DiskReduce Analysis

Bin Fan

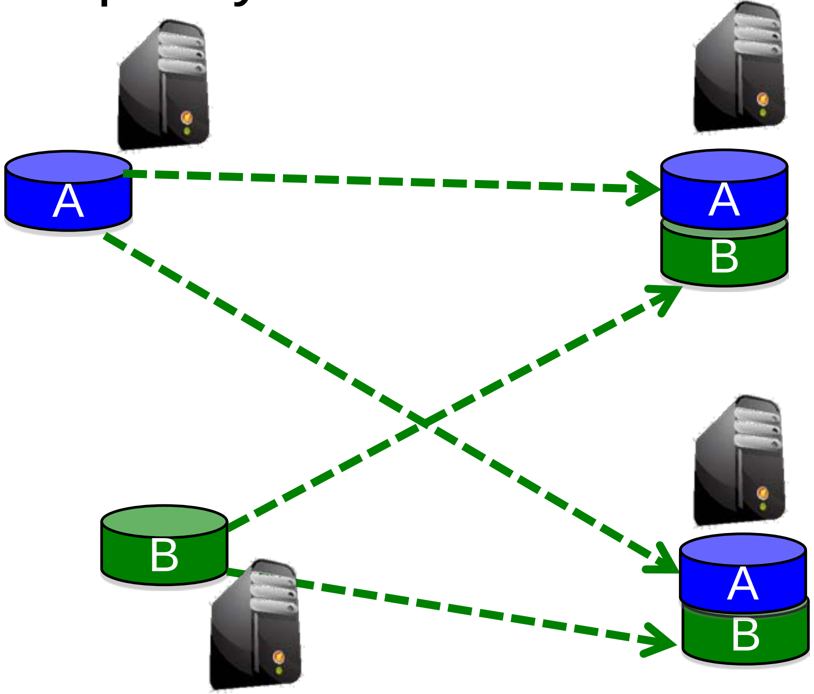
Wittawat Tantisiriroj, Lin Xiao, Garth Gibson

PARALLEL DATA LABORATORY

Carnegie Mellon University

Motivation

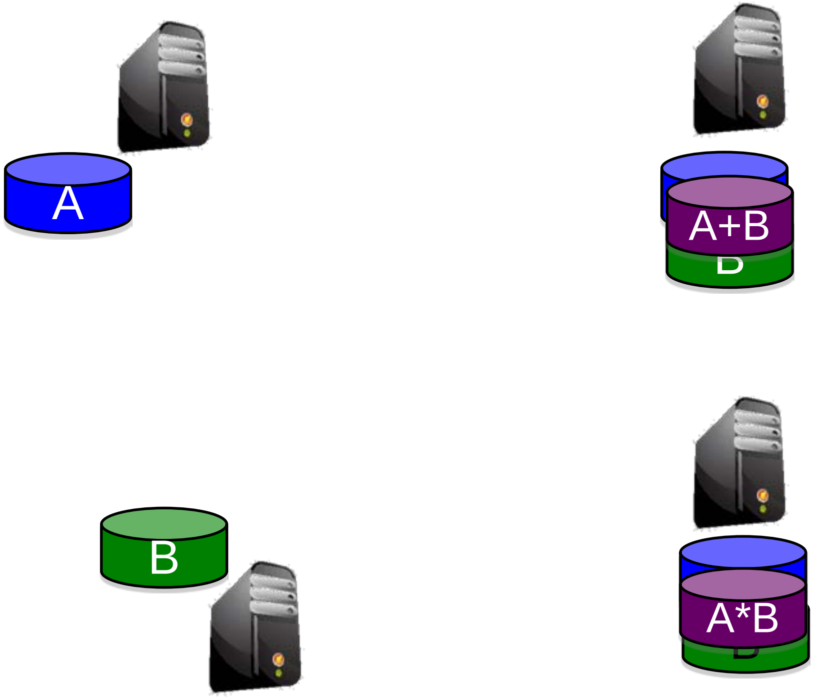
- GFS/HDFS use replicated data:
  - + tolerate node faults
  - + higher read bandwidth
  - -- storage capacity overhead: 200% for triplication



Carnegie Mellon
Parallel Data Laboratory

Basic Idea

- Reduce Overhead by RAID encoding:
  - RAID 6: survive from 2 node failures



Problems

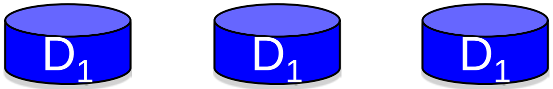
- How much saving in storage
- Performance penalty
- Reliability penalty – covered by Lin
- Implementation issues – covered by Wittawat

Design Choices

- How to group blocks?
  - Per file: blocks from the same file in a RAID set
    - E.g. HDFS-RAID @Facebook
  - Across file: blocks from different files in a RAID set
    - E.g. RAIDTool @Yahoo
- When to encode?
  - Immediate encoding:
    - E.g. Colossus, MSR 2010
  - Background encoding:
    - E.g. RAIDTool

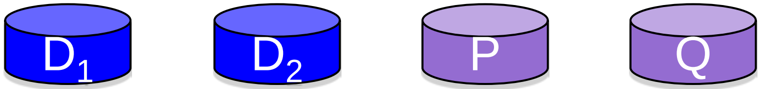
Storage Overhead

- Overhead to store parity blocks
- Triplication:



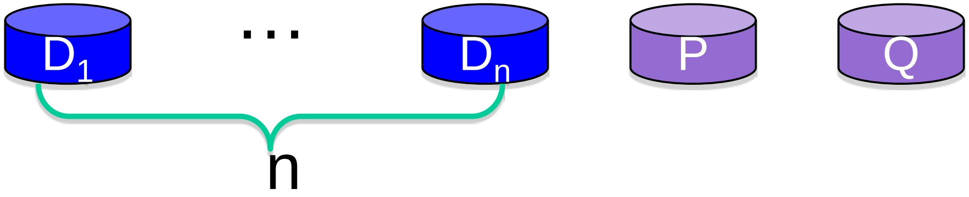
200 %

- RAID 6, group size = 2



100 %

- RAID 6, group size = n



200/n %

- raditional RAID 6: n = 4~20

10 % ~ 50 %

Per-file RAID

- Simple to maintain
  - All blocks in one RAID set deleted together
- Easy to deploy
- Better access permission control
- Used by Panasas , HDFS-RAID

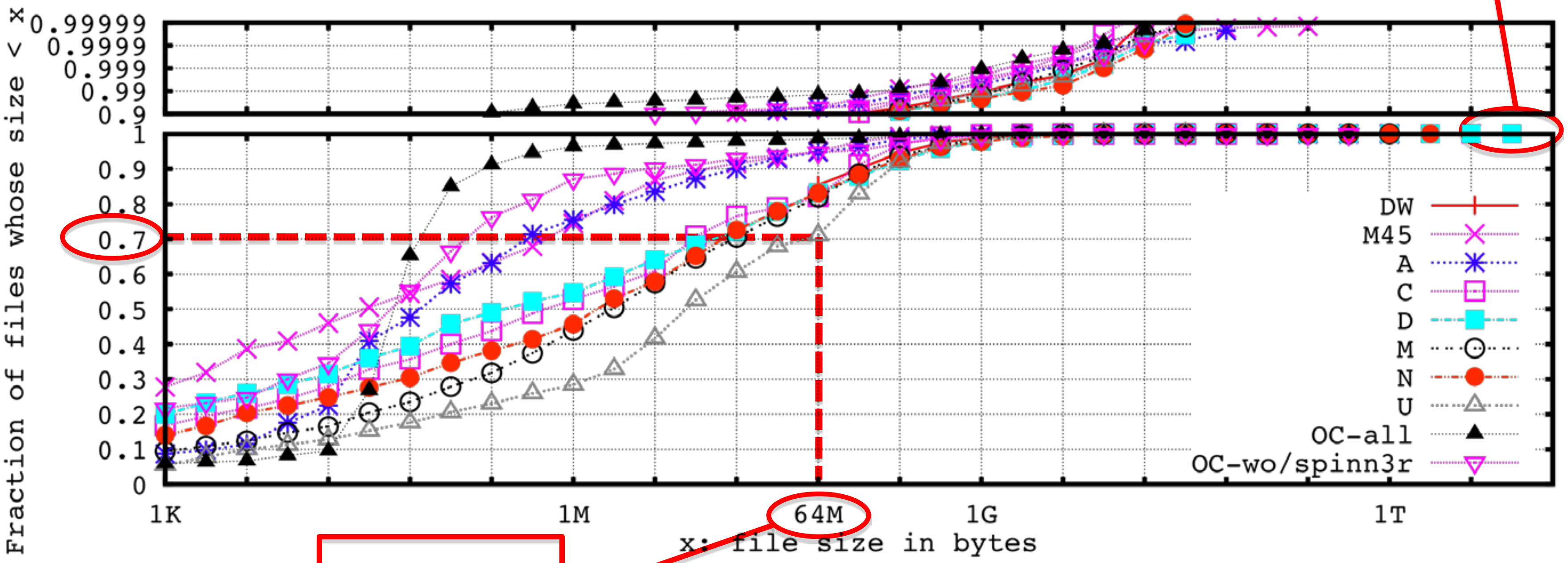
Data from Real Clusters

- HDFS Clusters in CMU, Yahoo!, Facebook

ClusterID	# Nodes	Raw Capacity
DW	2000	21 PB
CMU OC	64	0.25 PB
M45	400	1.5 PB
A	1800	3.8 PB
C	800	3.5 PB
D	3000	6.5 PB
M	2000	6.3 PB
N	3500	10.6 PB
O	1000	7.5 PB
U	1700	13.0 PB

Most Files Smaller than a Single Block

- File size distribution
  - Largest file > 4 TB
  - Average file size: 22 MB to 577 MB
  - 70% files < 64 MB (single block)

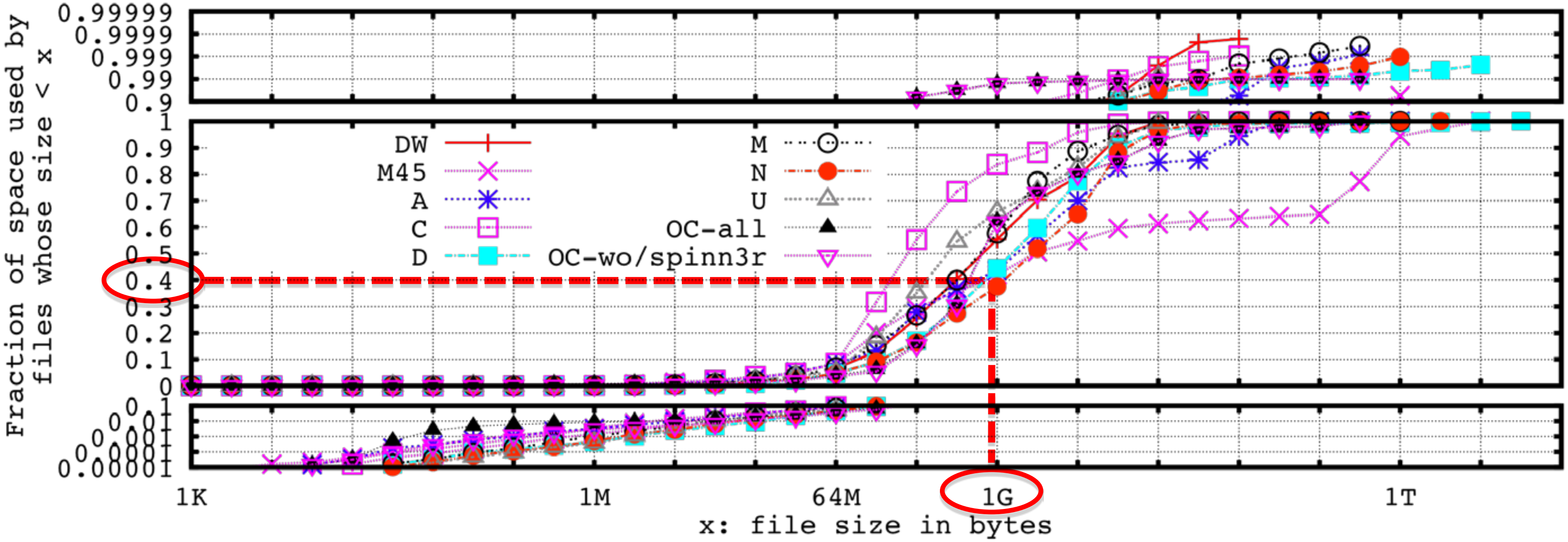


Largest File

One Block

Lots of Capacity Used by “Small” Files

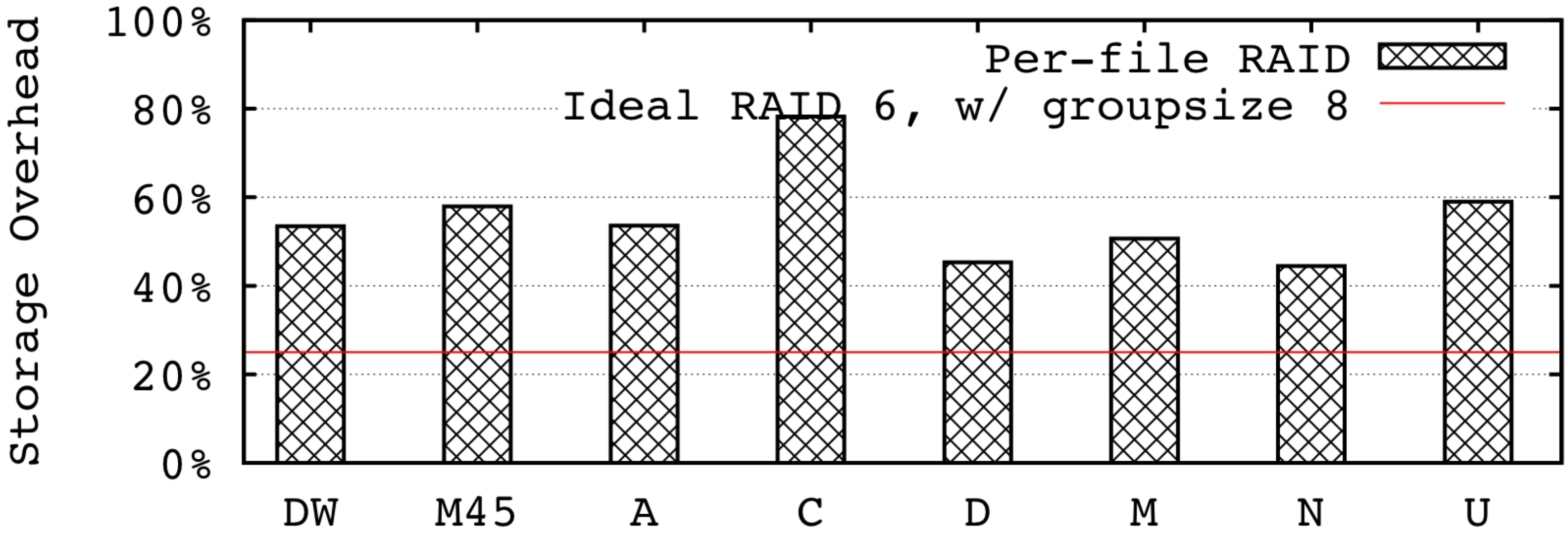
- Storage occupied:
  - 40% storage used by files < 1 GB or 16 blocks



Carnegie Mellon
Parallel Data Laboratory

Per-file RAID Doubles Overhead

- Apply per-file RAID, group size = 8
  - ~ 50% storage overhead for encoding



Per-file RAID in HDFS: storage overhead is still high

Carnegie Mellon
Parallel Data Laboratory

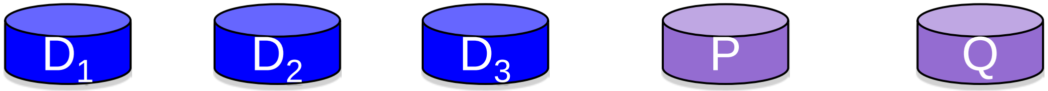
Across-file RAID

+ Low storage overhead:

- More blocks to group

-- Small write problem even with immutable HDFS files

- In traditional RAID: to modify D2:



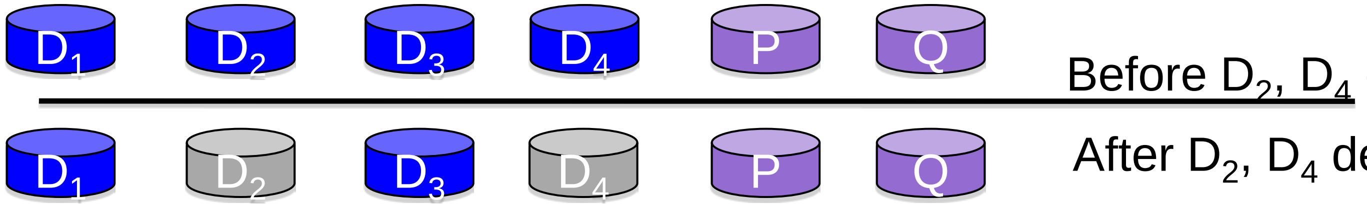
  - Read back D_2, P, Q,
  - Recalculate new parities P', Q' with D_2, D_2'
  - Write back new D_2', P', Q'

- In HDFS: read-modify-write triggered on deletion

Carnegie Mellon
Parallel Data Laboratory

Possible Solutions

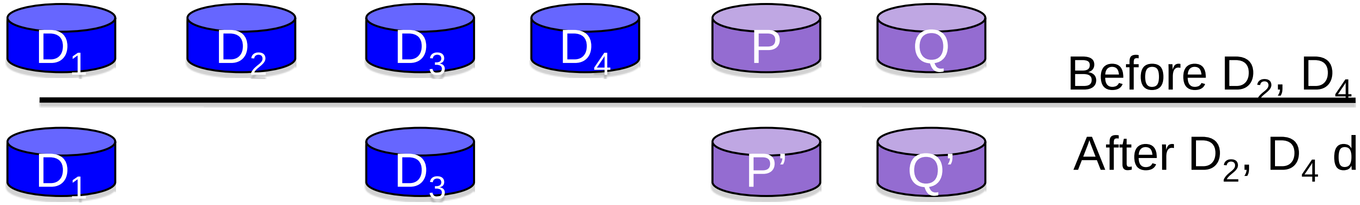
1. Never recalculate parities: live with holes



Before D_2, D_4 deleted

After D_2, D_4 deleted

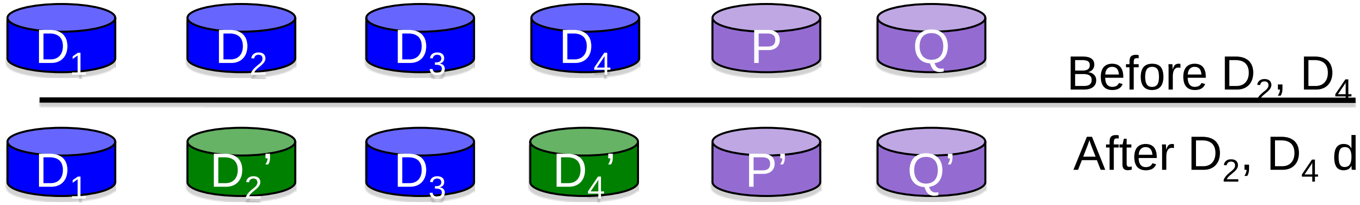
1. Recalculate parities: eliminate holes



Before D_2, D_4 deleted

After D_2, D_4 deleted

2. Replace data & recalculate parities: fill holes



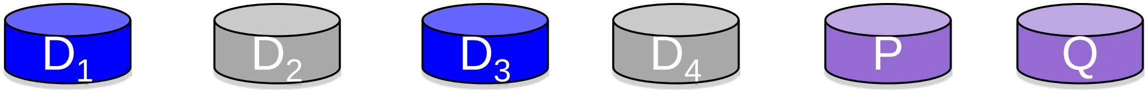
Before D_2, D_4 deleted

After D_2, D_4 deleted

Never Recal. Parities

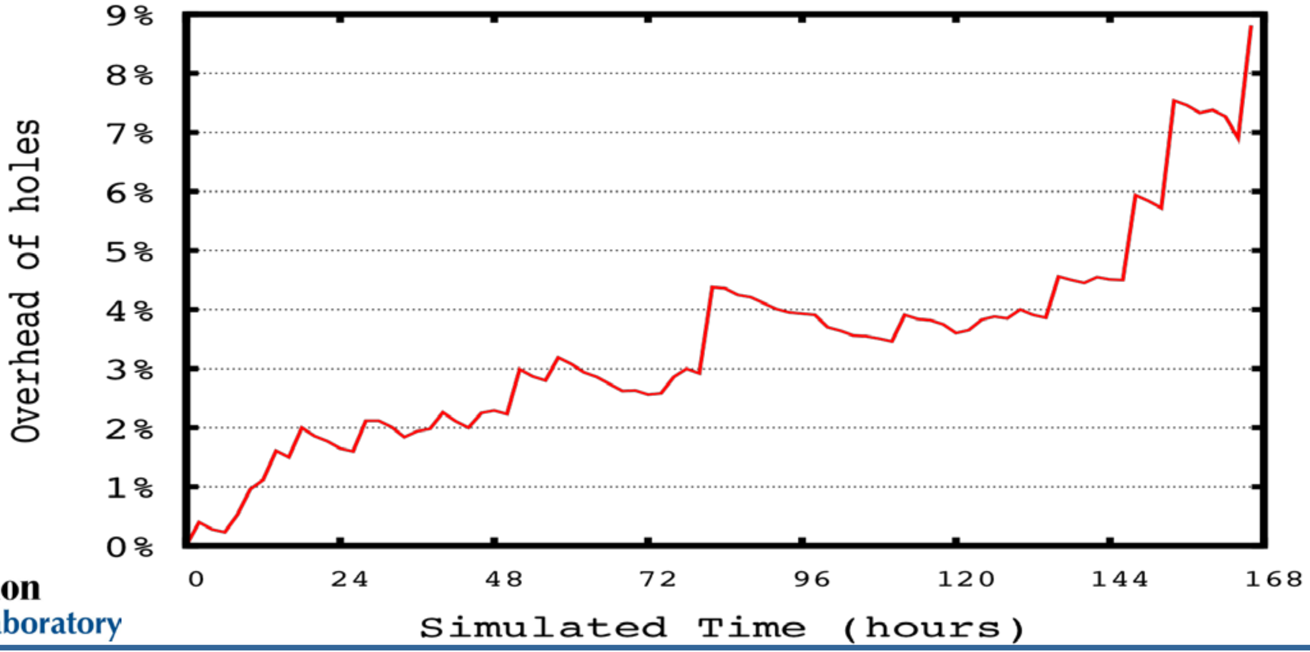
+ Easy, no extra work

-- Too many “holes” in RAID sets



After D_2, D_4 deleted

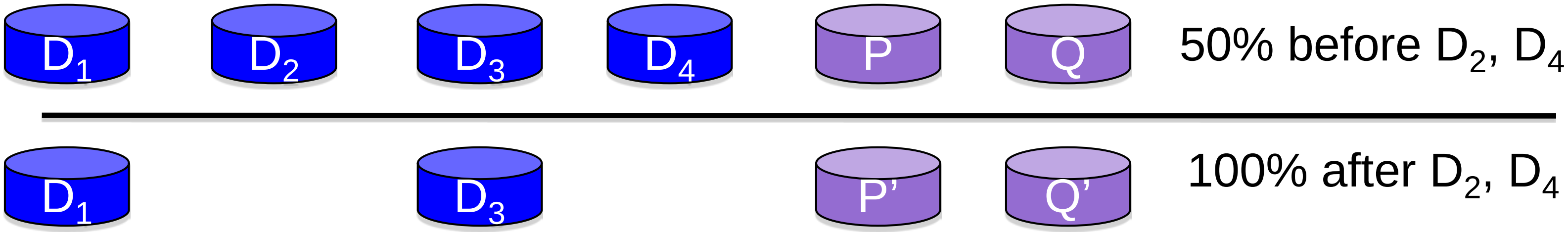
- D_2,D_4 are a hole: deleted but kept in storage
- 50% space unreclaimed



Recalculate Parities

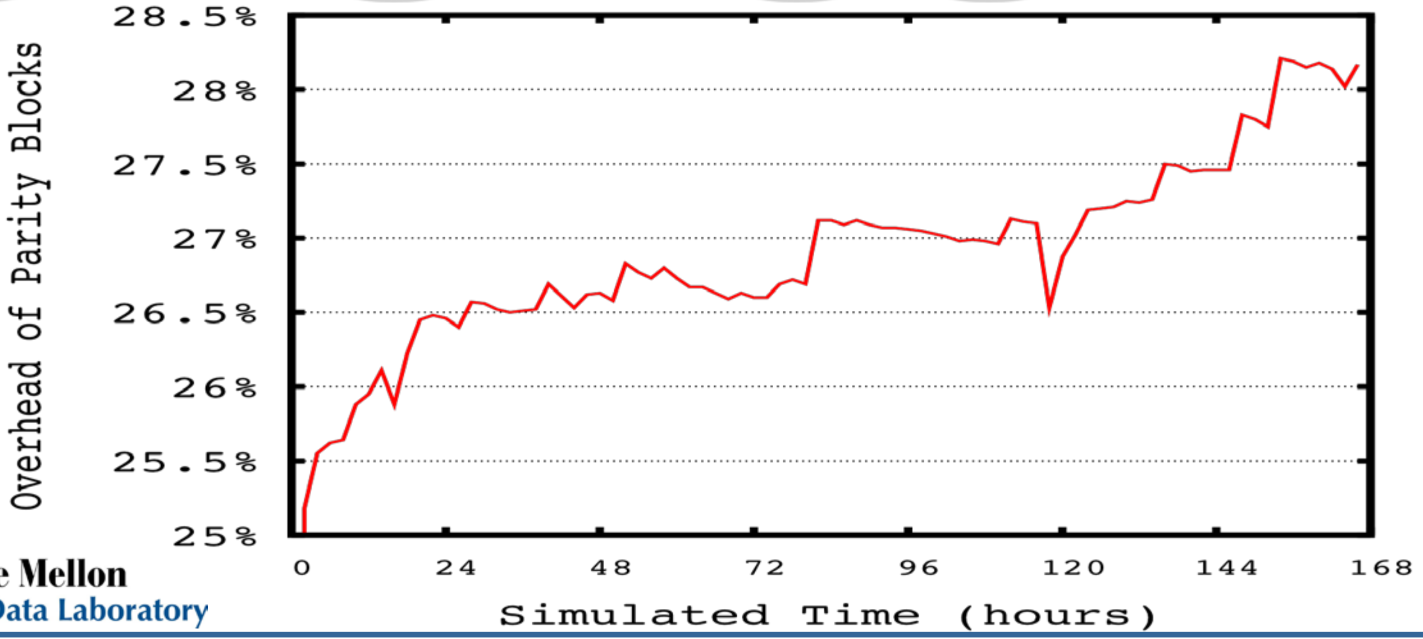
+ Extra cost gets amortized

-- Less data per RAID set: Increased overhead of parity



50% before D_2, D_4 deleted

100% after D_2, D_4 deleted



Replace Data and Recal. Parities

- Low storage overhead, at cost of more IO and network traffic
  - Total Storage Overhead < 26%
  - Each deletion requires reading 0.2 blocks from disk

Across-file RAID in HDFS:

- storage overhead
- pay more work/resource in deletion

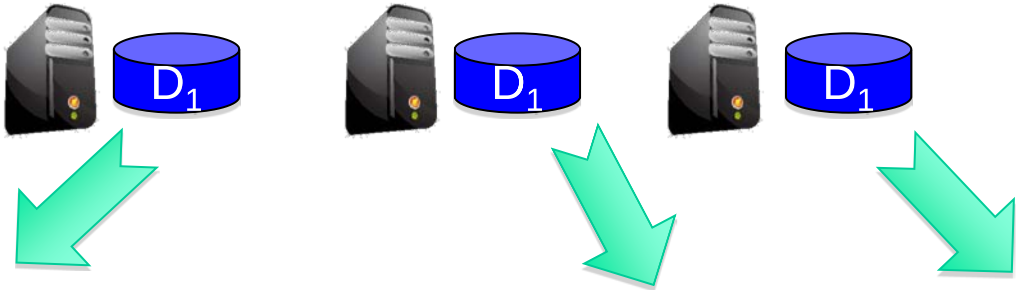
Carnegie Mellon
Parallel Data Laboratory

Problems

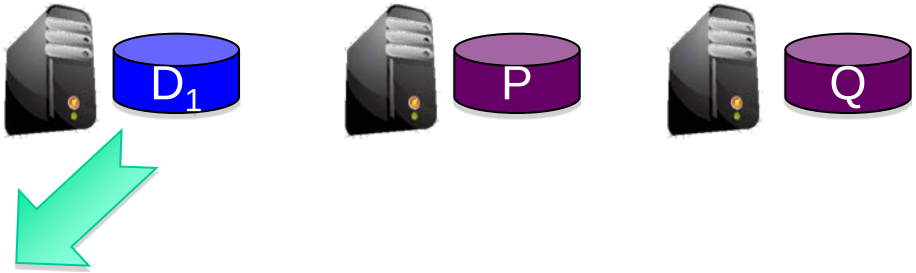
- How much saving in storage
- Performance penalty
- Reliability penalty – covered by Lin
- Implementation issues – covered by Wittawat

Performance Study

- Triplication: three nodes serving the data



- RAID 6: one clear copy + 2 parities



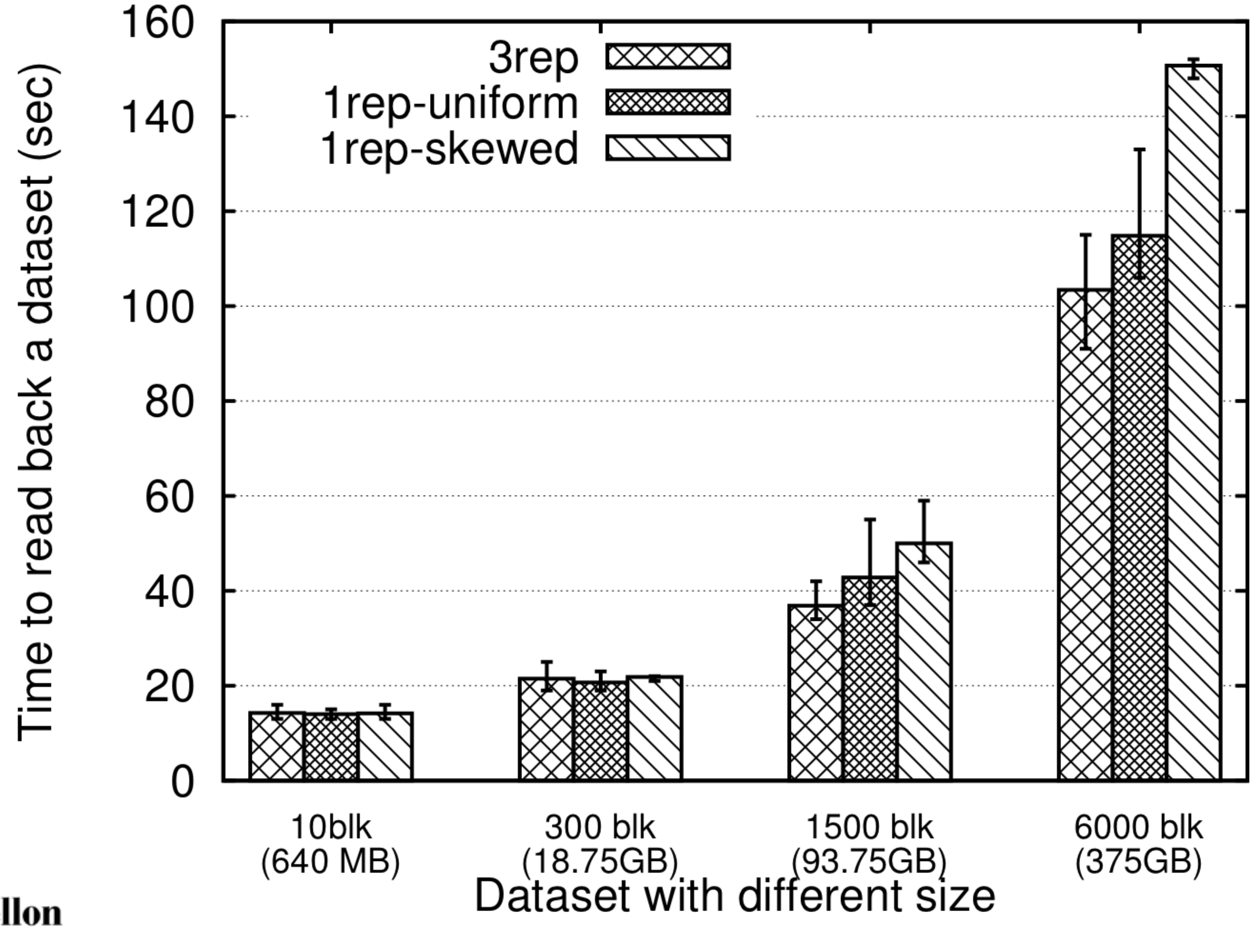
Carnegie Mellon
Parallel Data Laboratory

Benchmark1

- MapReduce job reading the same data
- Each piece of data is read by only one task
- 480 tasks, 60 nodes
- Data set size
  - 10, 300, 1000, 6000 blocks:
- Data set distribution
  - Triplication
  - Single copy, uniform distribution across nodes
  - Single copy, skewed distribution across nodes

Triplication Can Give You Faster Reads

- happens with large & skewed distribution



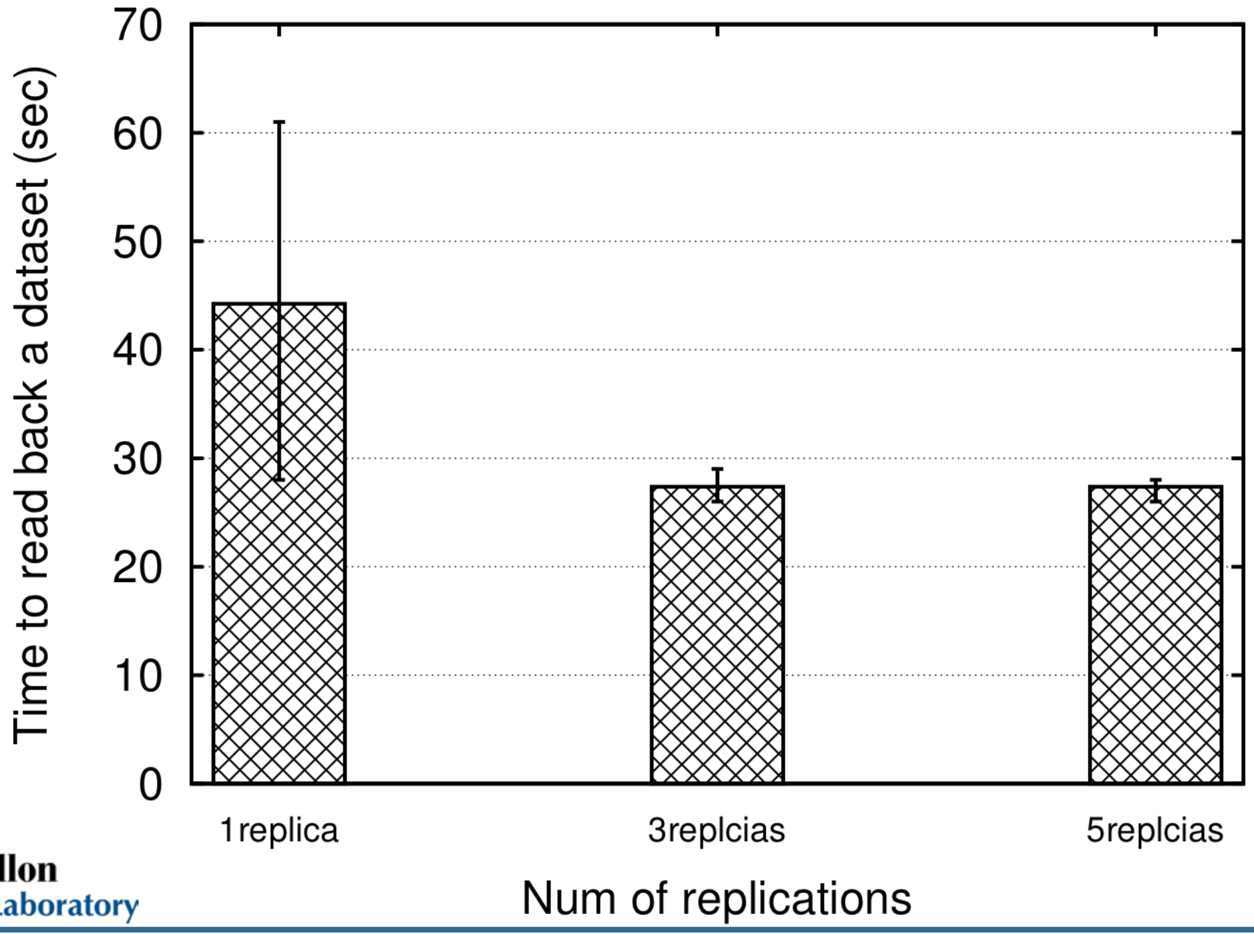
Carnegie Mellon
Parallel Data Laboratory

Benchmark2

- MapReduce job reading the same data
- Each piece of data is read by each task
- 480 tasks, 60 nodes
- Data set size:
  - 2 blocks:
- Data set replication level:
  - Triplication
  - Single copy

More Copies Help Hot-Spot

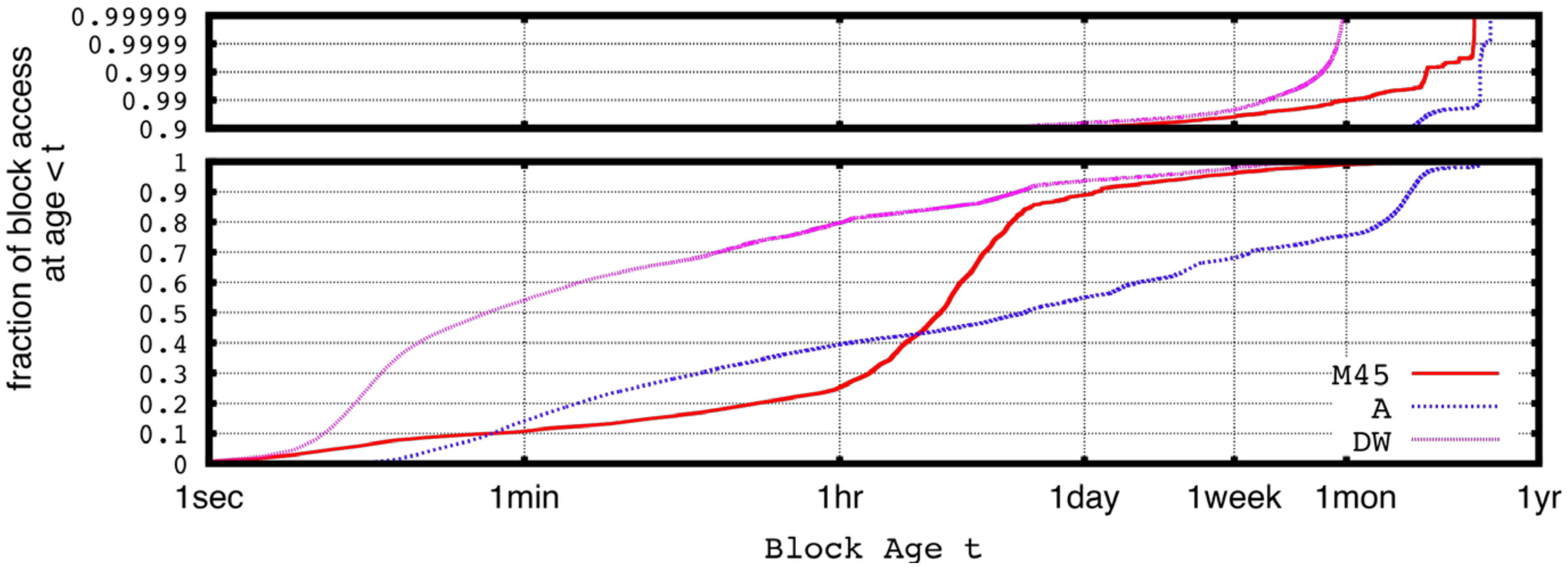
- More disk spindles



Carnegie Mellon
Parallel Data Laboratory

Temporal Locality in DISC workload

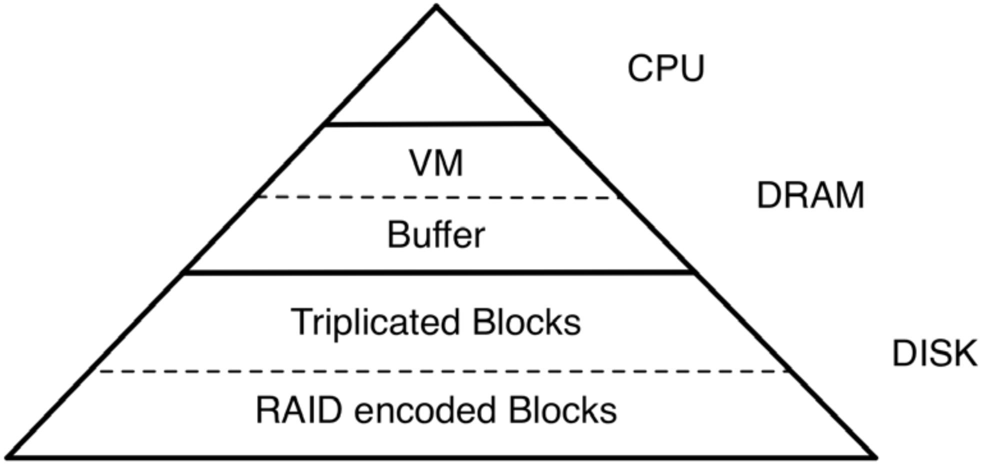
- Most data is accessed when young
- Delayed encoding may benefit performance



Carnegie Mellon
Parallel Data Laboratory

Cache Hierarchy

- Treat triplication as in cache
  - + Exploit temporal locality: most accesses go to triplicated data
  - -- Trade space for performance



Conclusion

- Per-file RAID has high storage overhead
  - Files are “small”
- Across-file RAID has overhead of maintaining RAID consistency
  - “Small write” problem on deletion
  - Extra work needed to update RAID parities
- RAIDed data may have performance penalty
- Temporal locality gives delayed encoding performance benefit