Efficiently Binding Data to Owners in Distributed Content-Addressable Storage Systems

Patrick Eaton, Hakim Weatherspoon, and John Kubiatowicz
University of California, Berkeley

Security in Storage Workshop
December 13, 2005
Big Picture

- Wide-area distributed storage service
- Exploit heterogeneity for improved availability and durability
Content-Addressable Storage

- Address data by function of content
  - Independent of physical location
- Simple `put()`/`get()` interface

\[
\text{put}(\text{name}, \ \text{Data})
\]

\[
\text{get}(\text{name}) = \ \text{Data}
\]

CAS

Name => Data
Advantages of CAS Interface

• Protect data integrity
  – Client can verify data retrieved from system

• Scalability
  – System can replicate, transfer data freely

• Adopted by distributed storage systems
  – CFS, OceanStore, Venti, etc.
Self-Verifying Data

• Data that can be verified using the name of the data and the data itself
• Hash-verified data - immutable
  – Name is a function of the data
    \[ \text{name} = f(\text{Data}) \]
• Key-verified data - mutable
  – Name is a function of the public key
  – Data includes a certificate signed by key pair includes a verifier that describes data

\[ \text{name} = f(\text{pub}) \quad \text{Cert: name} = f(\text{Data}) \quad \text{priv} \]
Apps and Self-Verifying Data

• Combine blocks into larger data structures with hash chaining/Merkle trees
Missing Feature

- Identify owner/writer of data
  - Enable per-user storage quotas
  - Enable per-user billing
Enabling Owner Identification

• Implementation is conceptually simple...
  – Attach a certificate to each block of data

• ...but practically hard
  – Creating, managing certificates is costly
An Illustrative Example

- Consider archiving 1 TB of data
  - Divide into 8 KB blocks
- More than 6 days to sign certificates
  - 3 GHz processor, 1024-bit RSA crypto
A Strawman Solution

• Aggregate blocks into collections
  – One certificate for each collection

• Archive 1 TB of data in 4 MB collections
  – Sign certificates in 17 minutes!
Issues with Aggregation

• How to handle small writes?
  – Buffer data ⇒ Limits durability
  – Small collections ⇒ Limits efficiency
• How to name and access data?
  – Read whole collection ⇒ Wastes bandwidth
Goals

• Identify owners of data
• Self-verifiability
  – Check integrity of blocks and collections
• Incremental update
  – Support small writes efficiently
• Fine-granularity access
  – Support small reads efficiently
• Decrease infrastructural overhead
  – Let system benefit from aggregation too
Outline

• Introduction
• Efficient Aggregation for CAS
• Building Applications
• Prototypes
• Future Work and Conclusion
Three Building Blocks

- **Extents**
  - A container for blocks
- **The append() operation**
  - Allow changes to an extent
- **Two-level naming**
  - Allow block-level access
Extents

- **Container for collection of blocks**
  - Blocks are of variable size
  - Blocks owned by single user
- **Each extent contains certificate**
  - Identifies owner
  - Includes verifier
- **Two types of extents**
  - Hash-verified ⇒ immutable
  - Key-verified ⇒ mutable
The append() Operation

- Allow modifications to extent
  - Not just “replace all”
- Enables incremental update
- Why only append()?  
  - Sufficient
  - Easier naming/verifiers
Two-Level Naming

- Name block by tuple
  - Name = (extent name, block name)
- Enables fine-granularity access
- Infrastructure tracks extents

get(extent, block)
Extents for CAS Systems

get(extent, block)

append( )

CAS
An API for Extents

Key-verified Extents
- create
- append
- truncate

Hash-verified Extents
- snapshot
- put
- get_cert
- get_blocks
- get_extent
Block Names, Verifiers, Extent Names

- Name blocks by secure hash
- Create extent verifiers by chaining
- Hash-verified extents ⇒ name = verifier
- Key-verified extents ⇒ name = F(public key)

\[ V_i = H(V_{i-1} + H(D_i)) \text{ where } V_{-1} = H(PK) \]
Outline

• Introduction
• Efficient Aggregation for CAS
• Building Applications
• Prototypes
• Future Work and Conclusion
Writing Apps with the API

• Can we still build apps with this API?
Writing Apps with the API

- How to create pointers for Merkle trees?
- Logically, construct chain of extents
- Identify extent by sequence number
Writing Apps with the API

- Chain metadata stores mapping to earlier extents
Outline

• Introduction
• Efficient Aggregation for CAS
• Building Applications
• Prototypes
• Future Work and Conclusion
Dummy Prototype

- Implements API as centralized server
- Goals
  - Validate API
  - Isolate client behavior
Client Evaluation

• **Throughput micro-benchmark**
  – Single client submits updates
  – An update is a set of 4 KB blocks

• **When extent is full, perform snapshot() and truncate()**
Client Throughput (Dummy Prototype)
Antiquity Distributed Prototype

- Demonstrate distributed implementation of API
- Main components
  - Storage servers
  - Admin
- Deployments
  - PlanetLab
  - Berkeley PSI cluster
Client Throughput (Antiquity - cluster)
Antiquity Latency (cluster)

CDF of Operation Latency

- Append
- Truncate
- Create
- Snapshot
- Put

Fraction of Events vs. Latency (ms)
Outline

• Introduction
• Efficient Aggregation for CAS
• Building Applications
• Prototypes
• Future Work and Conclusion
Other Interesting Problems

• Maintaining data
  – Policies for mutable and immutable?
  – How many replicas to create?
  – Where to place extent replicas?
  – When to repair extent replicas?
Future Work

• Support for weakly-connected clients
  – Certificate acts as a summary for an update
  – Decouple certificate propagation and data transfer
  – Trade-off visibility/durability for response time
Conclusions

• CAS promotes integrity and scalability
• But, accounting is hard in CAS systems
• Problem is efficiency
• Aggregation is the answer
• Support aggregation by extending API
Efficiently Binding Data to Owners in Distributed Content-Addressable Storage Systems

Patrick Eaton, Hakim Weatherspoon, and John Kubiatowicz
University of California, Berkeley

Security in Storage Workshop
December 13, 2005