Pond: the OceanStore Prototype

Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz

{srhea,eaton,geels,hweather,ravenben,kubitron}@cs.berkeley.edu

University of California, Berkeley
The Challenges

• Maintenance
  – Many components, many administrative domains
  – Constant change
  – Must be self-organizing
  – Must be self-maintaining
  – All resources virtualized—no physical names

• Security
  – High availability is a hacker’s target-rich environment
  – Must have end-to-end encryption
  – Must not place too much trust in any one host
Talk Outline

• Introduction

• System Overview
  - Tapestry
  - Erasure codes
  - Byzantine agreement
  - Putting it all together

• Implementation and Deployment

• Performance Results

• Conclusion
The Technologies: Tapestry

• Tapestry performs

Distributed Object Location and Routing

• From any host, find a nearby…
  – replica of a data object

• Efficient
  – $O(\log N)$ location time, $N = \#$ of hosts in system

• Self-organizing, self-maintaining
The Technologies: Tapestry (con’t.)
The Technologies: Erasure Codes

- More durable than replication for same space

- The technique:
The Technologies: Byzantine Agreement

• Guarantees all non-faulty replicas agree
  – Given $N=3f+1$ replicas, up to $f$ may be faulty/corrupt

• Expensive
  – Requires $O(N^2)$ communication

• Combine with primary-copy replication
  – Small number participate in Byzantine agreement
  – Multicast results of decisions to remainder
Putting it all together: the Path of a Write

- HotOS Attendee
- Primary Replicas
- Archival Servers (for durability)
- Secondary Replicas (soft state)
- Other Researchers
Talk Outline

• Introduction

• System Overview

• Implementation and Deployment

• Performance Results

• Conclusion
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 (J 2SE v1.3)
  – 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
Talk Outline

• Introduction

• System Overview

• Implementation and Deployment

• Performance Results
  – Andrew Benchmark
  – Stream Benchmark

• Conclusion
Performance Results: Andrew Benchmark

• Built a loopback file server in Linux
  – Translates kernel NFS calls into OceanStore API

• Lets us run the Andrew File System Benchmark
## Performance Results: Andrew Benchmark

- **Ran Andrew on Pond**
  - Primary replicas at UCB, UW, Stanford, Intel Berkeley
  - Client at UCB
  - Control: NFS server at UW

- **Pond faster on reads: 4.6x**
  - Phases III and IV
  - Only contact primary when cache older than 30 seconds

- **But slower on writes: 7.3x**
  - Phases I, II, and V
  - Only 1024-bit are secure
  - 512-bit keys show CPU cost

### Phase Comparison

<table>
<thead>
<tr>
<th>Phase</th>
<th>NFS</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0.9</td>
<td>2.8</td>
<td>6.6</td>
</tr>
<tr>
<td>II</td>
<td>9.4</td>
<td>16.8</td>
<td>40.4</td>
</tr>
<tr>
<td>III</td>
<td>8.3</td>
<td>1.8</td>
<td>1.9</td>
</tr>
<tr>
<td>IV</td>
<td>6.9</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>V</td>
<td>21.5</td>
<td>32.0</td>
<td>70.7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>47.0</strong></td>
<td><strong>54.9</strong></td>
<td><strong>120.3</strong></td>
</tr>
</tbody>
</table>

(times in milliseconds)
Closer Look: Write Cost

- Byzantine algorithm adapted from Castro & Liskov
  - Gives fault tolerance, security against compromise
  - Fast version uses symmetric cryptography

- Pond uses threshold signatures instead
  - Signature proves that $f+1$ primary replicas agreed
  - Can be shared among secondary replicas
  - Can also change primaries w/o changing public key

- Big plus for maintenance costs
  - Results good for all time once signed
  - Replace faulty/compromised servers transparently
Closer Look: Write Cost

• **Small writes**
  - Signature dominates
  - Threshold sigs. slow!
  - Takes 70+ ms to sign
  - Compare to 5 ms for regular sigs.

• **Large writes**
  - Encoding dominates
  - Archive cost per byte
  - Signature cost per write

<table>
<thead>
<tr>
<th>Phase</th>
<th>4 kB write</th>
<th>2 MB write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validate</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>Serialize</td>
<td>6.1</td>
<td>26.6</td>
</tr>
<tr>
<td>Apply</td>
<td>1.5</td>
<td>113.0</td>
</tr>
<tr>
<td>Archive</td>
<td>4.5</td>
<td>566.9</td>
</tr>
<tr>
<td>Sign Result</td>
<td>77.8</td>
<td>75.8</td>
</tr>
</tbody>
</table>

(times in milliseconds)
Closer Look: Write Cost

Update Throughput vs. Update Size

- Ops/s, Archive Disabled
- Ops/s, Archive Enabled
- MB/s, Archive Disabled
- MB/s, Archive Enabled

Total Update Operations per Second

Size of Update (kB)

Total Bandwidth (MB/s)

(run on cluster)
**Closer Look: Write Cost**

- **Throughput in the wide area:**

<table>
<thead>
<tr>
<th>Primary location</th>
<th>Client location</th>
<th>Tput (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>Cluster</td>
<td>2.59</td>
</tr>
<tr>
<td>Cluster</td>
<td>PlanetLab</td>
<td>1.22</td>
</tr>
<tr>
<td>Bay Area</td>
<td>PlanetLab</td>
<td>1.19</td>
</tr>
<tr>
<td></td>
<td>(archive on)</td>
<td></td>
</tr>
</tbody>
</table>

- **Wide Area Throughput**
  - Not limited by signatures
  - Not limited by archive
  - Not limited by Byzantine process bandwidth use
  - Limited by client-to-primary replicas bandwidth
Talk Outline

- Introduction
- System Overview
- Implementation and Deployment
- Performance Results
  - Andrew Benchmark
  - Stream Benchmark
- Conclusion
Closer look: Dissemination Tree

HotOS Attendee

Primary Replicas

Archival Servers

Secondary Replicas

Other Researchers
Closer look: Dissemination Tree

- **Self-organizing application-level multicast tree**
  - Connects all secondary replicas to primary ones
  - Shields primary replicas from request load
  - Save bandwidth on consistency traffic

- **Tree joining heuristic ("first-order" solution)**:
  - Connect to closest replica using Tapestry
    - Take advantage of Tapestry’s locality properties
  - Should minimize use of long-distance links
  - A sort of poor man’s CDN
Performance Results: Stream Benchmark

• **Goal:** measure efficiency of dissemination tree
  - Multicast tree between secondary replicas

• **Ran 500 virtual nodes on PlanetLab**
  - Primary replicas in SF Bay Area
  - Other replicas clustered in 7 largest PlanetLab sites

• **Streams writes to all replicas**
  - One content creator repeatedly appends to one object
  - Other replicas read new versions as they arrive
  - Measure network resource consumption
Performance Results: Stream Benchmark

- Dissemination tree uses network resources efficiently
  - Most bytes sent across local links as second tier grows
- Acceptable latency increase over broadcast (33%)
Related Work

• Distributed Storage
  – Traditional: AFS, CODA, Bayou
  – Peer-to-peer: PAST, CFS, Ivy

• Byzantine fault tolerant storage
  – Castro-Liskov, COCA, Fleet

• Threshold signatures
  – COCA, Fleet

• Erasure codes
  – Intermemory, Pasis, Mnemosyne, Free Haven

• Others
  – Publius, Freenet, Eternity Service, SUNDR
Conclusion

• OceanStore designed as a global-scale file system

• Design meets primary challenges
  – End-to-end encryption for privacy
  – Limited trust in any one host for integrity
  – Self-organizing and maintaining to increase usability

• Pond prototype functional
  – Threshold signatures more expensive than expected
  – Simple dissemination tree fairly effective
  – A good base for testing new ideas
More Information and Code Availability

• More OceanStore work
  – Overview: ASPLOS 2000
  – Tapestry: SPAA 2002

• More papers and code for Pond available at

  http://oceanstore.cs.berkeley.edu