Paper Reading Group

Secure File System Versioning at the Block Level

Jake Wires and Michael J. Feeley, 2007

Carsten Weinhold

<weinhold@os.inf.tu-dresden.de>

Motivation

“In typical file systems, valuable data is vulnerable to being accidentally or maliciously deleted or overwritten.”

Strategies to Protect Data

Two classes of data loss:

- Physical loss or malfunction of storage device
- Removal or modification after initial storage

Strategies to prevent data loss:

- Make each write redundant
- Create periodic snapshots
- Maintain a version history of all changes

Versioning

How to implement?

- Difficult to add to existing file systems
  - OS vendors don't want to
  - Increases complexity
- If part of file system, it is as vulnerable to bugs and attacks as rest of the system

VDisk approach:

- Isolate core functionality: VDisk secure kernel running in its own virtual machine
- Implement more complex functionality and recovery in untrusted user-mode tools

VDisk Architecture

Secure kernel:

- Provides writable block device for file system
- Logs all changes to protected block device
- Exports read-only block-write history
- Processes log-cleaning requests according to version retention policy

Untrusted user-space tools:

- Interpret and extract specific versions of files
- Decide which versions to remove from log
- Create proof-bearing cleaning requests

Version Logging

- Log partition sub-divided into segments
- Segments contain entries of:
  - *Data log*
  - *Metadata log*
- Each metadata entry contains
  - Physical sector number
  - Location in data log
  - Timestamp
  - Deleted
  - ...

Accessing Versioned Data

User-mode tools retrieve versions:

- Work on read-only logs
- Reconstruct file-system semantics
- Retrieve versions of the file system / specific files
- Can use arbitrary off-the-shelf tools as needed

MySQL-based prototype can retrieve:

- Specific version of a file
- Version history of a specific file

Version Pruning

Log cannot grow indefinitely!

- ➔ Versions need be coalesced

How to preserve data durability?

- ◆ Full control for user cannot be allowed
- ➔ VDisk secure kernel enforces declarative retention policy

Deleting Versions Securely

VDisk cleaner is split:

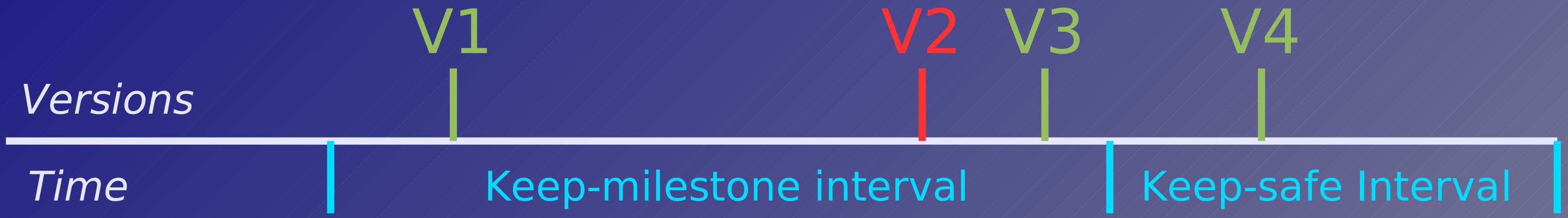
- Untrusted user-mode cleaner
  - Identifies versions to prune
  - Creates deletion-candidate and retention-proof lists
  - Identifies metadata log segments to be compacted
- Secure cleaner
  - Check provided proofs
  - Execute cleaning request if proof is valid

Retention Policies

- *Keep Safe:*
  - Keep all versions within a certain time interval
- *Keep Landmarks:*
  - Extension of *Keep Safe*
  - After keep-safe period: coalesce short-lived versions created within certain intervals
- ***VDisk:*** *Keep Milestones:*
  - Approximation of *Keep Landmarks*
  - Parameterized by keep-safe interval and constant milestone interval

Retention Proofs / Cleaning

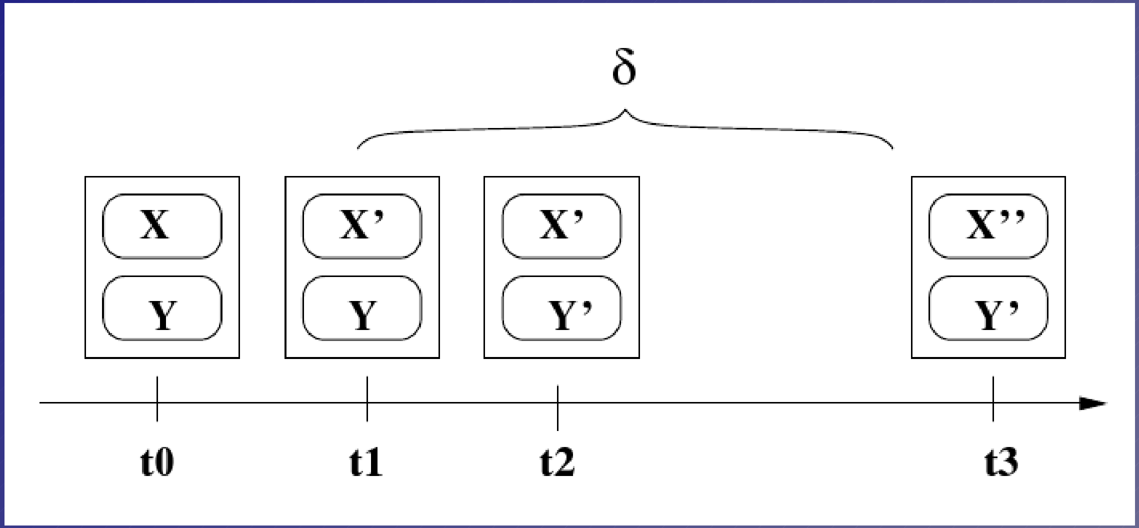
Proof consists of two versions:



After successful validation:

- Mark data blocks as obsolete in metadata log
- Move live blocks to new segment and free old segment

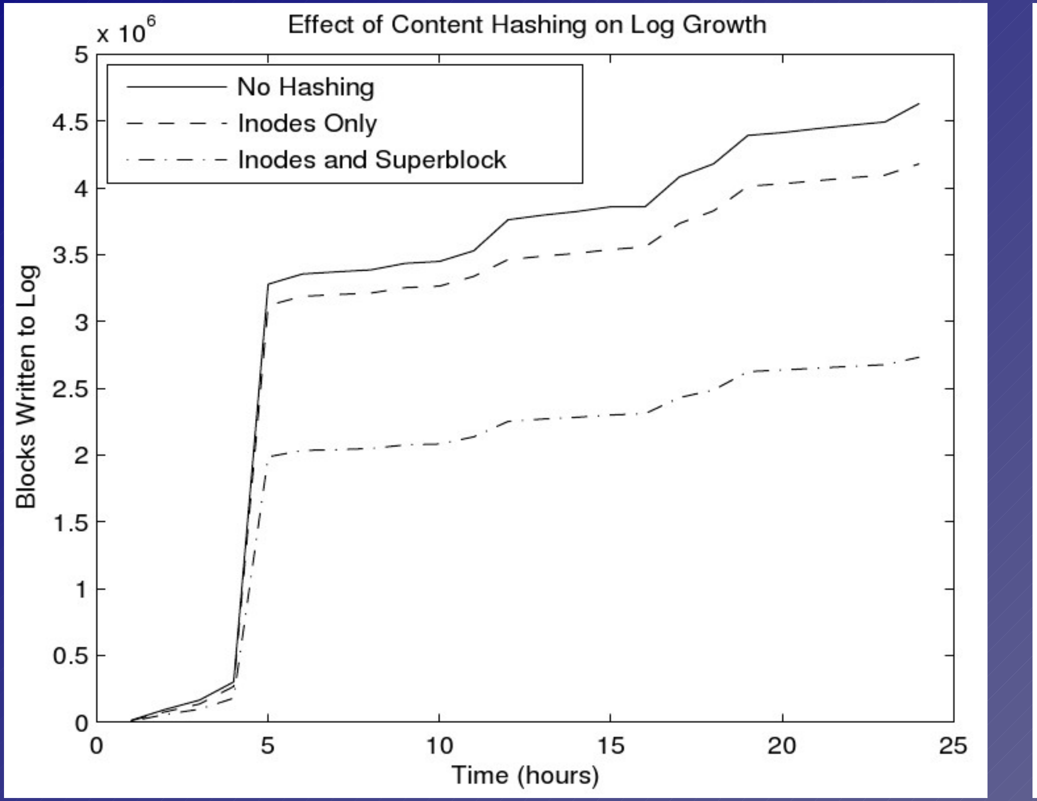
Milestone Constraint

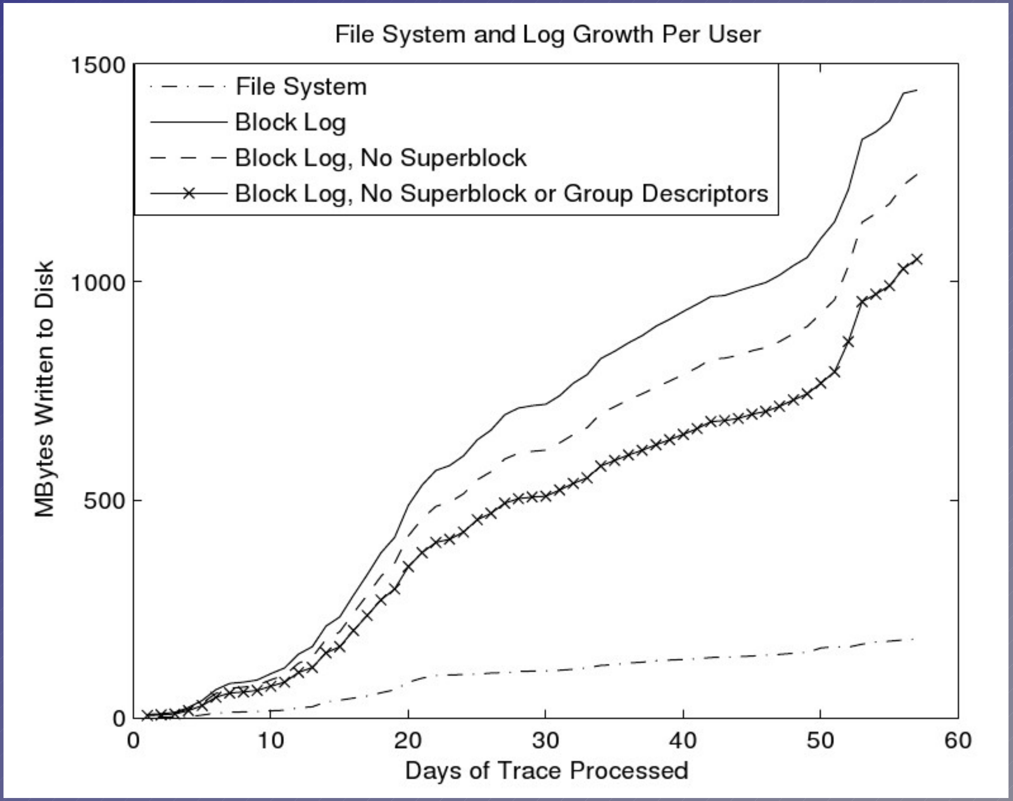


Additional Keep-Milestones check:

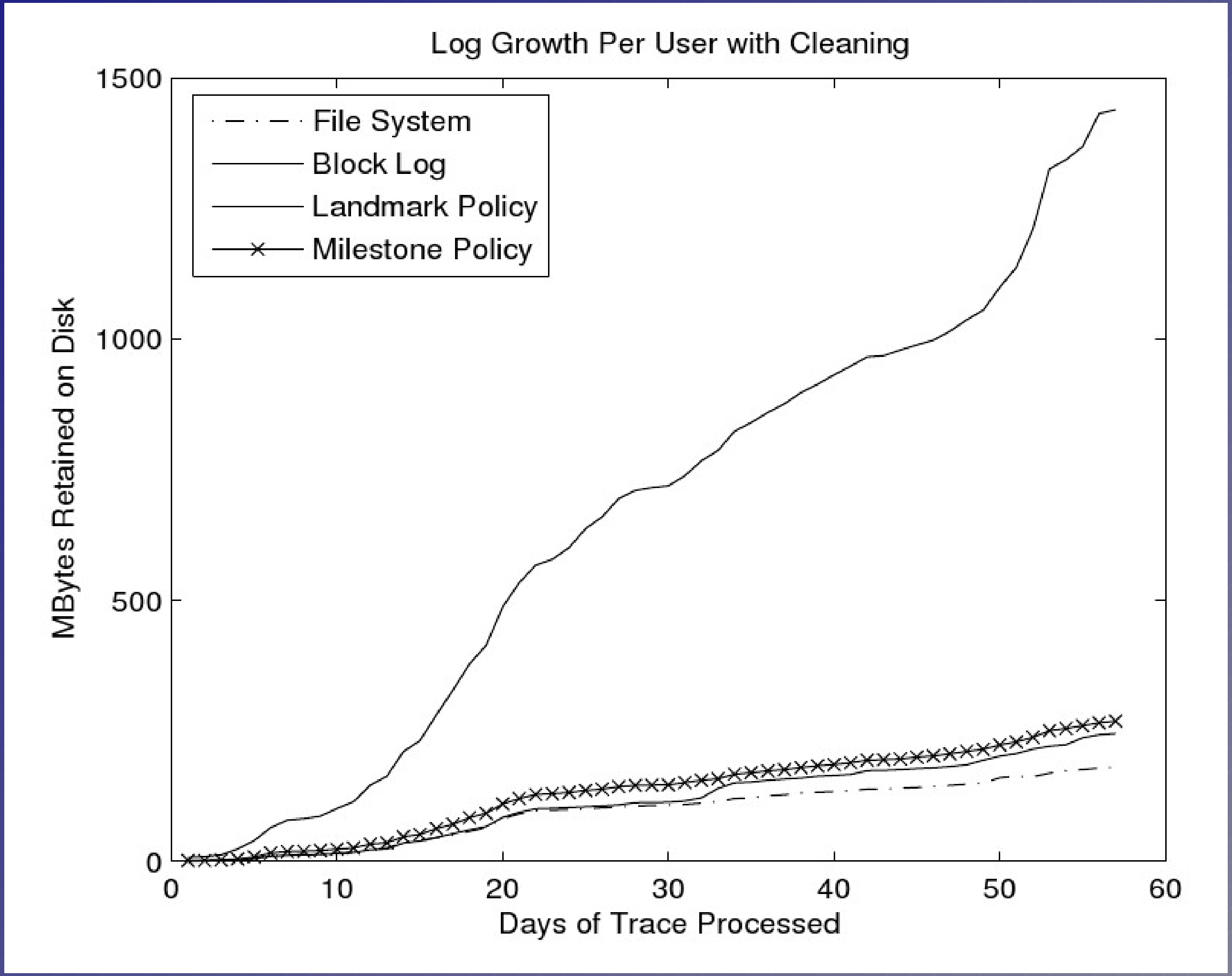
- As opposed to *Keep Landmarks*, only *t1* can be pruned

Evaluation: Optimizations

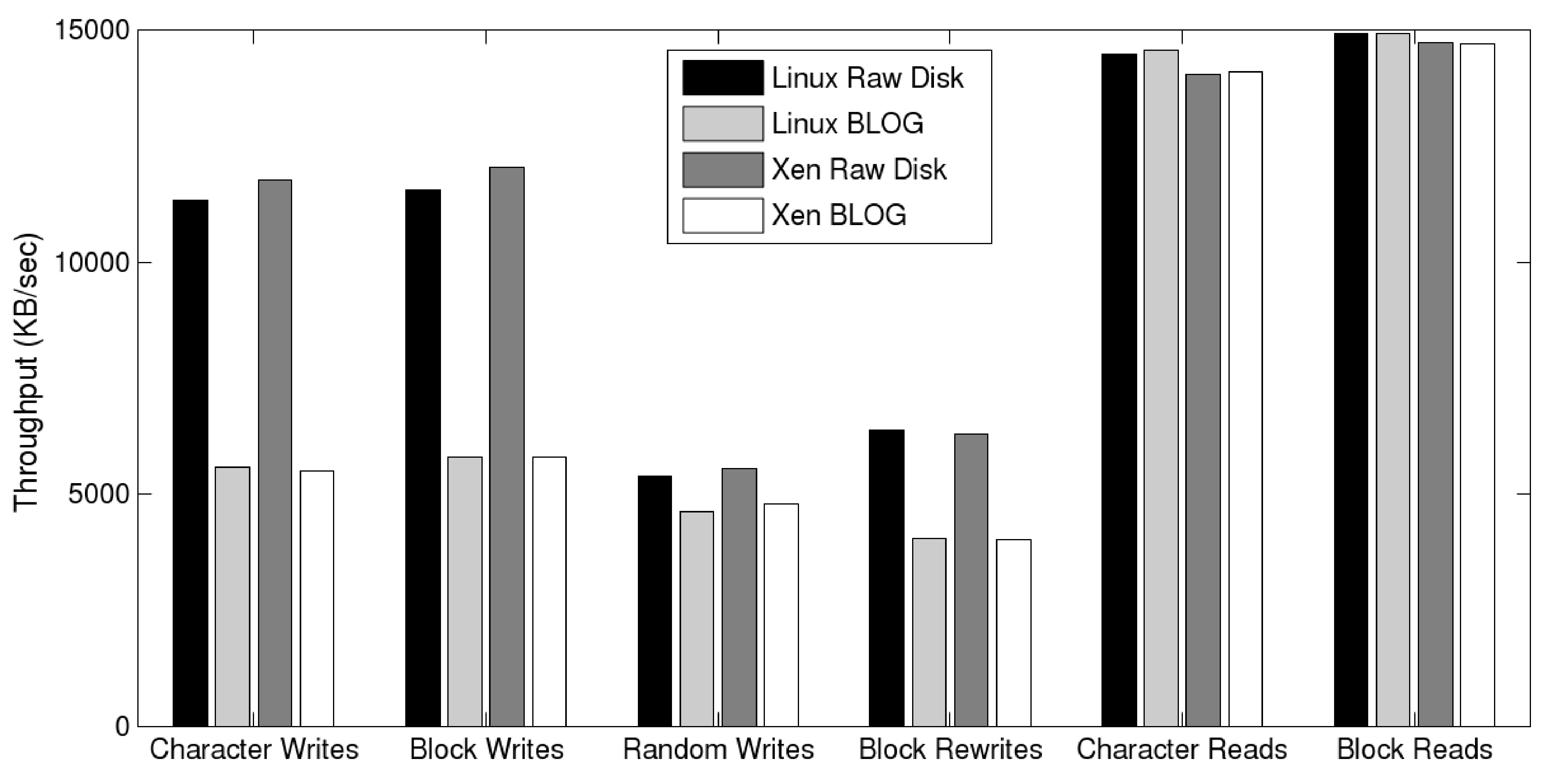




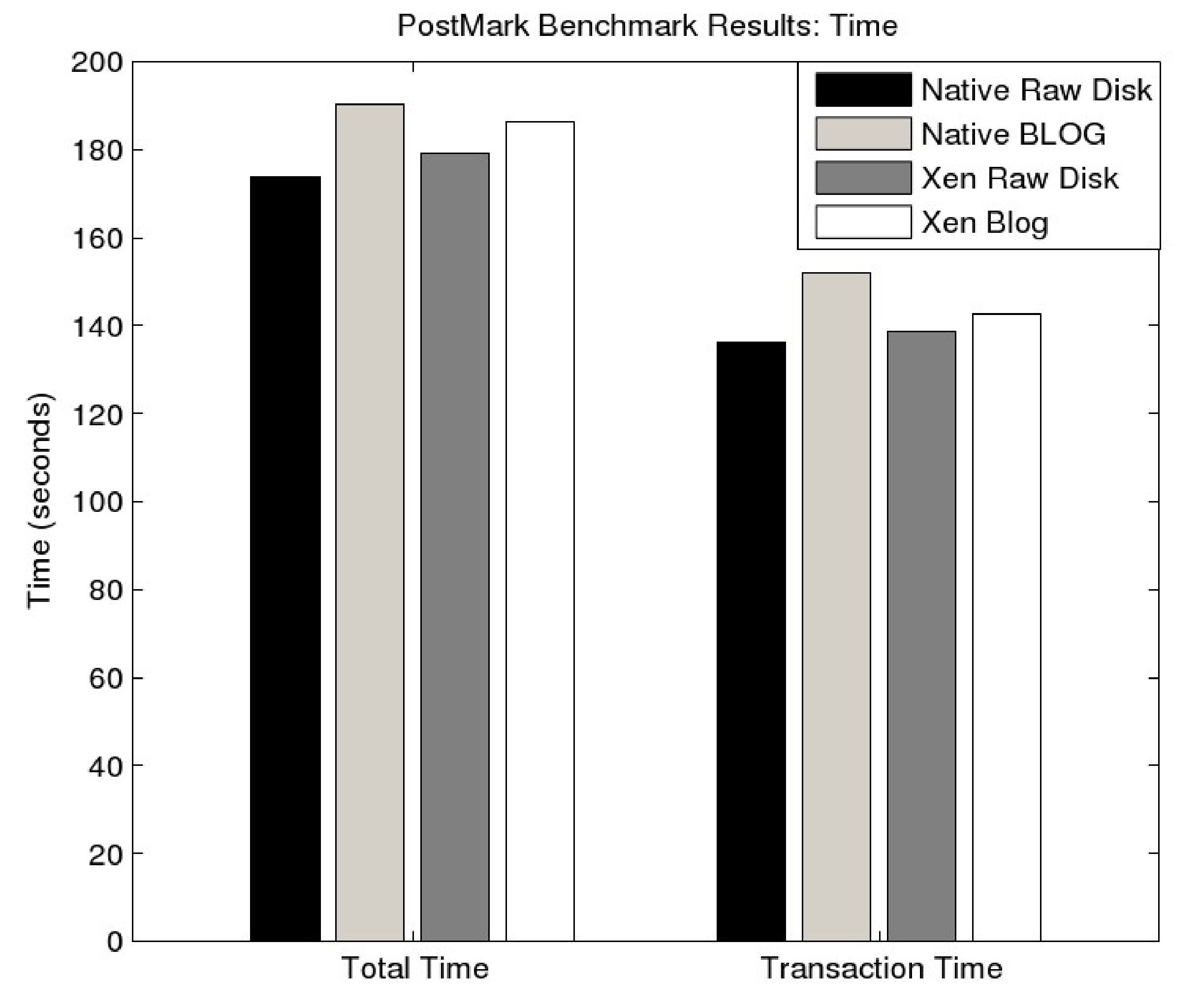
Evaluation: Log Growth



Evaluation: bonnie++



Evaluation: PostMark



Points of Discussion

- Optimizations usable for file systems other than ext2/ext3 (dynamically allocated inodes, ...)?
- Is lack of write ordering in ext2 a real problem?
- Your questions?

Retrieving a File Version

- Retrieval based on filename and timestamp
- Straightforward approach:
  - Retrieve superblock
  - Retrieve all directories specified in pathname
  - Last element is requested file / directory
  - File / directory contents found using metadata (inodes, ...)
- Implemented using SQL requests

Retrieving a File History

- Similar to retrieval of single file, but:
  - All versions of all path elements are examined
  - Inode blocks are scanned for inodes with modification time in requested interval

Logging Optimizations

- Avoid redundant writes
  - Hash table with information recently read sectors and their contents
  - Don't write if contents didn't change
- Log certain sectors only once
  - Don't write copies of ext2/3 superblocks and group descriptors