Chelonia

A Distributed Cloud Storage Prototype

Jon Kerr Nilsen
Dept. of Physics
Univ. of Oslo
j.k.nilsen@fys.uio.no
Chelonia Outline

- Features
- Components
- Test results
- Conclusion
The Chelonia Cloud

Storage cloud for anyone
- Can be created by anyone
- Can be shared with anyone
- Can be used by anyone
Global Namespace

- Global hierarchical namespace
  - Files are organized in collections
  - All users see the same tree
- Can use logical names
  - Paths in Chelonia namespace
  - Similar to paths in a regular local filesystem
Replication

- Files are replicated
- User specifies how many replicas are needed
- Chelonia itself takes care of the replication
- Easy to create new storage node
Access Control

- Access policies to files and collections
- Can grant access to
  - Individual users
  - If in a Grid environment, to entire virtual organizations
Several ways to access Chelonia

- Command line interface
- FUSE module
- In an ARC enabled Grid environment
  - Through job specification file
  - With ARC client tools
Chelonia Components

- **The Bartender**
  - User interface
- **The Shepherd**
  - Store files
- **The Librarian**
  - Handles metadata
- **The A-Hash**
  - Stores metadata
- **All services replicated**
Tests

- Depth test
- Width test
- Multiple clients test
- A-Hash performance test (if time)
Depth test
(over-organizing your home dir?)

Depth 0:
- Create root collection

Depth 1:
- Create $m$ collections in root collection

Depth 2:
- Create $m$ more collections inside one of the first collections

Depth $n$:
- Create $m$ more inside one of the depth $n-1$ collections

Depth test, centralized services
Depth test, WAN

- Client and bartender in Uppsala, Sweden
- Librarian and replicated A-Hash in Oslo, Norway
- Creating at ~ same speed
- Stating a little bit faster at deeper levels (due to replicated A-Hash)

Depth test, distributed services
Create \( m \) collections under root collection

Create \( n \) sub-collections in each of \( m \) collections

Average over \( m \) timings

Creating (much) faster than stat'ing for high number of entries

Width test, centralized services
Width test, WAN

- Client and bartender in Uppsala, Sweden
- Librarian and replicated A-Hash in Oslo, Norway
- Creating fast, stat'ing even slower
- In stat'ing full collection list is sent between all services → big messages over WAN
Multiple clients
(Hey guys! I think I found Higgs!)

- Services set up in Uppsala and Oslo
- Replicated A-Hash
- Multiple clients access system simultaneously
- Mean response time slowly increasing
- Fastest response time near constant
The replicated A-Hash is master-slave
- If master goes down, no writing possible
- As long as master is down, Chelonia stops
- Master is automatically elected through Paxos algorithm
- So does it work?
A-Hash test (cont'd)

Repeatedly write and get an A-Hash entry

- With centralized A-Hash
- With replicated A-Hash
  - Stable
  - Killing A-Hash clients
  - Killing A-Hash clients or master
  - Killing master
Reading from A-Hash

- Centralized
- Replicated, stable
- Unstable clients
- Unstable master
- Unstable servers

Avg (ms) vs Min (ms)
Reading from A-Hash (worst case)

- Centralized
- Replicated, stable
- Unstable clients
- Unstable master
- Unstable servers
Writing to A-Hash

Centralized
Replicated, stable
Unstable clients
Unstable master
Unstable servers

Avg (ms) Min (ms)
Writing to A-Hash (worst case)

- Centralized
- Replicated, stable
- Unstable clients
- Unstable master
- Unstable servers

Note: seconds!

Max (s)
Conclusion

- Initial tests looks promising
- Planned tests:
  - Long term stability testing in progress
    - CPU usage
    - Memory usage
  - File replication
- Need real-life testing (users)
See us at EGEE'09 (Sept. 21-25)

Demo booth
Mon 16:30-17:00
Wed lunch

Demo preview @ youtube:

Chelonia - a lightweight self-healing distributed storage
Backup slides
Components of Chelonia

- Consists of four services
  - The Bartender
  - The Shepherd
  - The Librarian
  - The A-Hash
- Can have multiple instances of all services
The Shepherd

- Manages the storage node with the actual file data
- Replication requires more storage nodes, each managed by a Shepherd
- Adding a Shepherd, replica will automatically be created (if needed)
The Librarian and A-Hash

- The Librarian manages the metadata
- Uses the A-Hash to store the data
- Both metadata and Librarians can be replicated
  - More fault tolerant
  - Better load balance
Serves the users

- By negotiating with
  - the Librarian to query files and collections
  - the Shepherd to upload and download files

- More than one Bartender to eliminate single point of failure

- Users can contact any Bartender and get the same result
## Reading from A-Hash (table)

<table>
<thead>
<tr>
<th>Level</th>
<th>Average (s)</th>
<th>Min. (s)</th>
<th>Max. (s)</th>
<th>No. of requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>····378</td>
<td>····330</td>
<td>····1340</td>
<td>153126</td>
</tr>
<tr>
<td>Replicated, stable</td>
<td>····374</td>
<td>····320</td>
<td>····1326</td>
<td>154798</td>
</tr>
<tr>
<td>Replicated, unstable clients</td>
<td>····370</td>
<td>····310</td>
<td>····1289</td>
<td>154201</td>
</tr>
<tr>
<td>Replicated, all nodes unstable</td>
<td>····379</td>
<td>····337</td>
<td>····1230</td>
<td>153802</td>
</tr>
<tr>
<td>Replicated, always kill master</td>
<td>····376</td>
<td>····344</td>
<td>1.97113</td>
<td>153478</td>
</tr>
</tbody>
</table>
### Writing to A-Hash (table)

<table>
<thead>
<tr>
<th>Level</th>
<th>Average (s)</th>
<th>Min. (s)</th>
<th>Max. (s)</th>
<th>No. of requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>0.00426</td>
<td>0.00383</td>
<td>0.01446</td>
<td>01388</td>
</tr>
<tr>
<td>Replicated, stable</td>
<td>0.03390</td>
<td>0.01687</td>
<td>1.05760</td>
<td>10905</td>
</tr>
<tr>
<td>Replicated, unstable clients</td>
<td>0.03424</td>
<td>0.01643</td>
<td>1.13114</td>
<td>10727</td>
</tr>
<tr>
<td>Replicated, all nodes unstable</td>
<td>0.03543</td>
<td>0.01652</td>
<td>10.45923</td>
<td>10639</td>
</tr>
<tr>
<td>Replicated, always kill master</td>
<td>0.04487</td>
<td>0.04487</td>
<td>60.90286</td>
<td>9156</td>
</tr>
</tbody>
</table>