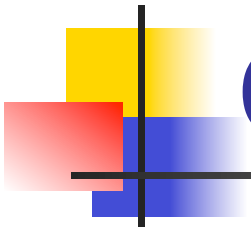
Deciding When to Forget in the Elephant File System

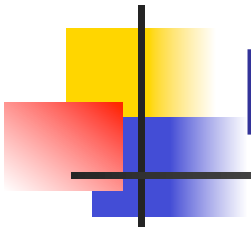
Douglas. S. Santry, Michael J. Feeley, Norman C. Hutchinson, Ross W. Carton, Jacob Ofir, and Alistair. C. Veitch (1999)

Presented by norm



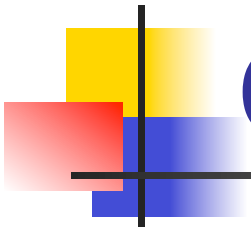
Outline

- Motivation
- Key Issues
- Design
- Implementation
- Performance
- Conclusion
- Discussion



Motivation

- **Need:** A File system that protects users from their mistakes
- Existing File Systems
  - Cedar – protection from accidental overwrite, but not delete
  - Trash can – protection from delete but not overwrite. Also provide limited undo capacity
  - Checkpoint – change between checkpoints not recoverable and limited number and frequency of checkpoints.

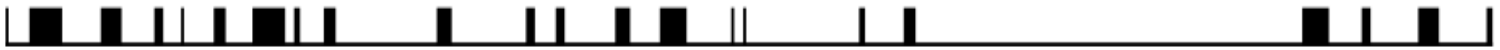


Other solutions

- Users make and maintain multiple copies of data and avoid deletes whenever possible
- File editors provide “undo” semantics

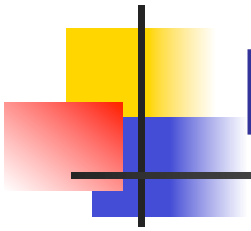
Histories

- Files are edited in bursts



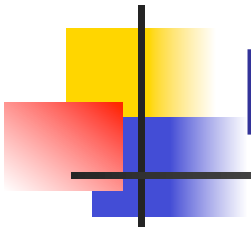
Elapsed time = 25.8 days

Elapsed time = 18.5 days



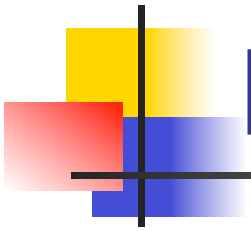
Key Issues

- **Storage reclamation** separated from **file writes** and **reads**
  - Deleting a file must not release its storage and file updates must not overwrite existing file data.
- Variety of retention policies
  - Many different types of files (Read-only, Derived, Cashed, Temporary, user-modified)
  - Specified by users, BUT implemented by the system
- Undo
  - Require complete history for limited period of time
- Long term histories
  - Don’t retain all versions
  - The file system assists the user in identifying important versions



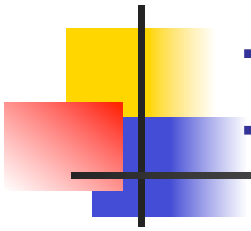
Design: Key Principle

- Separate *storage management* from the common file system *operations*
- Deleting a file does not release its storage
- Implemented using Copy-on-write
  - create a new version of a file block whenever it is written



Design: File retention policies

- Keep One – No versioning
- Keep All – Complete versioning
- Keep Safe – Undo protection
- Keep Landmarks – Long-term history



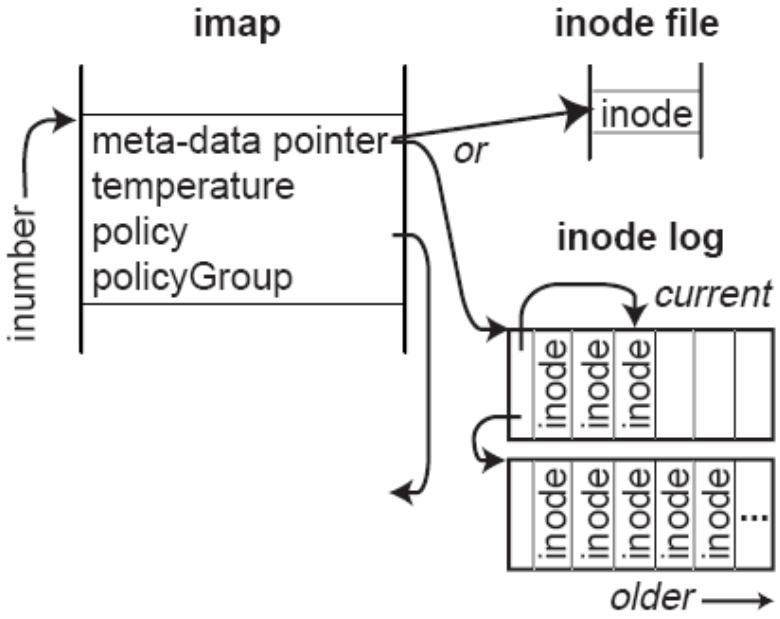
Implementation

- Defining Versions
  - Inode
  - Inode log
  - Name log
  - Non-versioned inodes

Implementation

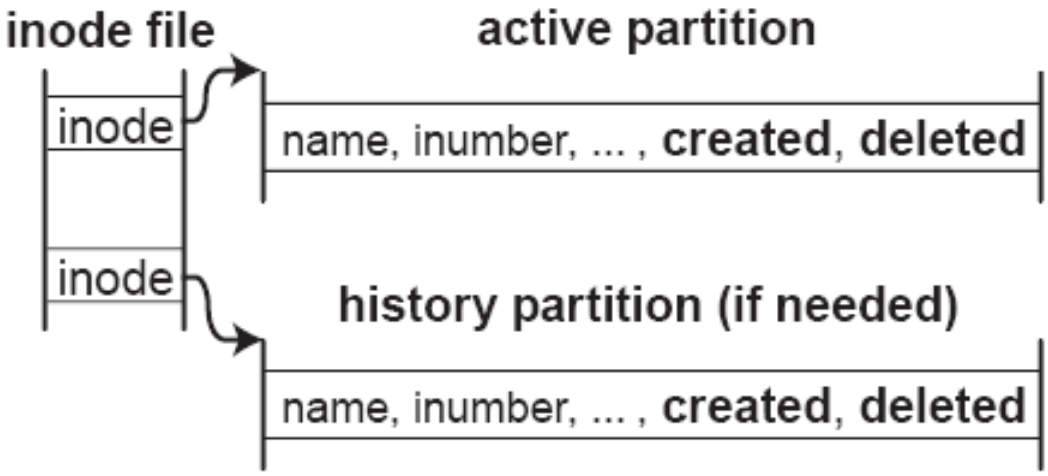
- Imap

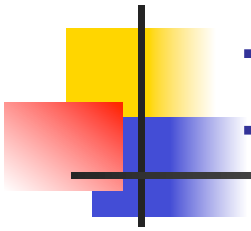
  provides a level of indirection between an inumber and the disk address of a file's inode or inode log.



Implementation

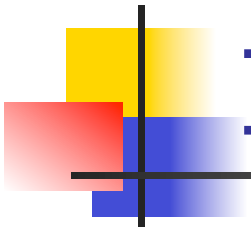
- Directories
  - Initially, one inode (the active inode)
  - After deletes, a second inode
  - Scan both inodes





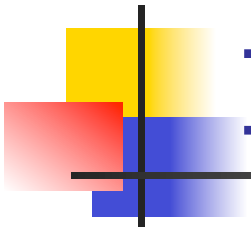
Implementation

- System Interface for User-Mode App
  - setCurrentEpoch (timestamp)
  - setHistory (file) => history
  - setLandmark (file)
  - unsetLandmark (file)
  - setPolicy (file, policyID)
  - groupFiles (fileA, FileB)
  - ungroupFile (file)



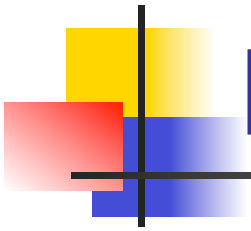
Implementation

- System Interface for Storage Reclamation
  - mapImap (pathN) => mntPt
  - lockFile (mntPt, iNum)
  - unlockFile (mntPt, iNum)
  - readBlock (mntPt, block)
  - writeBlock (mntPt, block)
  - freeBlock (mntPt, block)



Implementation

- System Interface for Application defined Retention Policies
  - registerPolicy (pathN) => policyID
  - unregisterPolicy (policyID)
  - lookupPolicy (policyID) => pathN
  - cleanFile (fileHistory) => (verList, newTemp)



Performance

- Compare the performance of the Elephant prototype to the standard FreeBSD Fast File System
- Examine the types of files stored by a large file system to estimate what portion of its files would be versioned
- Analyze file-system trace data to estimate how much extra storage an Elephant file system might consume for history information

Performance
Elephant vs. FFS

Operation	EFS-V (μs)	EFS-O (μs)	FFS (μs)
open (current)	134	133	132
open (20th ver)	140	—	—
open (75th ver)	160	—	—
write (4 KB)	58.7	54.5	47.3
close (upd)	35.8	34.5	34.9
close (upd 24th ver)	121	—	—
create (0 KB)	5040	3750	3930
delete (0 KB)	452	2154	3010
delete (64 KB)	446	4522	4732

Performance
File system profile - static

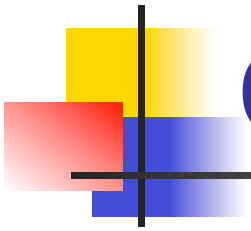
File Type	Files (%)	Bytes (%)
Source	14.6	3.4
Documents	22.6	11.0
Derived	20.6	53.3
Archive	3.9	28.5
Temporary	13.0	3.0
Other	25.2	0.8

- **Keep One**: 33.6% of files – 56.3% of bytes
- **Keep Safe**: 3.9% of files – 28.5% of bytes
- **Keep Landmarks**: 62.4% of files – 15.2% of bytes

Performance

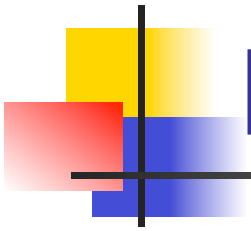
Extra Storage Consumed – dynamic

Policy	Files (%)	Bytes (%)	Writes (% Bytes)
Keep One	33.6	56.3	98.7
Keep Safe	3.9	28.5	0.6
Keep Landmarks	62.4	15.2	0.7



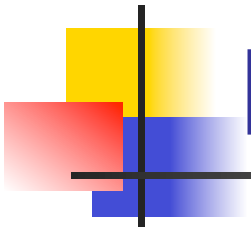
Conclusion

- Providing protection from user mistakes requires the separation of file system modification operations and file system storage reclamation.
- Four storage reclamation policies that are valuable to users
- Both system defined policies and application-defined policies can be implemented using a simple interface for file versioning



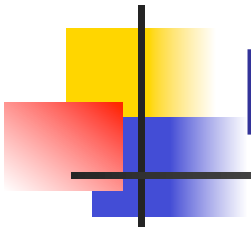
Discussion – name

- Why is the file system called Elephant?



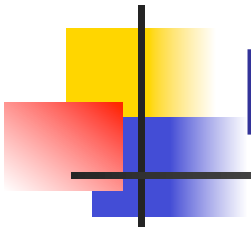
Discussion – details

- What are some examples of derived files and how can they be generated from the original sources?



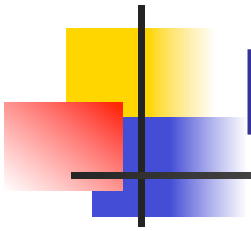
Discussion – details

- When a block is written a new inode will be created and updated and a new block allocated. For appending, copy-on-write is avoided. Doesn't appending to a file create a new version which must be reflected in a new inode?



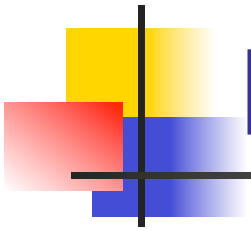
Discussion – details

- Does Elephant support the fragments that were introduced in FFS?



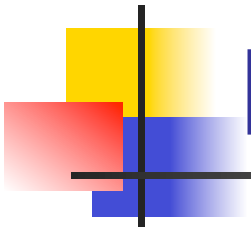
Discussion – details

- Can applications set conflicting policies for the same files?



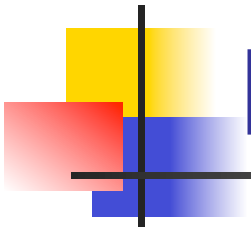
Discussion – details

- Isn’t fragmentation a bit problem? Since only changed blocks are copied, won’t a file that is continually being modified become hopelessly fragmented over time?



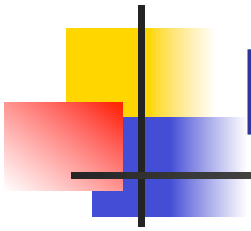
Discussion – details

- How does file name lookup for files with the same name and path but at different times? How are both the name histories in a directory and the file versions in inode logs managed when looking for an old version of a file?



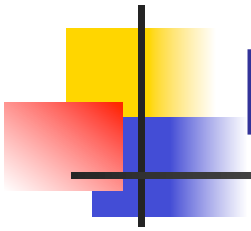
Discussion – details

- When a directory is rolled back (say to recover a deleted file), how does that affect files that have been created in that directory in the mean time, are they also rolled back?



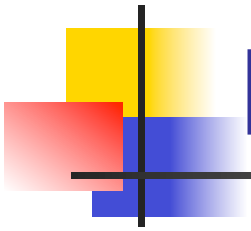
Discussion – details

- What if one user undoes a file back to an old version at the same time another user is editing the most recent version of the file? How does the system manage this conflict?



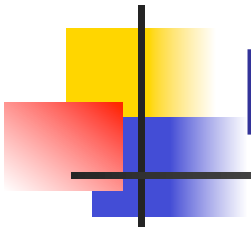
Discussion – policy parameters

- How should the proper second-chance interval for the keep safe policy be determined?
- Ditto the proper aging policy for identifying landmarks in the keep landmarks policy?



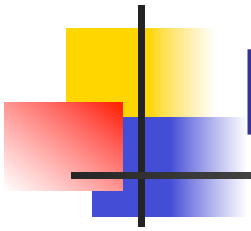
Discussion – security

- Will keeping versions and history information make the system more vulnerable to attacks?



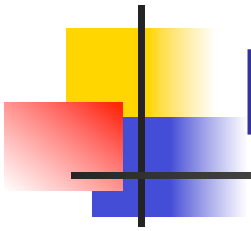
Discussion – usability

- Do you think users would prefer a general versioning scheme like Elephant, or specialized schemes for different kinds of files (SVN, GoogleDocs, etc.)?



Discussion – usability

- Is giving the user a command to set the policy for a file sufficient? Or will users find it tedious and end up having the wrong policy for files?



Discussion – impact

- What is the current status of versioning file systems? Does versioning appear in commodity file systems today?