



File System and Storage Integrity

Kiron Vijayasankar

<http://www.fsl.cs.sunysb.edu/~kvijayas/cse590papers.html>



Overview

- Data Integrity
- Ensuring Data Integrity in Storage
- IRON File Systems
- Fast and Secure Distributed Read-Only File System
- Venti
- Tripwire
- I³FS
- Conclusion



What is Data Integrity?

http://en.wikipedia.org/wiki/Data_integrity

- The condition in which data is identically maintained during any operation, such as transfer, storage, and retrieval.
- The preservation of data for the intended use.
- Ensuring that data is "whole" or complete



Data Integrity in Storage

- Data corruption : A serious problem
 - Critical applications depend on stored data
- Applications have varying requirements
 - Need: basic guarantees for data integrity



Ensuring Data Integrity in Storage: Techniques and Applications

Gopalan Sivathanu, Charles P. Wright, and Erez Zadok

StorageSS'05



Causes of Integrity Violations

- Hardware and Software Errors
 - Device Drivers
 - File Systems
 - Networks
- Malicious Intrusions
- Inadvertent User Errors



Common Integrity Techniques

- Data Redundancy
- Mirroring
 - RAID-1
- RAID Parity
 - RAID-3/4/5
- Checksumming
 - Cryptographic hash functions



Scope of Integrity Assurance

- Avoidance
 - Read-only Storage
 - Journaling
 - Cryptographic File Systems
 - Transactional File Systems
- Detection
 - Checksumming
 - Mirroring
 - CRC
 - Parity



Scope of Integrity Assurance (contd.)

- Correction
 - Majority Vote
 - RAID Parity
- Detection and Correction
 - ECC
 - FEC
 - RAID Level 2



Logical Layers

- Hardware Level
 - On-Disk Integrity Checks
 - ECC, Reed-Solomon codes
 - Semantically-Smart Disk Systems
 - Hardware RAID
- Device Driver Level
 - Network Attached Secure Disks(NASDs)
 - Based on cryptographic capabilities. External file manager generates private keys using MAC for clients. NASD can compute the same private key and compare.
 - Software RAIDs



Logical Layers (contd.)

- File System Level
 - On-Disk File Systems
 - *fsck*
 - Stackable File Systems
 - I³FS, NCryptfs
 - Hardware RAID
 - Distributed File Systems
 - Google File System, SFSRO
- Application Level
 - Tripwire
 - Samhain
 - Radmin
 - Osiris
- Online vs Offline Integrity Checks
 - I³FS, SFSRO vs Tripwire, fsck



Uses of Integrity Checks

- Security
 - Intrusion Detection
 - Non-Repudiation and Self-Certification
 - Trusting Untrusted Networks
- Performance
 - Duplicate Elimination
 - Venti
 - Indexing
 - SFSRO
 - Detecting Failures
 - Non fail-stop disk failures



Implementation Choices

- Granularity of Integrity Checking
 - Block Level
 - Network Request Level
 - Page Level
 - File Level
- Storing Redundant Information
 - Handles
 - Venti, SFSRO
 - Parallel Files
 - NCryptfs
 - In-kernel databases
 - I³FS
- Semantics of data can be used to verify integrity



IRON File Systems

Vijayan Prabhakaran, Lakshmi N. Bairavasundaram,
Nitin Agrawal, Haryadu S. Gunawi, Andrea C. Arpaci-
Dusseau, and Remzi H. Arpaci-Dusseau

SOSP'05



Introduction

- Fail-partial failure model for disks
- Disks exhibit
 - Latent sector faults
 - Blocks become silently corrupted
 - Transient performance problems
- Most file systems
 - Illogical inconsistency in failure policy
 - Incorrect implementation of failure policy
- How can file systems handle modern disk failures?
 - Don't trust the disk



Introduction (contd.)

- Why do disks fail?
 - Media
 - Mechanical
 - Electrical
 - Drive firmware
 - Transport
 - Bus controller
 - Low-level drivers
- The fail-partial failure model
 - Entire disk failure
 - Block failure
 - Block corruption



The IRON Taxonomy

- Levels of detection
 - Zero
 - ErrorCode
 - Sanity
 - Redundancy
- Levels of Recovery
 - Zero
 - Propagate
 - Stop
 - Guess
 - Retry
 - Repair
 - Remap
 - Redundancy



Failure Policy: Results

- Linux ext3
 - Overall simplicity
- ReiserFS
 - First, do no harm
- JFS
 - The kitchen sink
- NTFS
 - Persistence is a virtue



An IRON File System

- Checksumming
- Metadata replication
- Parity
- Transactional checksums
- Cleaning overheads



Fast and Secure Distributed Read-Only File System

Kevin Fu, M. Frans Kaashoek, and David Mazieres

*ACM Transactions on Computer Systems, Vol. 20,
No. 1, February 2002*



Introduction

- Distributing public, read-only data securely
 - Replication generally comes at the cost of security
- Why a file system
 - File namespace can be referred from any context – shell scripts to C code to web browser
- Uses the naming scheme of SFS



SFS read-only file system

- Consists of
 - Database generator (*sfsrodb*)
 - Server (*sfsrosd*)
 - Client (*sfsrocd*)
- Content can be replicated to untrusted machines
- Uses Rabin public key cryptosystem
 - Fast signature verification
- SHA-1 used to compute cryptographic hash of content



SFS Read-Only Protocol RPCs

- *getfsinfo*

- Takes no argument
- Returns digitally signed FSINFO structure
- Client verifies signature using public key embedded in the server's name

- *getdata*

- Takes a 20-byte cryptographic hash
- Returns a data block which is identified by the cryptographic hash



SFS Read-Only Data Structures

- Read-Only Inode
 - Usual metadata minus permissions
 - Permissions can be synthesized on the client
- Directories
 - An inode of type directory or opaque directory
 - Consists of (*name*, *handle*) pairs
 - Entries are sorted lexicographically by name
 - Also contains its full pathname from the root of the filesystem.



File System Updates

- How to deal with data that no longer exists in the file system?
 - Since servers are not trusted, clients cannot necessarily believe “*handle not found*” errors
 - Using *fhdb*: *fhdb* is the root of a hash tree, the leaf nodes of which contain a sorted list of every handle in the file system
 - Pathname-based approach: Client tracks the pathnames of all files accessed. It chooses NFS file handles that are bound to pathnames rather than to hashes of the read-only inodes.
 - Incremental Update and Transfer: *sfsrodb* could update the database incrementally after changes are made to the file system, recomputing only hashes from changed files up to the root handle.



Applications

- Certificate Authorities
 - SFS uses file systems to certify public keys of servers. They are file systems serving symbolic links that translate human-readable names into public keys that name file servers
 - Certificate revocation is just removing a symbolic link.
- Software Distribution
 - Doesn't need SSH or secure HTTP
 - Users can browse the the distribution as a regular file system.
 - Provides revocation support.
 - Can sign a collection of RPMs that constitute a single system



Venti: a new approach to archival storage

Sean Quinlan and Sean Dorward

FAST'02



Introduction

- A network storage system that uses a unique hash of a block's content as the block identifier
- An archival storage system imposes a *write-once* policy
- No periodic "cleaning up"
- Venti provides a write-once archival repository that can be shared by multiple client machines and applications



Introduction (contd.)

- Tape backups are error prone and tedious
- Full backups is difficult to generate but easy to restore
- Incremental backups are easy to generate but tough to restore
- Snapshots avoid the tradeoff. Snapshots and the active file system share any blocks that remain unmodified
- Device that stores the snapshots must efficiently support random access for reasonable performance



The Venti Archival Server

- Block-level network storage system intended for archival data
- Does not provide the services of file or backup system by itself, but rather the backend archival storage for these types of applications
- Data blocks accessed by their *fingerprint*. Hence a block cannot be modified without changing its address (*write-once*)
- Writes are idempotent. Multiple writes of the same data can be coalesced. Even duplicate data from different applications and machines can be eliminated if the clients write the data using the same block size and alignment.
- Since contents of a particular block are immutable, the problem of data coherency is greatly reduced; a cache or mirror cannot contain a stale version of a block.



The Venti Archival Server (contd.)

- Choice of hash function
 - SHA-1
- Choice of storage technology
 - Magnetic disks
- Applications
 - Can build complex data structures using the block level service provided by venti.
 - Hash tree using fingerprint of blocks



Applications

- vac and unvac
 - Archives data at the file or logical level
 - Similar to tar/zip and untar/unzip
 - Contents of selected files are stored as a tree of blocks on a Venti server
 - The root fingerprint for this tree is written to a vac archive file specified by the user, which consists of an ascii representation of the 20 byte root fingerprint plus a 25 byte header
 - For a user, it appears that vac compresses any amount of data to 45 bytes



Applications (contd.)

- Physical backup
 - Simplest form is to copy the raw contents of one or more disk drives to venti
 - The fingerprint needs to be recorded outside of Venti
- Plan 9 File system
 - Combined with a small amount of read/write storage, Venti can be used as the primary location for data rather than a place to store backups



The Design and Implementation of Tripwire: A File System Integrity Checker

Gene H. Kim and Eugene H. Spafford

ACM CCS (1994)

<http://sourceforge.net/projects/tripwire/>



Introduction

- An integrity checking tool designed for the UNIX environment to aid system administrators to monitor their file systems for unauthorized modifications.
- Simplest version: a database is created with some unique identifier for each file to be monitored. By recreating the identifier and comparing against the saved version, it is possible to determine if a file has been altered.



Problem Definition

- Detect and notify system administrators of changed, added, or deleted files in some meaningful and useful manner
- Administrative issues
 - Most sites have hundreds of heterogeneous networked machines
- Reporting issues
 - Meaningfully reporting changed files is difficult
- Database issues
 - The database used by the integrity checker should be protected from unauthorized modifications
- File signature issues
 - Change detection
 - Signature spoofing
- Performance and resource issues
- Other issues
 - Trusting auxiliary programs
 - Database should be human readable



Implementation of Tripwire

- Uses two inputs
 - A *configuration* describing file system objects to monitor
 - A *database* of previously-generated signatures
- Administrative model
 - Portability
 - Written in K&R C adhering to POSIX standards
 - Human readable database files
 - Network byte order used in signatures
 - Scalability
 - Core configuration files conditionally included in the configuration file for each machine
 - Non-encrypted databases so that runs can be completely automated
 - Configurability and flexibility
 - Machines may share a configuration file, but each generates its own database file



Implementation of Tripwire (contd.)

- Reporting model
 - *Selection-masks* specify file system attributes and signatures to monitor
 - File name, expected and actual values of monitored attributes are reported on mismatch
- Database model
 - Inviolability
 - Tamper-proof media
 - Secure server
 - Read-only remote file system
 - Semantics
 - Added/deleted/updated file
 - Added/deleted/updated entry
 - Interface
 - Command-line



Implementation of Tripwire (contd.)

- Signatures model
 - Included in tripwire
 - MD5
 - MD4
 - MD2
 - 4-pass Snefru
 - 128-bit HAVAL
 - SHA
 - CRC-32 and CRC-16
 - Can be customized to use additional signature routines
- Performance
 - Local policy dictates which signatures are compared and how often



Experiences with Tripwire

- Securing the database
 - People have modified Tripwire to support alternate channels for receiving the database and transmitting the report, adding layers for networking support, encryption and host authentication
- Concealing Tripwire operation
 - Art of obfuscation
 - “Underground” publication warning the need for special vigilance when attempting to crack system running Tripwire



Experiences with Tripwire (contd.)

- Tracking Tripwire configurations
 - One configuration file shared by all machines
- Simple configuration files
 - Workstations with minimal disk resources can have a configuration file containing only “/”
- Frequency of Tripwire runs
- Validating the integrity checking scheme
 - Have detected intruders using Tripwire
- Evaluating Tripwire portability
 - 28 UNIX platforms and more
- Frequency of file system changes
 - Addition of interactive update facility for Tripwire databases



I³FS: An In-Kernel Integrity Checker and Intrusion Detection File System

Swapnil Patil, Anand Kashyap, Gopalan Sivathanu, and
Erez Zadok

LISA 2004



Introduction

- Host-based system which performs integrity checking at the file system level
- On-access integrity checking file system that compares the checksums of files in real-time
- A stackable file system which can be mounted over any underlying file system



Design

- Uses MD5 for checksums
- Passes file system calls to the lower file system. Injects its integrity checking operations and based on return values to system calls, it affects the behavior that user applications see
- Good balance between security and performance



Design (contd.)

- Threat model
 - Malicious replacement of vital files
 - Unauthorized modification of data
 - Corruption of disk data due to hardware errors
- Policies
 - Goals: versatility and ease of use
 - Syntax similar to Tripwire



Design (contd.)

- I³FS Databases
 - Four different in-kernel databases
 - Stores databases in B+ tree format
 - *policydb*
 - *datadb*
 - *metadb*
 - *accessdb*
- Caching in I³FS
 - Policy
 - Results of previous integrity check
 - Cached in inode private data



Design (contd.)

- Securing I³FS components
 - Authentication mechanism to ensure that updates to the checksum are made by authorized personnel
 - Databases stored in encrypted form
 - Authentication
 - Hash of passphrase stored in *policydb* during first-time mount
 - `ioctl` provided to change modes of operation
- Actions upon failure
 - BLOCK
 - NO-BLOCK



Implementation

- Initialization and Setup
- Mount options
- Metadata integrity checking
 - *i3fs_permission*
- Data integrity checking
 - PER_PAGE
 - WHOLE_FILE
- Frequency of checks
- Updating policies
 - ioctls
- Updating checksums
- Inheriting policies



Conclusion

- Integrity is essential for a good storage system
- Techniques differ in properties and cost
- Different techniques for different purposes
- Performance vs Security tradeoff



Questions??



Thank you 😊
