Log Structured File Systems

Learning Outcomes

- An understanding of the performance of Inode-based files systems when writing small files.
- An understanding of how a log structured file system can improve performance, and increase reliability via improved consistency guarantees without the need for file system checkers.
- An understanding of “cleaning” and how it might detract from performance.

Motivating Observations

- Memory size is growing at a rapid rate

⇒ Growing proportion of file system reads will be satisfied by file system buffer cache

⇒ Writes will increasingly dominate reads

Motivating Observations

- Creation/Modification/Deletion of small files form the majority of a typical workload
- Workload poorly supported by traditional Inode-based file system (e.g. BSD FFS, ext2fs)
  - Example: create 1k file results in: 2 writes to the file inode, 1 write to data block, 1 write to directory data block, 1 write to directory inode
    - ⇒ 5 small writes scattered within group
  - Synchronous writes (write-through caching) of metadata and directories make it worse
    - Each operation will wait for disk write to complete.
- Write performance of small files dominated by cost of metadata writes

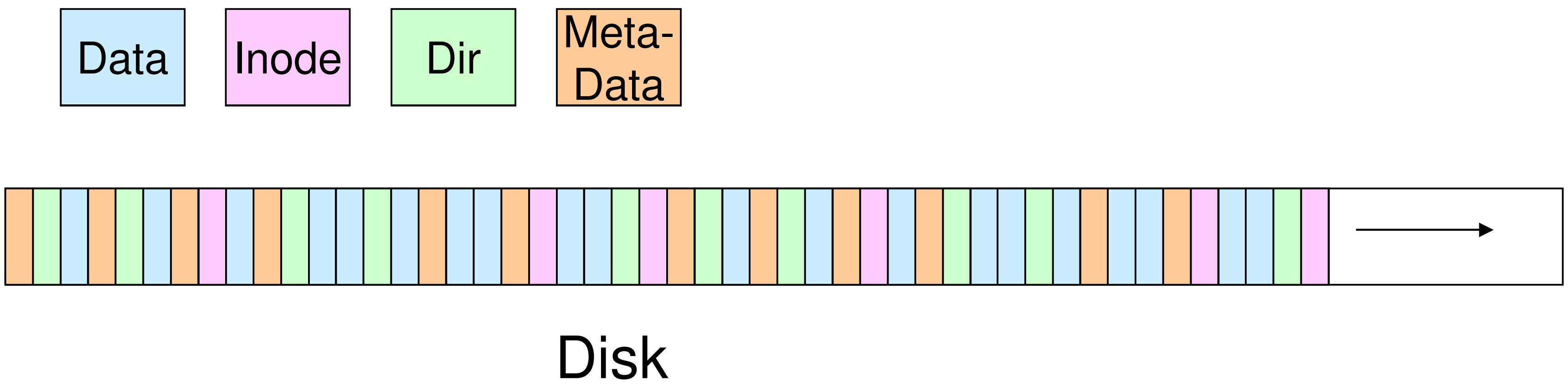
Super Block	Group Descrip-tors	Data Block Bitmap	Inode Bitmap	Inode Table	Data blocks

Motivating Observations

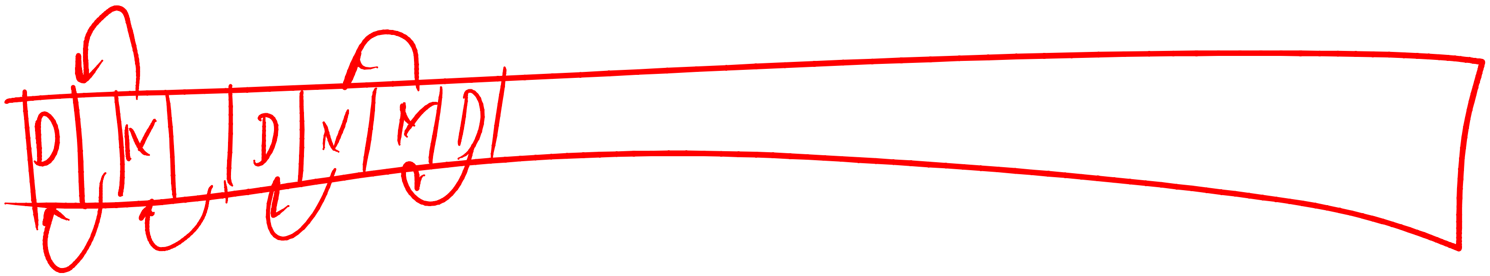
- Consistency checking required for ungraceful shutdown due to potential for sequence of updates to have only partially completed.
- File system consistency checkers are time consuming for large disks.
- Unsatisfactory boot times where consistency checking is required.

Basic Idea!!!

- Buffer sequence of updates in memory and write all updates sequentially to disk in one go.



Example



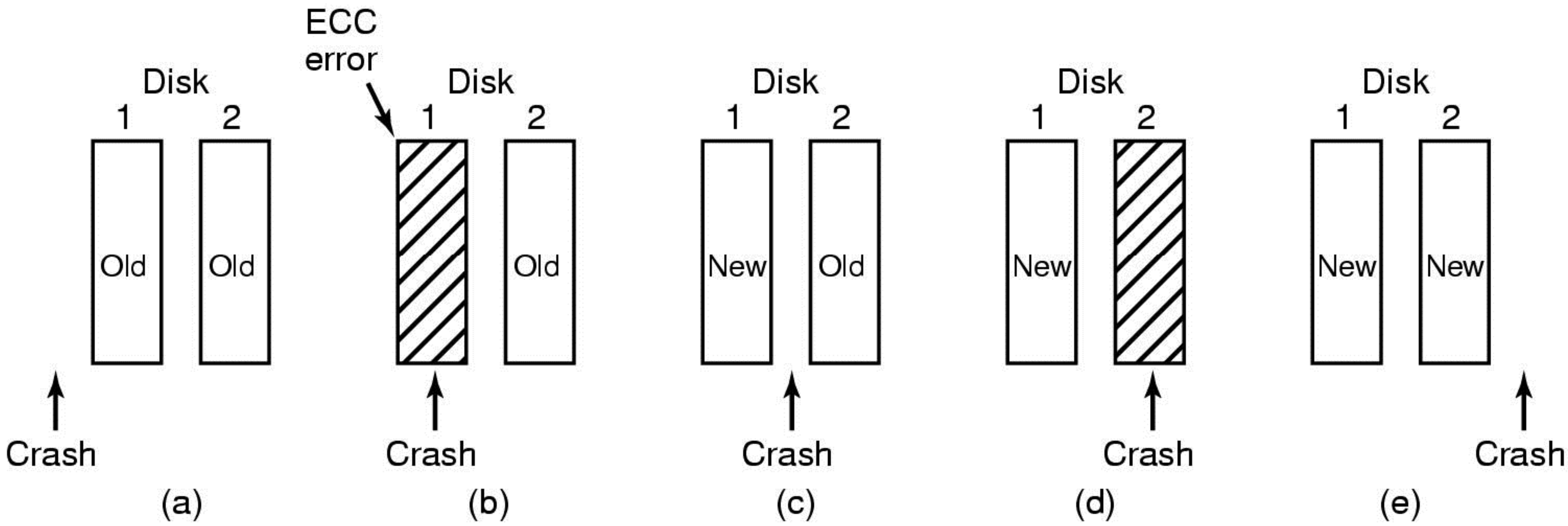
Issues

- How do we now find I-nodes that are scattered around the disk?

⇒ Keep a map of inode locations

  – Inode map is also “logged”
  – Assumption is I-node map is heavily cached and rarely results in extra disk accesses
  – To find block in the I-node map, use two fixed location on the disk contains address of block of the inode map
    - Two copies of the inode map addresses so we can recover if error during updating map.

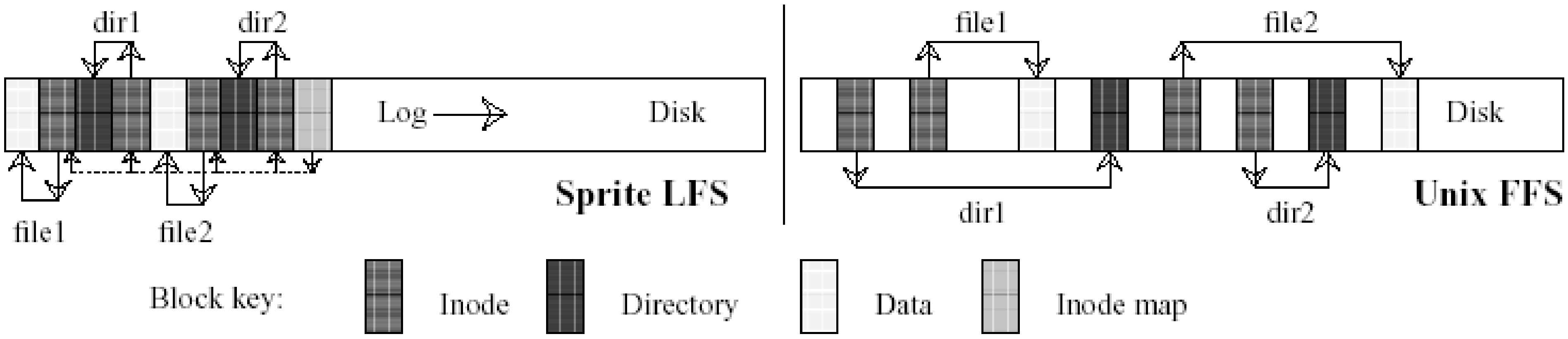
Implementing Stable Storage



- Use two disks to implement stable storage
  - Problem is when a write (update) corrupts old version, without completing write of new version
  - Solution: Write to one disk first, then write to second after completion of first

LFS versus FFS

- Comparison of creating two small files

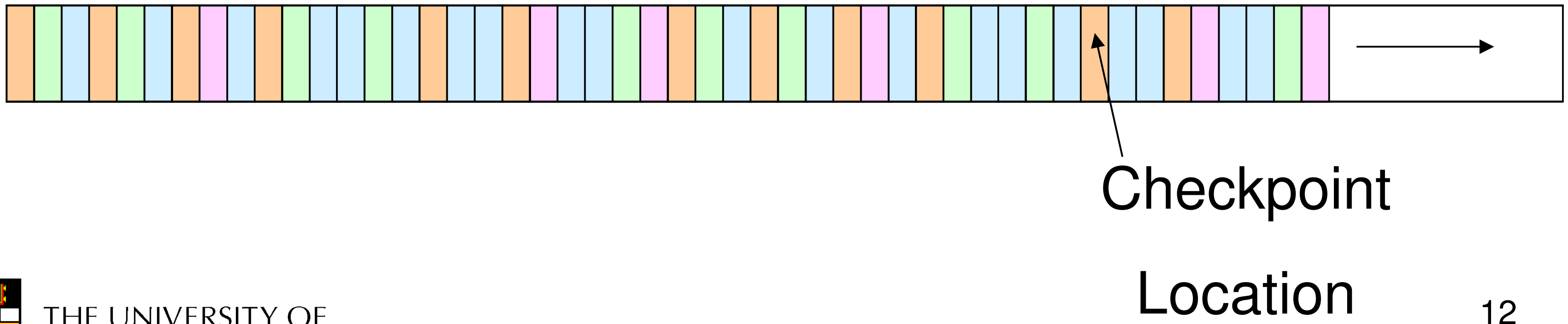


Issue
Disks are Finite in Size

- File system "cleaner" runs in background
  - Recovers blocks that are no longer in use by consulting current inode map
    - Identifies unreachable blocks
  - Compacts remaining blocks on disk to form contiguous segments for improved write performance

Issue
Recovery

- File system is check-pointed regularly which saves
  - A pointer to the current head of the log
  - The current Inode Map blocks
- On recovery, simply restart from previous checkpoint.
  - Can scan forward in log and recover any updates written after previous checkpoint
  - Write updates to log (no update in place), so previous checkpoint always consistent

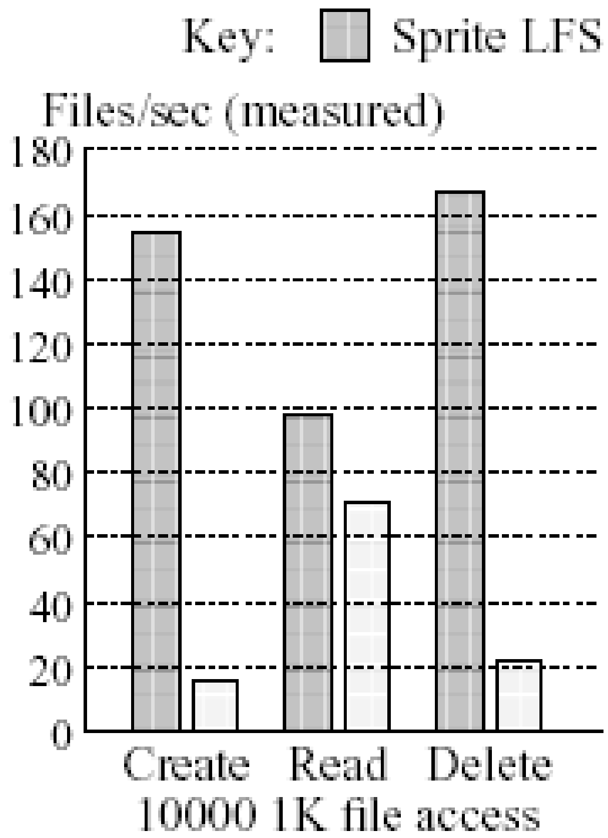


Reliability

- Updated data is written to the log, not in place.
- Reduces chance of corrupting existing data.
  - Old data in log always safe.
  - Crashes only affect recent data
    - As opposed to updating (and corrupting) the root directory.

Performance

- Comparison between LFS and SunOS FS
  - Create 10000 1K files
  - Read them (in order)
  - Delete them
- Order of magnitude improvement in performance for small writes

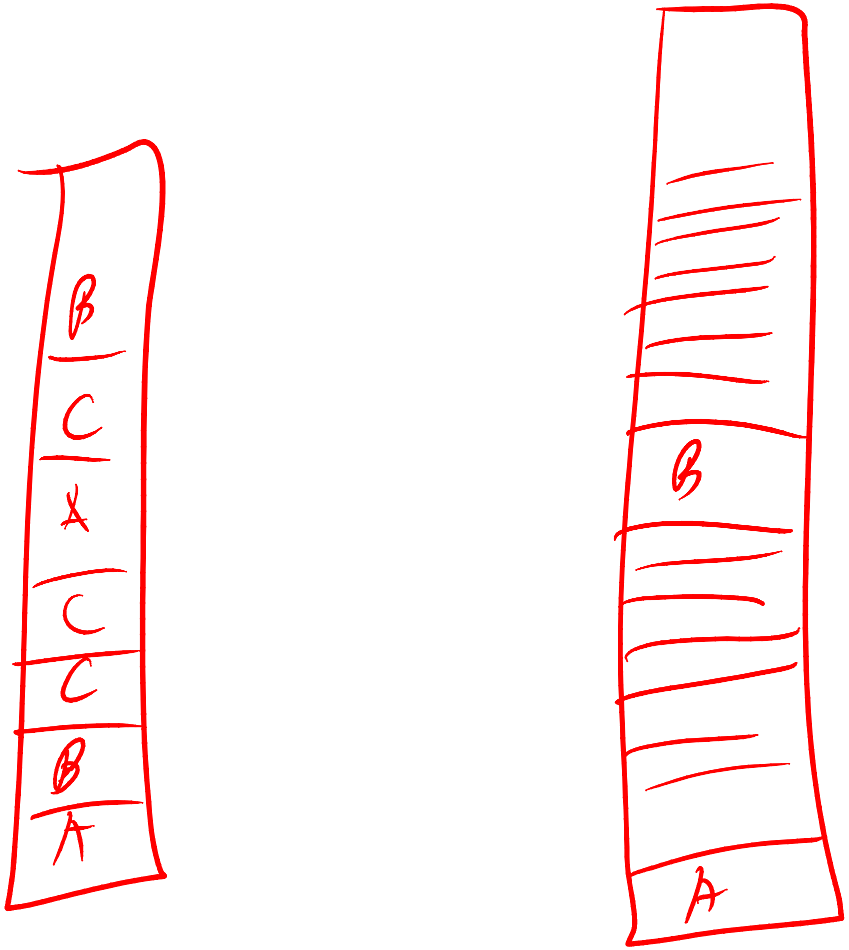


LFS a clear winner?

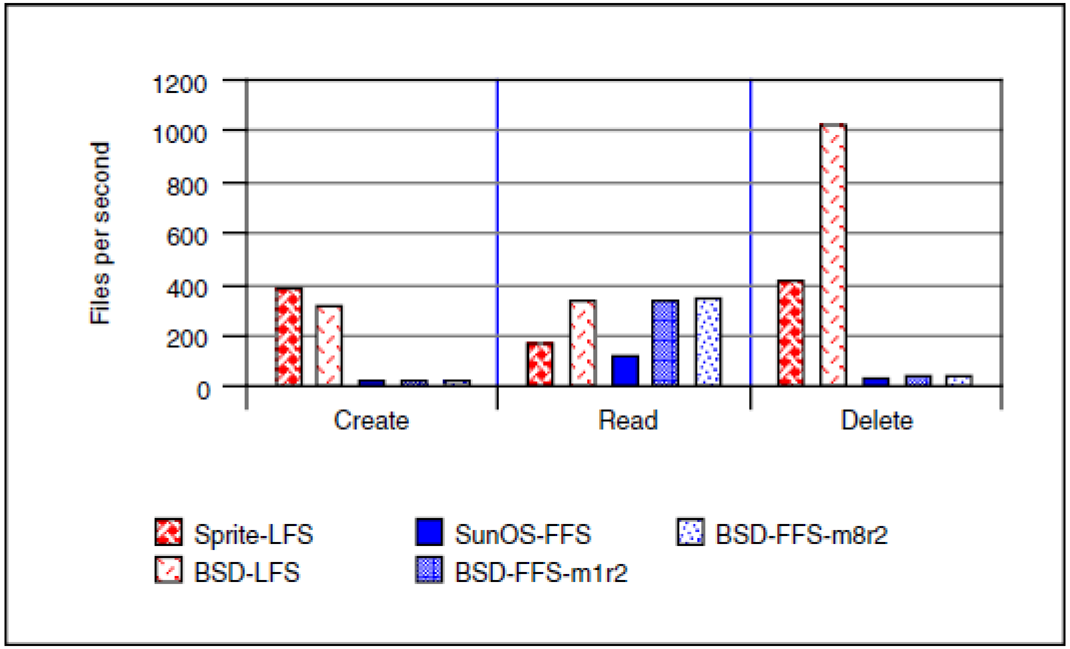
Margo Seltzer and Keith A. Smith and Hari Balakrishnan and Jacqueline Chang and
Sara Mcmains and Venkata Padmanabhan
"File System Logging Versus Clustering: A Performance Comparison"

- Authors involved in BSD-LFS
  - log structured file system for BSD 4.4
  - enable direct comparison with BSD-FFS
    - including recent clustering additions
- Importantly, a critical examination of cleaning overhead

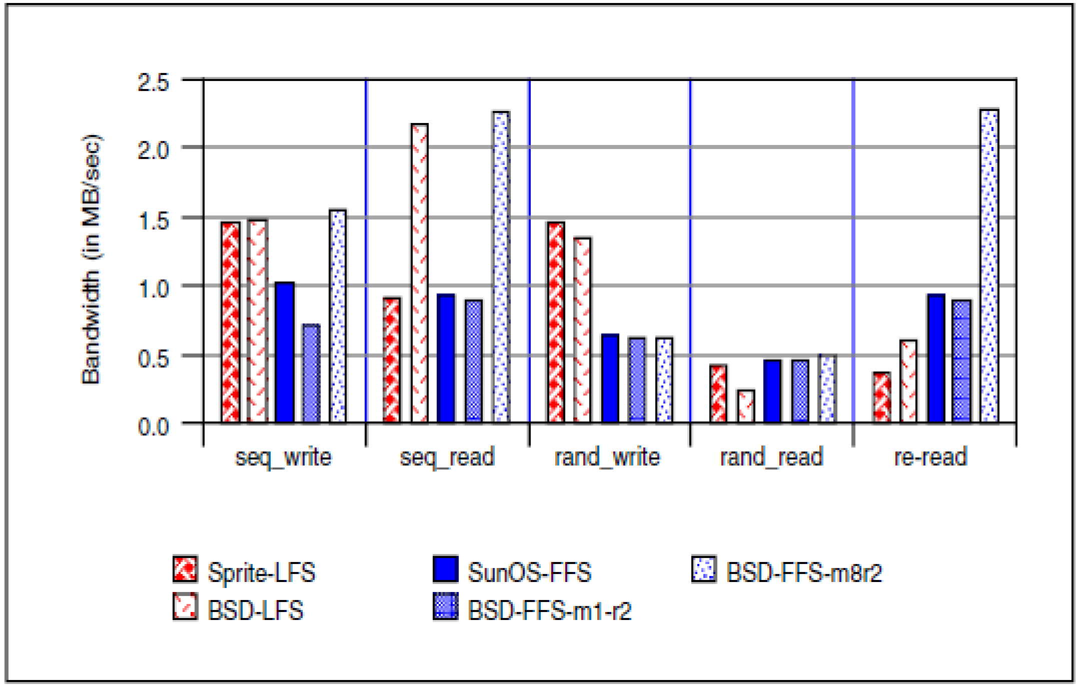
Clustering



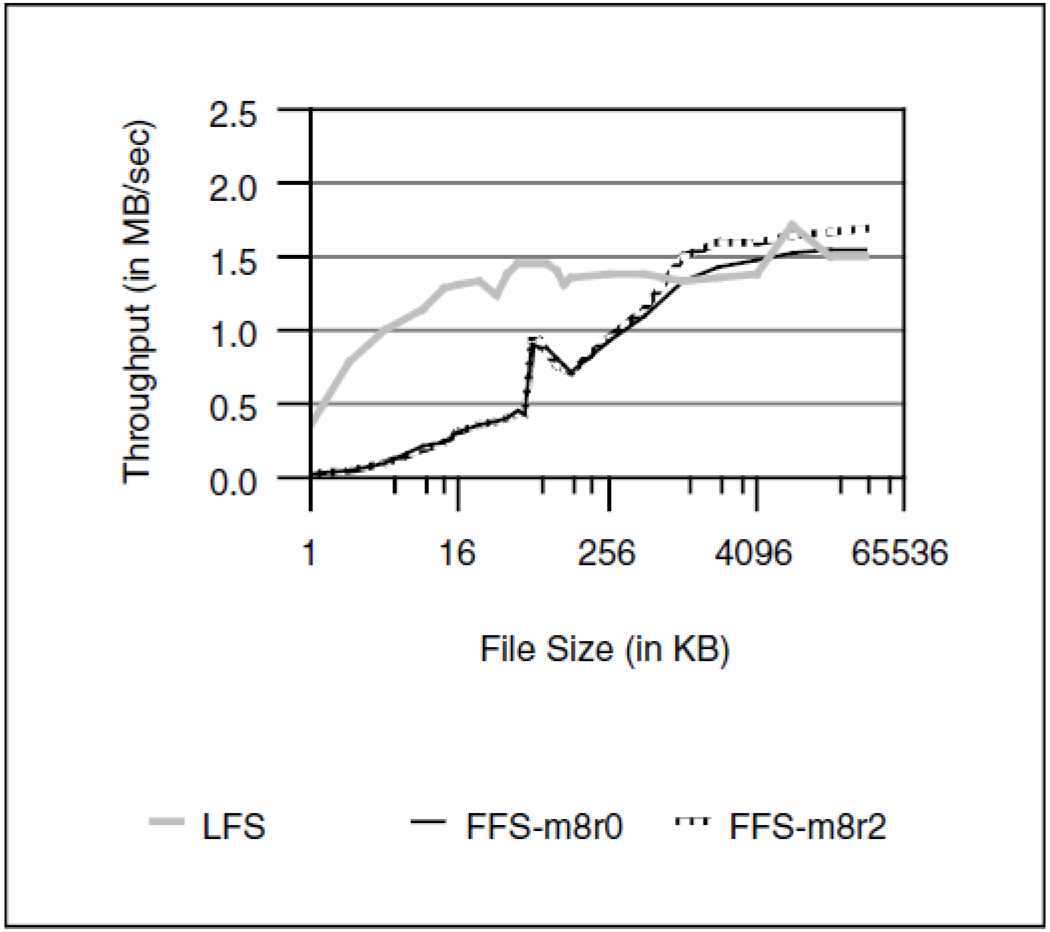
Original Sprite-LFS Benchmarks Small file

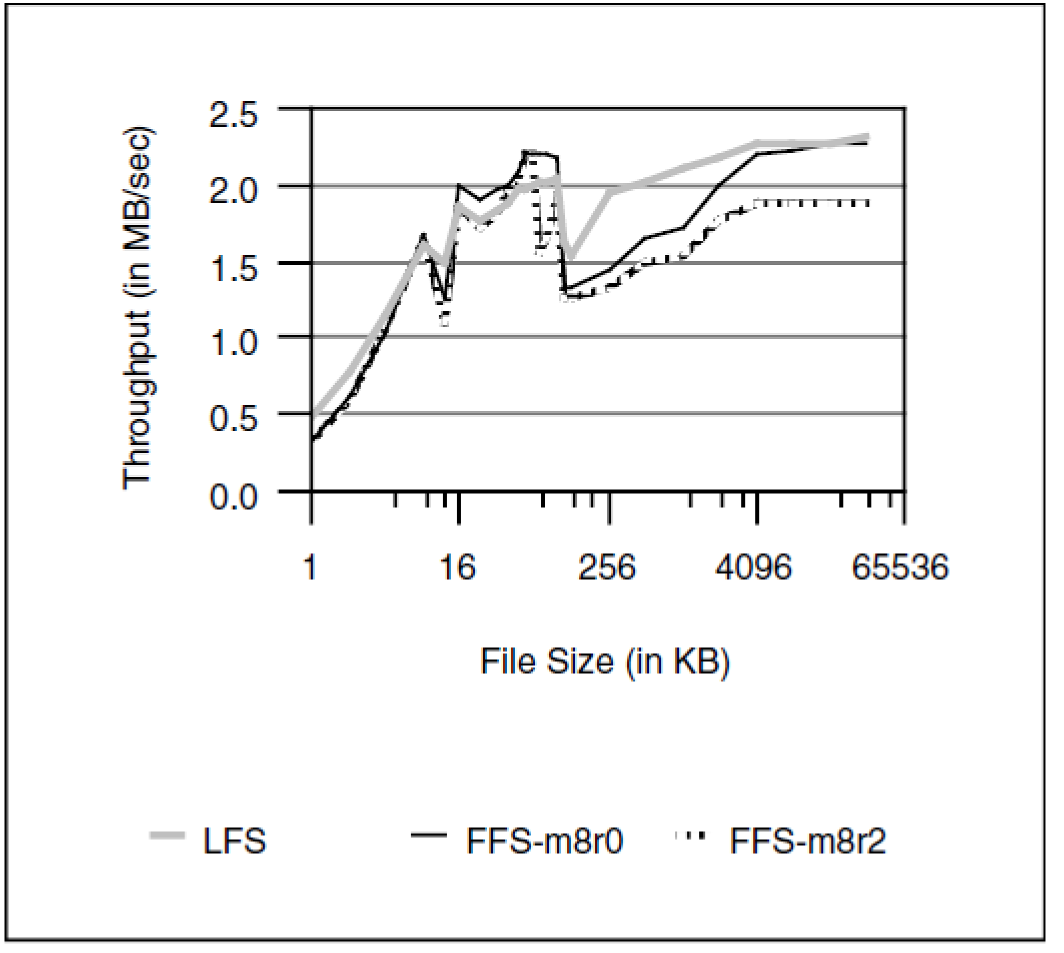


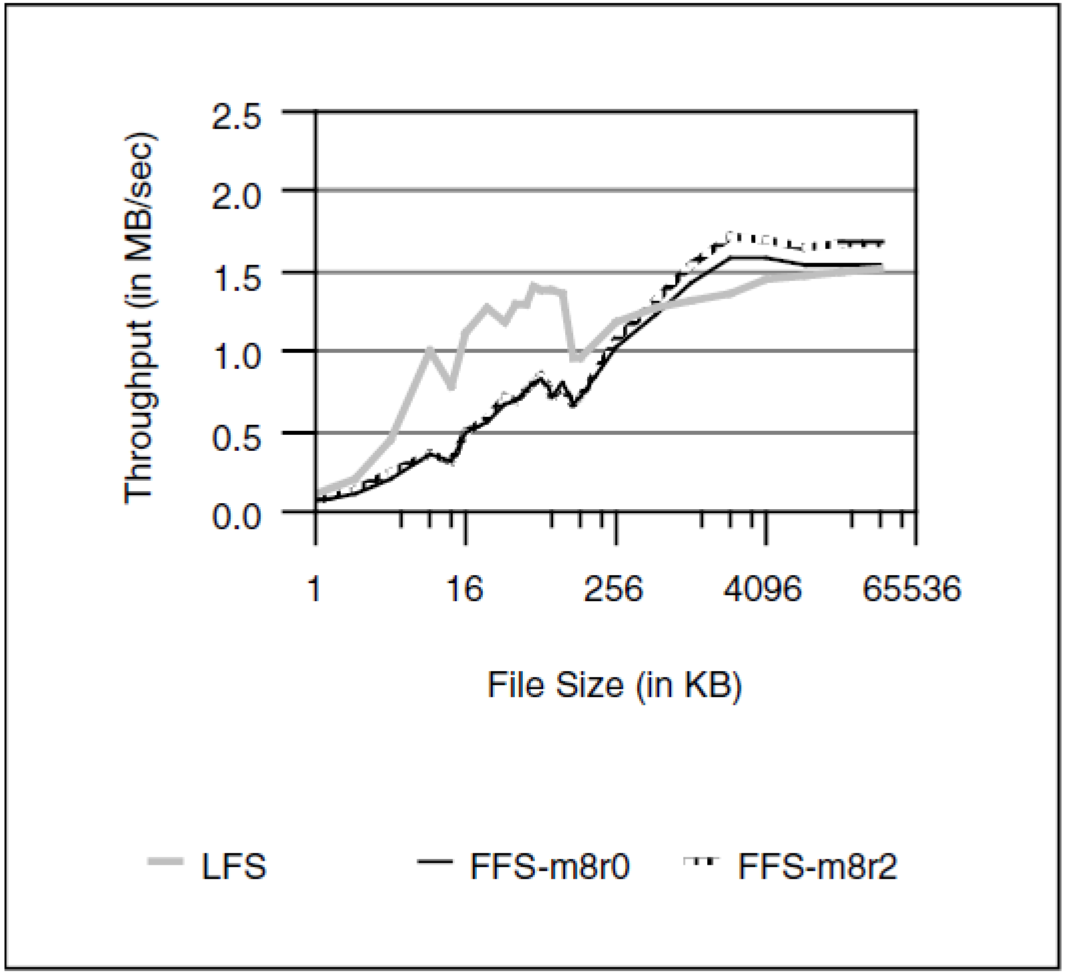
Large File Performance
100 Meg file

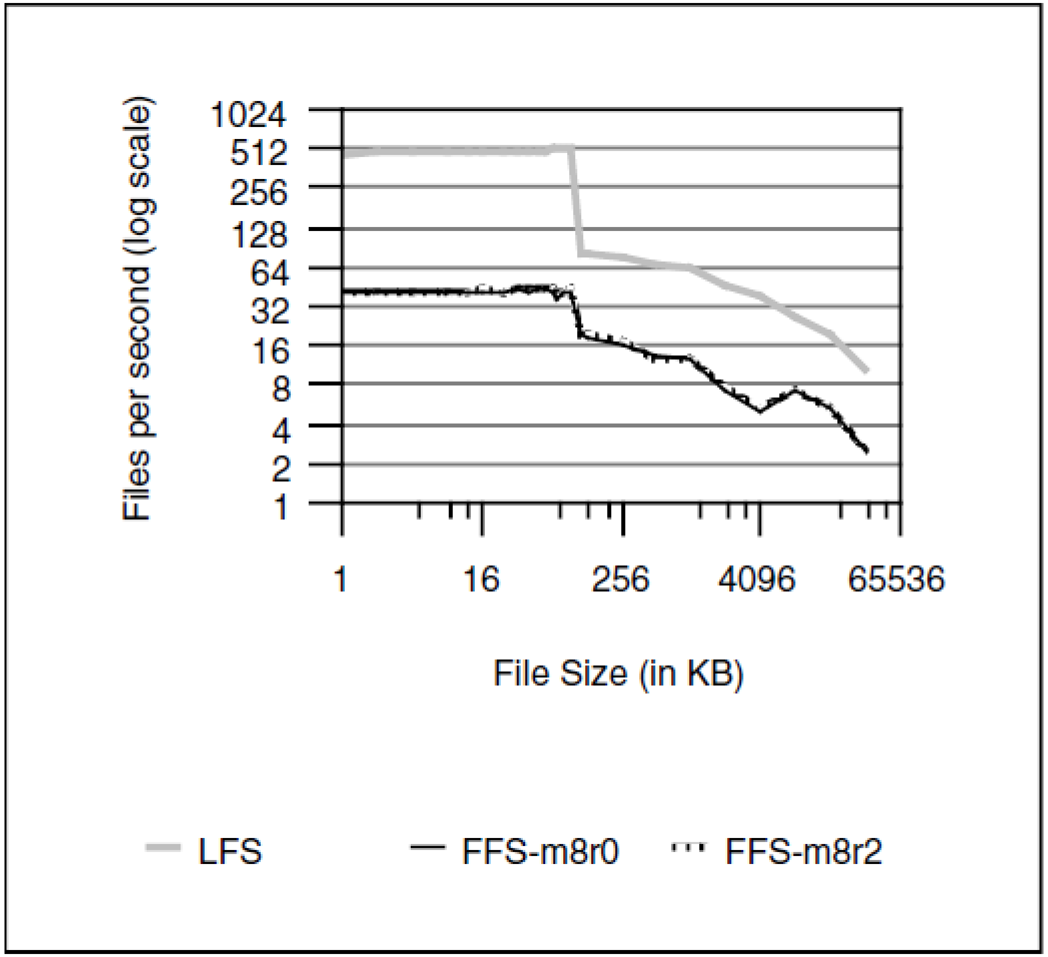


Create performance

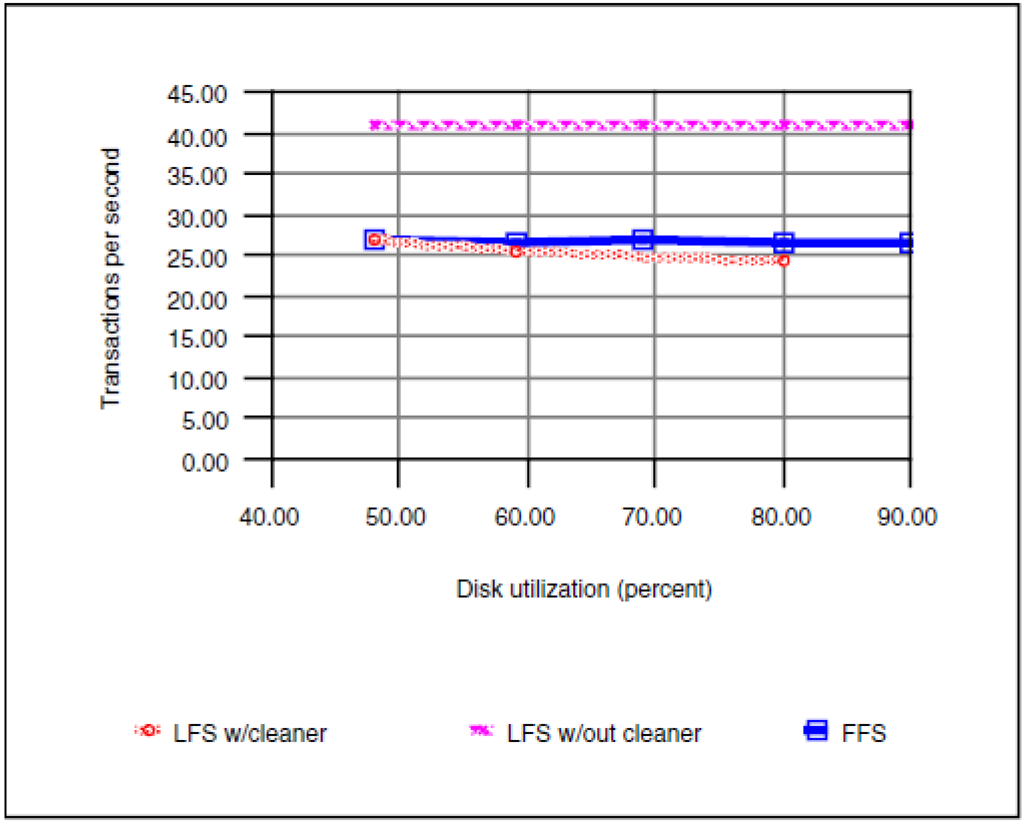



Throughput (in MB/sec)
2.5
2.0
1.5
1.0
0.5
0.0
1
16
256
4096
65536
File Size (in KB)
LFS
FFS-m8r0
FFS-m8r2


Throughput (in MB/sec)
2.5
2.0
1.5
1.0
0.5
0.0
1
16
256
4096
65536
File Size (in KB)
LFS
FFS-m8r0
FFS-m8r2



THE UNIVERSITY OF
NEW SOUTH WALES



THE UNIVERSITY OF
NEW SOUTH WALES

LFS not a clear winner

- When LFS cleaner overhead is ignored, and FFS runs on a new, unfragmented file system, each file system has regions of performance dominance.
  - LFS is an order of magnitude faster on small file creates and deletes.
  - The systems are comparable on creates of large files (one-half megabyte or more).
  - The systems are comparable on reads of files less than 64 kilobytes.
  - LFS read performance is superior between 64 kilobytes and four megabytes, after which FFS is comparable.
  - LFS write performance is superior for files of 256 kilobytes or less.
  - FFS write performance is superior for files larger than 256 kilobytes.
- Cleaning overhead can degrade LFS performance by more than 34% in a transaction processing environment. Fragmentation can degrade FFS performance, over a two to three year period, by at most 15% in most environments but by as much as 30% in file systems such as a news partition.

Journaling file systems

- Hybrid of
  - I-node based file system
  - Log structured file system (journal)
- Many variations
  - log only meta-data to journal (default)
  - log-all to journal
- Need to write-twice (i.e. copy from journal to i-node based files)
- Example – ext3
  - Main advantage is guaranteed meta-data consistency