance penalty for write-heavy files)
- Utility-driven Replication
  - Trade capacity for performance based on utility
  - Maximize benefit (performance) while minimizing cost (capacity/write overhead)

What is the replication utility for a given file?

Incremental File Utility

$$\left[\left(\frac{R}{K}+WK\right)-\left(\frac{R}{K+1}+W(K+1)\right)\right]$$

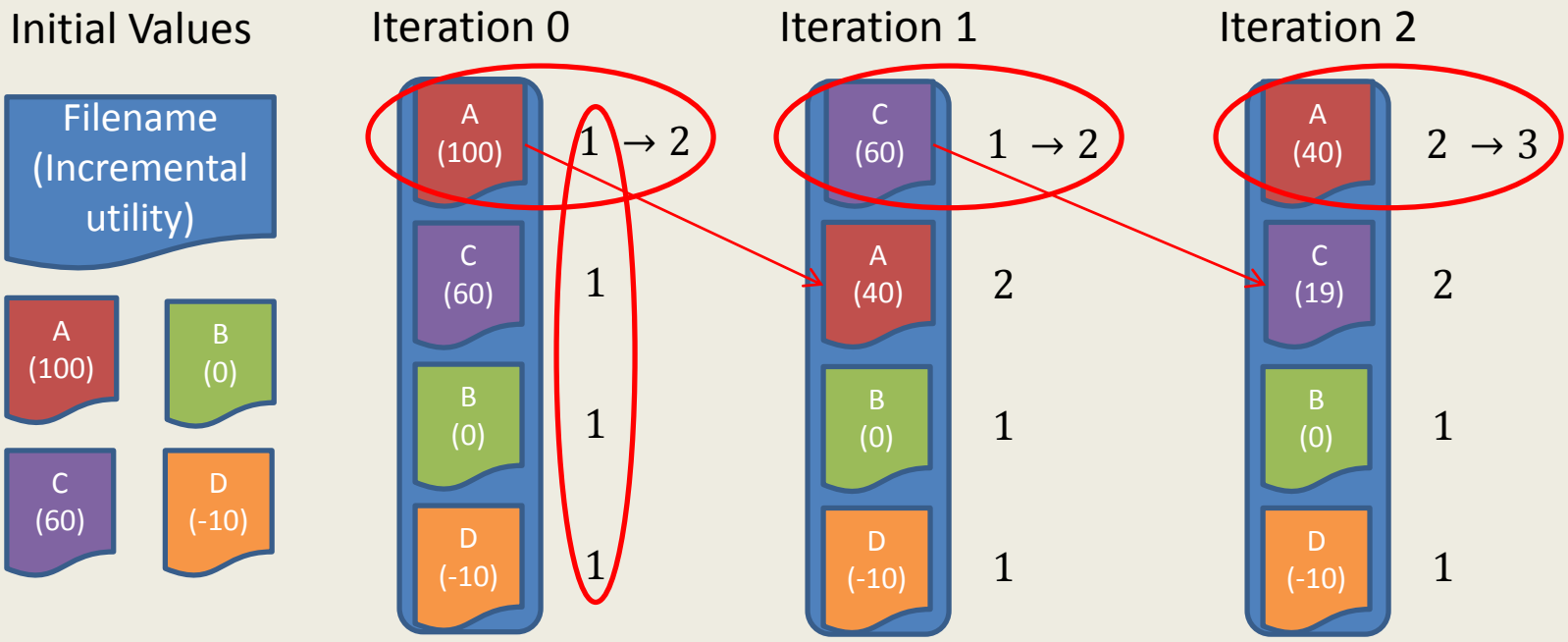
$R = reads$

$W = writes$

$K = replicas$

- Incremental utility of adding the (K+1)th replica
- File reads distributed over all replicas (**positive utility**)
- File writes sent to all replicas (**negative utility**)

Utility-driven Replication Algorithm



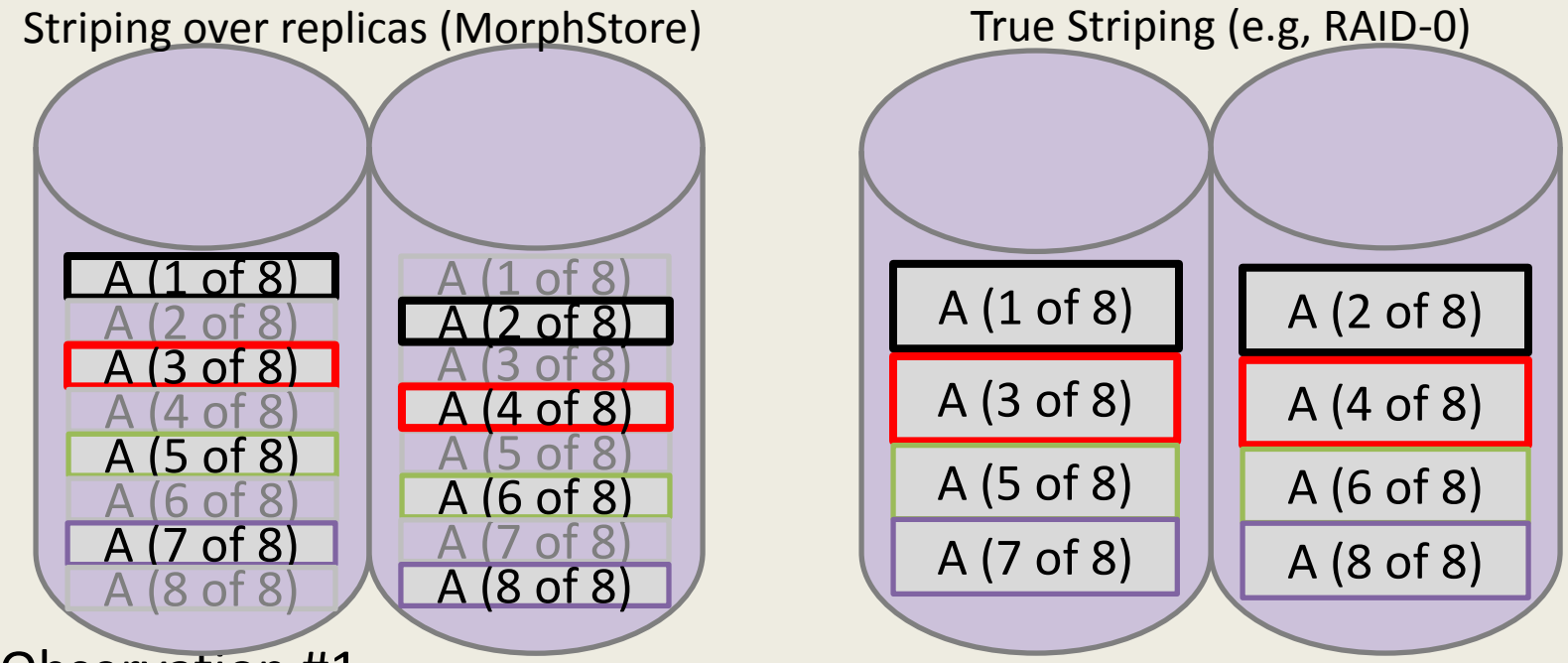
- Greedy replication in utility order
  - Highest (positive) utility first
  - Utility recomputed after replica allocation

Greedy allocation based on global ordering

Outline

- Background (RAID Comparison)
- **MorphStore**
  - Utility-driven Replication
  - **Load-adaptive Access Scheduling**
- Implementation and Evaluation
- Results

Striping over Replicas



- Observation #1
  - **Replicas may be leveraged for striping**
- Not as good as RAID-0, but still helps
  - Parallelism (like RAID-0)
  - Reduced locality (unlike RAID-0)

Load-adaptive Access Scheduling

- Observation #2
  - **Capturing unused bandwidth requires load-adaptive processing**
- Load detection
  - Simple threshold mechanism of average open **inodes** in recent time window
- Scheduling
  - Steer at high loads (RAID-1 inter-request)
  - Stripe at low loads (RAID-0 intra-request)

Summary

- MorphStore's components
  - Utility-driven replication
    - Time-scale of days
    - Determine which files to replicate and to what degree
  - Load-adaptive access scheduling
    - On every access
    - Determines whether to target inter-request or intra-request parallelism.
    - Rule: Inter-request if available; Intra-request if not.

Outline

- Background (RAID Comparison)
- MorphStore
  - Utility-driven Replication
  - Load-adaptive Access Scheduling
- **Implementation and Evaluation**
- Results

Implementation

- MorphStore file system modification
  - Used ext2 as base
- From multiple device (MD) layer
  - Number of disks
  - Size of disks
- Meta-data storage
  - On-disk: extended attributes
  - In-memory: small structure attached to VFS inode
- Load-level detection window
  - Circular buffer to log open inodes on kernel timer.

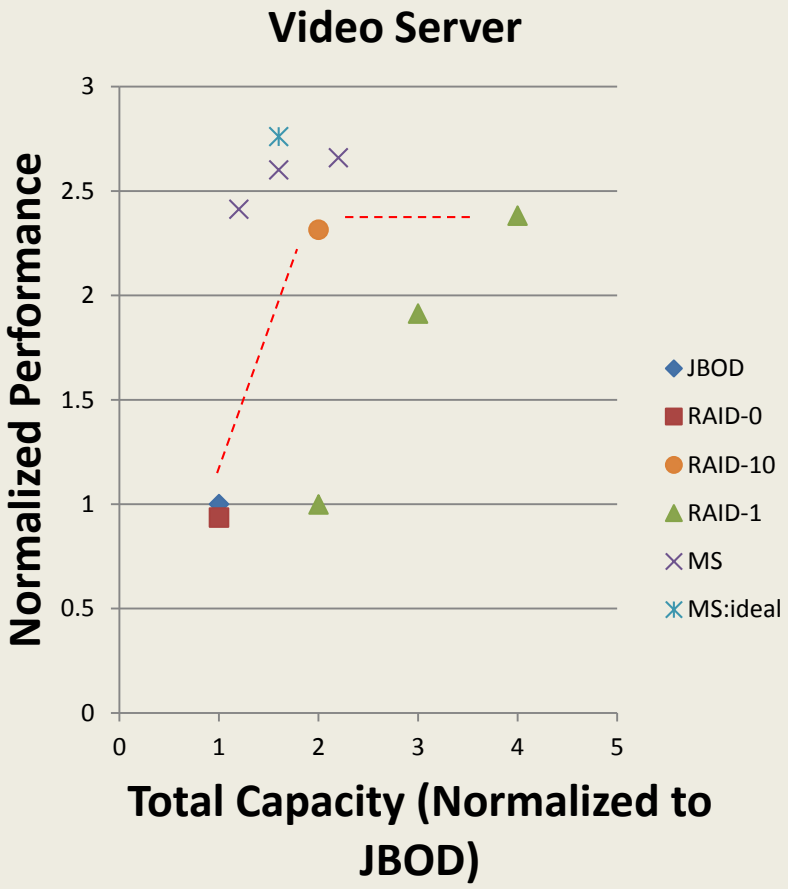
Evaluation

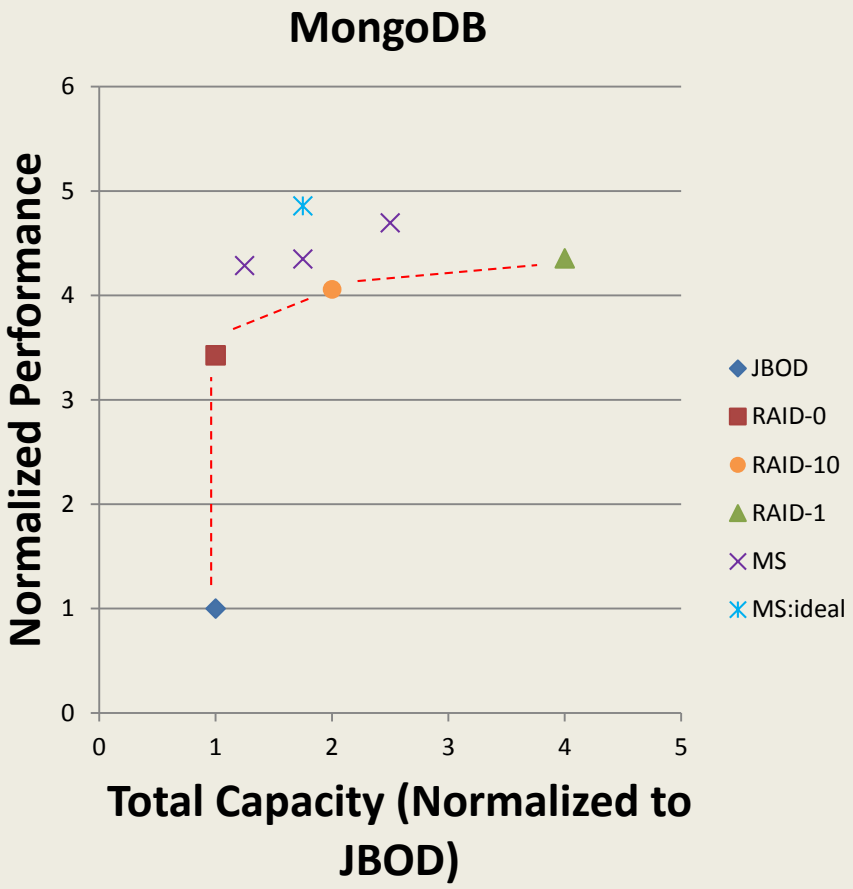
- System
  - Itanium 2 (2 cores)
  - 3 GB RAM
  - 4x500GB drives (~50 MB/s bandwidth)
  - File systems
    - Ext2 (base)
    - MorphStore (modified Ext2)
  - RAID levels (Kernel 3.3 MD driver)
    - JBOD/RAID-0/RAID-1/RAID-10
- Benchmarks and measurement tools
  - Filebench
    - Video server and MongoDB (7 runs)
    - File system measurement
  - IOStat
    - Device measurement (In paper)

Outline

- Background (RAID Comparison)
- MorphStore
  - Utility-driven Replication
  - Load-adaptive Access Scheduling
- Implementation and Evaluation
- **Results**

Pareto Frontiers

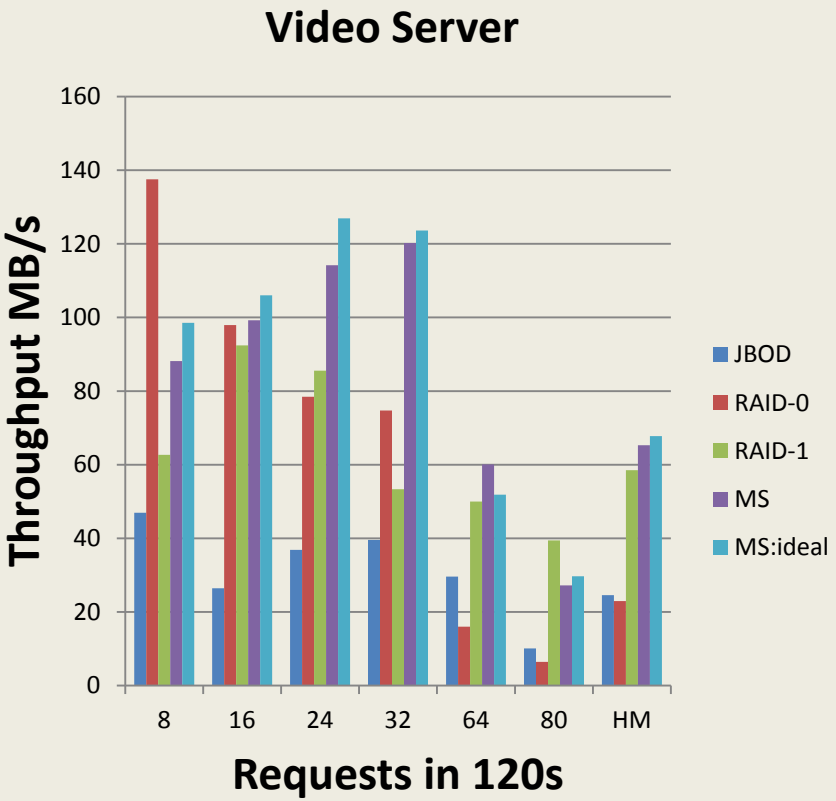


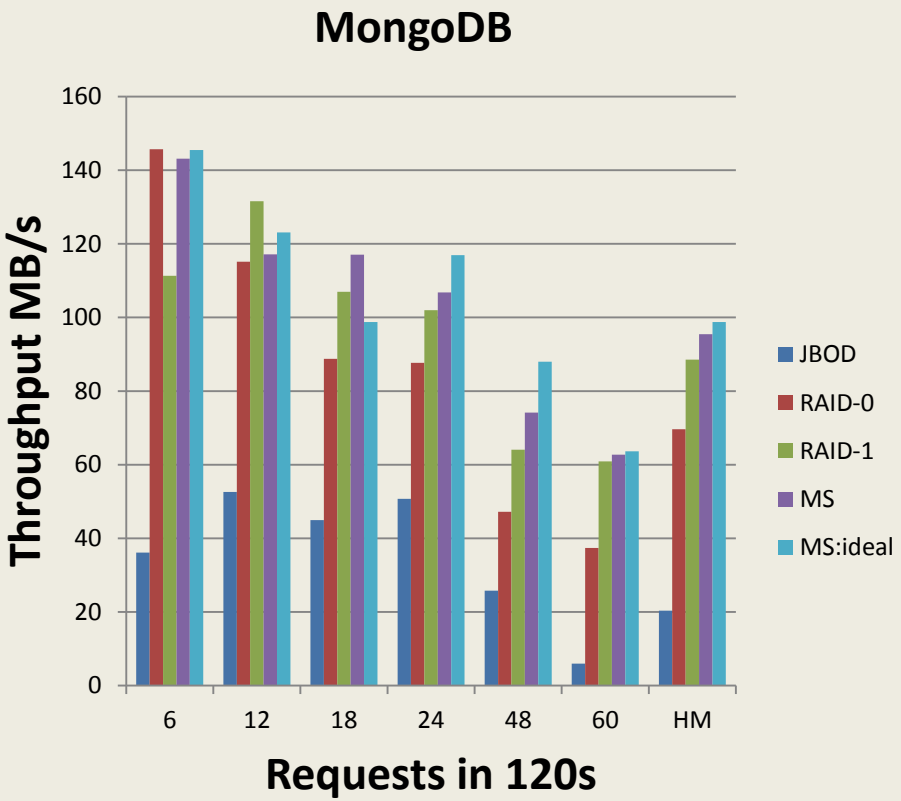


- 2.84X/1.12X better performance than RAID-0/1 with 1.60X capacity overhead for Video Server
- 1.27X/1.00X better performance than RAID-0/1 with 1.75X capacity overhead for MongoDB

CONTRAST with 4X capacity overhead for RAID-1

File System Results





Closer to RAID-0 at low loads; closer to RAID-1 at high loads

Conclusion

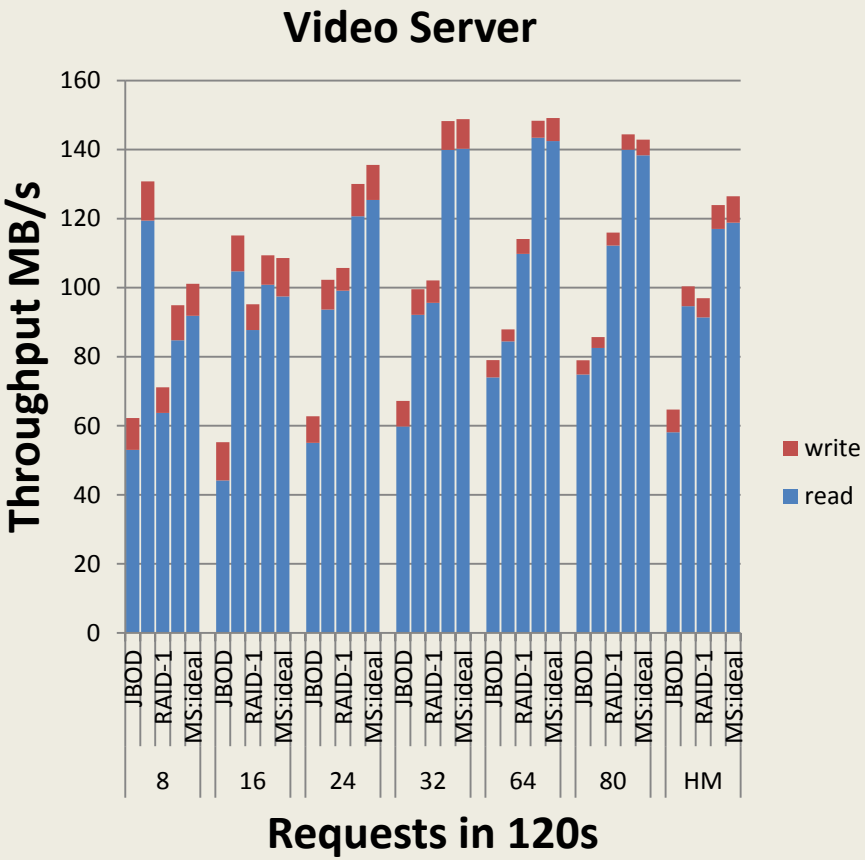
- MorphStore better utilizes device bandwidth via
  - Utility-driven replication
  - Load-adaptive access scheduling
- Capacity performance trade off
  - 2.84X/1.12X better performance than RAID-0/1 with 1.60X capacity overhead for Video Server
  - 1.27X/1.00X better performance than RAID-0/1 with 1.75X capacity overhead for MongoDB

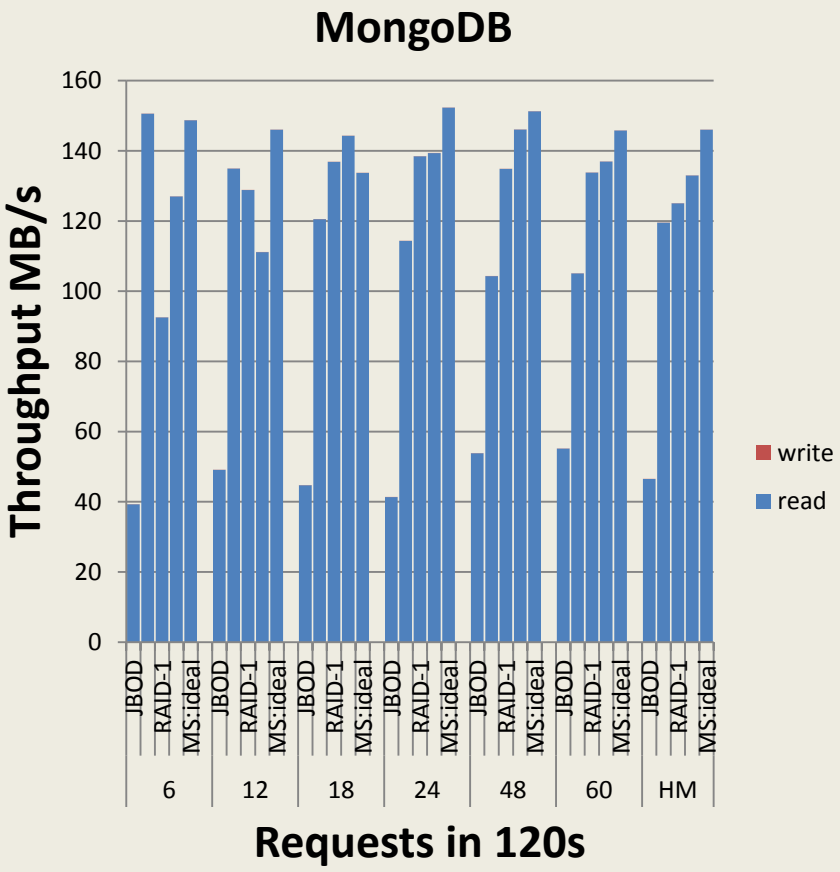
MorphStore extends the capacity/bandwidth Pareto frontier

Questions

Extra Slides

Device Level Results





High device throughput

Optimality

- Greedy choice is optimal
  - Given definition of “utility”
  - Assumes good placement (only true conflicts)
  - Equi-sized items
- In our benchmarks we maintained equi-sized items
  - Utility-driven replication is optimal
- Not necessarily true for general case
  - Non equi-sized items
  - Random placement may have false conflicts

Linux Software RAID-10

- Adds Flexibility over standard RAID-10
  - Stripe over RAID-1
  - Near/far/offset layouts
- Still a static technique
  - All files replicated
    - VS. selective replication (popular read-mostly files)
  - Stripe maximally over disks
    - VS. striping over replicas (variable number of replicas)

Lustre

- Global file system
  - Site wide (above local fs)
- Ability to stripe
  - Multiple object storage targets (OST)
  - Dynamically set (FS, dir, file)
- Static stripe
  - No adaptive scheduling based on load
  - Overhead reliability at global level

Utility-driven Replication