Why OceanStore?

Data Storage as Utility
Inherently Untrusted Infrastructure

The Data Utility Vision

- What if we could treat data [storage] as a utility?
  - Multiple providers
  - Economies of scale – high availability, performance, reliability

*Think Google, Amazon, Microsoft...*
Inherently Untrusted Infrastructure

1. No servers are trusted with data
   - All data is encrypted

2. Data should be “anywhere, anytime”
   - Nomadic data, promiscuous caching
   - Ensures high availability, robustness

Both follow from the (idealized) data utility model, where providers are freely interchangeable.

OceanStore Architecture

Naming
Access Control
Data Location & Routing
Update Model
Hybrid Serialization
Deep Archival
OceanStore API
Introspection
OceanStore Architecture

Naming
Access Control
Data Location & Routing
Update Model
Hybrid Serialization
Deep Archival
OceanStore API
Introspection

Quick Stuff

Naming
- GUID identifies all objects (files, directories, servers)
- Self-certified by hashing private key + name

Access Control
- Reader restriction – cryptography at client
- Writer restriction – Access Control Lists at server

Façade Legacy API
- E.g. provides POSIX-like file interface
OceanStore Architecture

- Naming
- Access Control
- Data Location & Routing
- Update Model
- Hybrid Serialization
- Deep Archival
- OceanStore API
- Introspection

Combined Data Location & Routing

- Two Stage Routing
  - Fast probabilistic Attenuated Bloom Filters
  - Deterministic Plaxton randomized hierarchical distributed mesh

- Data Location
  - Objects have a preferred root nodes (*primary tier*)
  - Each node maintains destination map of nearby objects
  - Short-circuit queries directly to destination

*With data replication, this scheme is fault-tolerant.*
Each node maintains an array of Bloom filters per edge.
Bloom filter $i$ is a union of objects at exactly $i$ hops away.
Attenuated Bloom Filter (e.g.)

Basic idea: treat node-ID as location in tree, rooted at the destination node—$O(\log n)$ traversal.

Route by resolving ID piece-wise.
Example routing table for an 8-bit address, taken 2 bits at a time. Other pages assume a 16-bit address taken 4 bits at a time. Note that the project shifted from prefix-based to suffix-based at some point. They are equivalent.

- L2 nodes all share at least a 1-digit suffix w/ the node's ID ("1")
- L3 nodes all share at least a 2-digit suffix w/ the node's ID ("31")
- L4 nodes all share at least a 3-digit suffix w/ the nodes ID ("331")

Routing:
- Node first uses local L1 column to route a request to a node sharing one digit of suffix w/ dest.
- Second node uses it's local L2 column to find a node that shares a length 2 suffix w/ the dest.
- Third node uses it's local L3 column to find a node that shares a length 3 suffix w/ the dest.
- etc..
Plaxton Mesh (e.g. to 4598)

Plaxton Mesh (e.g. to 4598)
Plaxton Mesh (e.g. to 4598)

Plaxton Mesh + Location
Bayou-like Update Model

- *Predicate-action*: Commit updates when predicates hold true.

- **Can only operate on cipher-text due to untrusted servers!**
  - Limited predicates: *compare-version, compare-size, compare-block, search.*
  - Limited updates: *replace-block, append, insert-block, delete-block*
  - Indirection *index-block* to support insert/delete.

Hybrid Serialization Scheme

Concurrent optimistic and pessimistic updates:

1. *Primary tiers* use slow Byzantine agreement
2. *Secondary tiers* use fast tentative epidemic updates
Serialization – Update

Serialization – Agreement
Serialization – Disseminate

Deep Archival (encode)

- Archive data robustly using erasure codes (e.g. interleaved Read-Solomon, Tornado)
Deep Archival (decode)

- Can reliably recreate the object using any $n$ fragments
- Fragmentation increases reliability

Introspection

- *How to administer/tune millions of servers?*
- Introspection facility to summarize node activity
- Can be used for:
  1. Recognizing object clusters
  2. Managing replicas/archive fragments
Pond: the OceanStore Prototype

Technologies

- Crypto
  - public key
  - private key
  - threshold signatures
- Security
  - modified Byzantine fault tolerance
- DOLR
  - tapestry
Data Object

- Ordered sequence of read-only versions
  - AGUID names all versions
    - hash of app-specific name and creator’s public key
  - Blocks kept in B-tree
    - VGUID top block, merkle tree
  - every version sort-of kept forever

App-Specific Consistency

- Updates are *atomic*, and have:
  - Predicates
    - version, comparing range of bytes to expected value
  - Actions
    - replacing range, appending, truncating
  - Can implement ACID transactions?
Replication

- Primary copy
  - each object has a primary replica
- On update
  - primary creates cert mapping AGUID to VGUID
- Cert (also called “heartbeat”) contains:
  - AGUID, VGUID, timestamp, version number
- To get latest, a client:
  - sends nonce
  - primary returns signed nonce, client name, cert
- Primary also
  - access control
  - serializes updates (how across multiple objects?)
  - … is actually a byzantine FT group

Archival Storage and Caching

- Erasure coding
  - immediately after being committed at primary
  - m = 16, n = 32, r = 0.5
  - any oceanstore server can host a fragment
    - chosen by hash of BGUID and fragment #
  - how maintained?
Caching

- Whole-block caching
  - reader publishes own location after retrieving
    - not clear how load balancing done
    - some sort of dissemination tree for updates
    - soft-state
  - latest found by first retrieving heartbeat from primary

Primary replica

- Castro Byzantine
  - using public keys
  - message authentication codes (MACs)
- Oceanstore: MACs + proactive threshold signatures
  - create $l = 3f + 1$ signature shares
  - any $k=f+1$ can sign
  - change inner ring by creating $l$ new shares
    - overall signature does not change
Prototype Implementation

- All major subsystems operational
  - Self-organizing Tapestry base
  - Primary replicas use Byzantine agreement
  - Secondary replicas self-organize into multicast tree
  - Erasure-coding archive
  - Application interfaces: NFS, IMAP/SMTP, HTTP
- Event-driven architecture
  - Built on SEDA
- 280K lines of Java (J2SE v1.3)
  - actually more like 50k
  - JNI libraries for cryptography, erasure coding

Deployment on PlanetLab

- http://www.planet-lab.org
  - ~100 hosts, ~40 sites
  - Shared .ssh/authorized_keys file
- Pond: up to 1000 virtual nodes
  - Using custom Perl scripts
  - 5 minute startup
- Gives global scale for free