Building a Better Data Object

Patrick R. Eaton
University of California, Berkeley
eaton@cs.berkeley.edu

January 13, 2003
Outline

- Background and Context
- Requirements
- Current Design
- Proposed Improvements
  - New B-tree algorithms
  - Improved verifiability protocols
- Conclusions
Background: Peer-to-Peer Design Philosophy

- Client-centered architecture
- “Traditional” peer-to-peer system
- Self-organizing networks of machines that communicate symmetrically
- Application functionality at client
- Infrastructure routes messages
- Example: Napster
Background: Peer-to-Peer Design Philosophy

- Infrastructure-assisted architecture
- Application functionality distributed throughout infrastructure
- Use powerful, well-connect, well-administered machines in the network
- Designers must export application vocabulary into infrastructure
- Example: OceanStore
Background: The Path of an Update

- Inner ring applies updates to data object
- Data object format exported to infrastructure
Requirements

• Self verifying

• Support client-side encryption for privacy

• Support partial file caching

• Provide efficient update dissemination

• Low overhead versioning (copy-on-write)

• Expressive data structure operations
  – random access read, append, replace, insert, truncate, scan
Data object named by AGUID

Versions identified by VGUID

New versions are appended to chain

Old versions are read-only

Btree stores object data

References to children by secure pointer

Btree can reference blocks from previous versions

Support copy on write

Metadata prepended to root of btree
Design Errors

- Btree not sufficiently flexible
  - did not support encrypted data
  - did not support efficient insertions

- Update dissemination overcomplicated

- Inefficient metadata storage

- Poor support for tentative data

- Poor support for version branches
• All offsets are relative to file

• One-byte insert modifies all following interior blocks
Inspiration for New Btree Algorithms

- Exodus Storage Manager
  - Object and File Management in the Exodus Extensible Database System, Carey et. al. (VLDB 1986)

- Supported objects composed of variable-sized blocks that changed over time

- Key feature: index in interior nodes records offsets relative to the block, not to the file
• Offsets are relative to the block

• One-byte insert modifies only blocks between new block and root
OceanStore Extensions

- **Problem:**
  - Cannot modify encrypted blocks outside of client
  - Limits flexibility of insert

- **Solution:**
  - Include offset, length fields
  - Allow blocks to be referenced multiple times
• Offsets are relative to the block

• One-byte insert modifies only blocks between new block and root

• Encrypted blocks are unmodified
Background: Dissemination Tree

- Per-object, self-organizing, application-level multicast tree
- Root at the inner ring
- Distributes update results to replicas
Background: Types of Uncommitted Data

- **“Tentative” Data**
  - updates are propagated epidemically
  - update applied optimistically on secondary replica
  - result checked after receiving result from inner ring
  - version rolled back if necessary

- **“Unnamed” Data**
  - data created at inner ring before GUID computation
  - archiving is slowest part of applying update
  - roughly 4 times slower than other computation
  - 3% overhead for small updates; 50%+ for large updates
Current Dissemination Approach

- “Tentative” data is not shared
- “Unnamed” data shared via dissemination tree
- **GUID** - hash of content and archive fragments
  - slow to compute (?)
  - globally verifiable name
  - published in Tapestry
- **VHASH** - simple secure hash of contents
  - fast to compute
  - verifies only reconstructed blocks
  - communicated via dissemination tree
  - useless for verifying content from archive

- **COMPLEXITY PERVADES CODE**
Proposed Dissemination Approach

- Simplify, simplify, simplify
- Only external name for data is the GUID
- Data may have other names within a single node
- Validate local computation with inner ring certificates
Proposed Dissemination Approach

- Still possible to share uncommitted data
  - Shared via updates instead of actual data
  - Updates include verifiability measures
  - Nodes apply update locally
  - Verify local computation by inner ring certificate

- Outstanding issues
  - How to populate a new replica?
  - How to recover from missed update?
Other Issues

- Better metadata handling
- Support for version branching
- Support for tentative updates
- Format translation
Conclusions

• Conflicting requirements

• Prototype design met most requirements

• Proposed changes help satisfy remaining requirements
  – Handling of encrypted data
  – Efficient insertions
  – Efficient dissemination