End-to-end Data Integrity for NFS

Chuck Lever
Consulting Member of Technical Staff
Today’s Discussion

• What is end-to-end data integrity?

• T10 PI overview

• Adapting T10 PI for byte-stream files

• Provisional feature requirements

• Protocol considerations
End-to-end Data Integrity Protection

• Prevent the storage or use of corrupted data
  • “Protection Information” allows detection and/or correction of data corruption (e.g., CRC)
  • Application provides PI, which is stored with data on permanent storage
    • Storage stack generates PI if application does not provide
  • Data integrity can be verified at every node in I/O path during both writes and reads
T10 PI Overview

• Defined in T10 SBC-2 and enhanced in SBC-3
• Data integrity for block storage
• “Type 1” defines contents of eight bytes of PI for every logical block
  • 16-bit CRC
  • 16-bit application tag
  • 32-bit reference tag (low order 32-bits of LBA)
• This is an open standard: allows any node in I/O path to verify that data and PI match
Data Integrity eXtensions

- Proposed by Oracle, not a standard
- T10 PI protects path between O/S buffer and block storage
- DIX extends protection up to applications
- Data and PI specified in separate buffers
- A lower-overhead guard tag is used
- Still block-oriented
Protecting Byte-Stream Files

• What API do applications use to specify reads and writes with accompanying PI?

• How is integrity of memory mapped data protected?

• Can an advanced file system store protected and unprotected data in the same volume?

• How does an advanced file system treat replicated blocks (snapshots or de-duplication)?
Protecting Byte-Stream Files

Application API

- Applications form the PI and submit it with the data
  - Apps need to know which protection mechanism is in use

- Protected reads and writes are logical block-aligned
  - Apps need to know size of logical block

- PI can be specified via ioctl(), scatter/gather, or other separate system call

- Data integrity failure can be reported via new errno
  - Application knows to employ a special system call to retrieve extended information
Protecting Byte-Stream Files
Advanced File System considerations

• File systems may alter application-specified PI during I/O on complex device types (e.g., RAID)
  • But, all devices backing an FS use same protection type

• All files on a particular volume are either protected or unprotected
  • Applications may choose not to supply PI; file system can generate it appropriately

• File system may choose to protect blocks storing its metadata
Protecting Byte-Stream Files

NFS client considerations

- Clients need to know which protection mechanism is in use on each FSID on a server
  - Can then advertise this to local applications

- Clients can use integrity-protecting transports along with PI

- Need to protect against write-re-ordering due to network or server instability
Provisional Feature Requirements

• End-to-end
  • Must allow protection from application write to read
  • Must permit verification at all nodes in path
  • Like RPCSEC, MUST implement, but deployment optional
  • File system operation must appear the same whether or not application is using or is even aware of data integrity

• Based on existing data integrity standards
  • IETF has no purview over physical storage
  • Better adoption if we expose existing standards on wire
  • Open standard means every node can participate
Provisional Feature Requirements

• Allow co-existence with other mechanisms
  • Should not interfere with serial or concurrent use of other data integrity verification mechanisms
  • Extensible: we want to allow other types of data protection
  • Mechanism must not interfere with access to data that is not protected by an end-to-end data integrity mechanism

• Agnostic to access method
  • Should work with any layout type
  • Non-pNFS access should also work
  • Should allow local access on file server, if appropriate
Provisional Feature Requirements

• Agnostic to server file system
  • No mandate for how a server’s file system supports data integrity protection, only how it looks to NFS clients

• Protection for NFS metadata operations not mandatory
  • So far I have not considered metadata operation protection
  • Partially accomplished using an integrity-protecting transport

• Minimal performance impact
  • We know there will be some, let’s try to keep it minimal
Protecting NFS Files

Example Protection Envelopes

- **NFS server-only**
  - Server does not advertise data integrity capabilities
  - Or client does not utilize data integrity capabilities
  - Data integrity failures appear to client as I/O errors

- **NFS client-server**
  - Client uses data integrity capability when communicating with server
  - Client does not advertise capability to applications
  - Client can accesses extended failure data, but apps can’t

- **Application-client-server**
  - Application can use data integrity on some or all of its files
  - Application can extract extended integrity failure data
Protecting NFS Files

Detecting Protection Types

- Server advertises protection type in use for an FSID
  - Can introduce a GETATTR per-filesystem attribute
  - Protection type MUST NOT change during FSID’s lifetime
  - FSIDs can use different protection types
  - Not all FSIDs protected
    - Pseudo-root
    - FedFS domain root

- Client advertises FSIDs protection type to applications
  - Applications may not use data integrity protection
  - Clients can choose not to use it, or generate it themselves
Protecting NFS Files

Possibilities for Reading and Writing PI

• New operations included in same compound as READ_PLUS, WRITE, or INITIALIZE
  • Works like GETATTR

• New enumerators for NFS4_DATA_CONTENT

• New arguments to READ_PLUS, WRITE, INITIALIZE

• New pNFS layout types
Protecting NFS Files

Asynchronicity

• Disk I/O can fail after a WRITE(UNSTABLE) operation completes
  • Failure MUST be reported at COMMIT time
  • Client then retries failing WRITE(UNSTABLE) via a WRITE(FILE_SYNC) to gather extended information about the failure

• Disk I/O can occur well before client reads data, due to server-side pre-fetch
  • Failure MUST be reported when specific block is read, not before
Protecting NFS Files

Generating PI

• T10 type 1 protection data
  • Application tag: Arbitrary or blank
  • Guard tag: 16-bit CRC
  • Reference tag: Lowest 32-bits of LBA
  • Protection envelope: I/O controller to block device

• Possible NFS protection data
  • Application tag: Arbitrary or blank
  • Guard tag: IP checksum
  • Reference tag: Middle 32-bits of file offset
  • Protection envelope: Application to NFS server
Protecting NFS Files
Reporting and Interpreting Failures

• T10 Type 1 failure report
  • Which tag failed to verify correctly
  • LBA of failure
  • Reporting node in I/O path

• Possible NFS failure report
  • Which tag failed to verify correctly
  • File offset of failure
  • Reporting layer
  • May be virtualized for simplicity
Protecting NFS Files
Multi-server Considerations

• Each DS participating in a layout MUST use the same protection type

• Each replica of an FSID listed in fs-locations MUST use the same protection type

• Destination server MUST support the same protection type as Source server
  • And, an FSID after migration MUST use the same protection type it was using before
Forthcoming Personal Draft

• Propose an architecture for end-to-end byte-stream data integrity protection based on T10 PI

• Enumerate and justify high-level requirements for NFS data integrity

• Provide enough meat to allow prototype implementations
Next Steps

- Complete and publish requirements document
- Build a prototype or two
- Consider support for other types of data integrity protection
  - Lustre
  - Native ZFS checksums