MetaSync

File Synchronization Across Multiple Untrusted Storage Services

Seungyeop Han (syhan@cs.washington.edu)
Haichen Shen, Taesoo Kim*, Arvind Krishnamurthy, Thomas Anderson, and David Wetherall

University of Washington *Georgia Institute of Technology
File sync services are popular

400M of Dropbox users reached in June 2015
Many sync service providers

- Dropbox (2GB)
- Google Drive (15GB)
- MS OneDrive (15GB)
- Box.net (10GB)
- Baidu (2TB)
Can we rely on any single service?

Cloud Storage Often Results in Data Loss

All The Different Ways That 'iCloud' Naked Celebrity Photo Leak Might Have Happened

Shutting down Ubuntu One file services

Dropbox confirms that a bug within Selective Sync may have caused data loss (githubusercontent.com)

128 points by ghuntley 6 days ago | comments
Existing Approaches

• Encrypt files to prevent modification
  – Boxcryptor

• Rewrite file sync service to reduce trust
  – SUNDR (Li et al., 04), DEPOT (Mahajan et al., 10)
MetaSync: Can we build a better file synchronization system across multiple existing services?

Higher availability, greater capacity, higher performance
Stronger confidentiality & integrity
Goals

• Higher availability
• Stronger confidentiality & integrity
• Greater capacity and higher performance

• No service-service, client-client communication
• No additional server
• Open source software
Overview

• Motivation & Goals
• MetaSync Design
• Implementation
• Evaluation
• Conclusion
Key Challenges

• Maintain a *globally consistent view* of the synchronized files across multiple clients

• Using only the service providers’ *unmodified APIs* *without any centralized server*

• Even in the *presence of service failure*
Design Choices

• How to manage files?
  – Content-based addressing & hash tree
• How to update consistently with unmodified APIs?
  – Client-based Paxos (pPaxos)
• How to spread files?
  – Stable deterministic mapping
• How to protect files?
  – Encryption from clients
• How to make it extensible?
  – Common abstractions
Design Choices

• How to manage files?
  – Content-based addressing & hash tree
• How to update consistently with unmodified APIs?
  – Client-based Paxos (pPaxos)
• How to spread files?
  – Stable deterministic mapping
• How to protect files?
  – Encryption from clients
• How to make it extensible?
  – Common abstractions
Overview of the Design

1. File Management

- Object Store
- Synchronization
- Replication

Backend abstractions

Local Storage

Remote Services

Dropbox

Google Drive

OneDrive

USENIX ATC '15
Object Store

• Similar data structure with version control systems (e.g., git)
• Content-based addressing
  – File name = hash of the contents
  – De-duplication
  – Simple integrity checks
• Directories form a hash tree
  – Independent & concurrent updates
• Files are chunked or grouped into blobs
• The root hash = f12... *uniquely* identifies a snapshot
Files are chunked or grouped into blobs
The root hash = f12... *uniquely* identifies a snapshot
Overview of the Design

MetaSync

Object Store

Synchronization

Replication

Backend abstractions

2. Consistent update

Local Storage

Remote Services

Dropbox

Google Drive

OneDrive

USENIX ATC '15
Updating Global View

Client1

Global View

Client2

Head
Current root hash

Prev
Previously synchronized point

v0 ab1...

Head

Prev

USENIX ATC '15
Updating Global View

Client1

Global View

Client2

Prev

Head

Prev

Head

v0 ab1...
v1 c10...

master

USENIX ATC '15
Updating Global View

Client1

Global View

Client2

USENIX ATC '15
Updating Global View

Client1

Global View

Client2

Head

Prev

Prev

Head

v0 ab1...

v1 c10...

master
Updating Global View

Client1

Global View

Client2

v0 ab1...
v1 c10...
v2 f13...
v2 7b3...

master

USENIX ATC '15
Updating Global View

Client1

Global View

Client2

v0 ab1... v1 c10... v2 f13...

v2 7b3...

Prev

Head

Prev

Head

master

USENIX ATC '15
Updating Global View

Client1

Global View

Client2

v0 ab1...
v1 c10...
v2 7b3...
v3 a31...

Prev

Head

master

USENIX ATC '15
Consistent Update of Global View

- Need to handle concurrent updates, unavailable services based on existing APIs
Paxos

• Multi-round non-blocking consensus algorithm
  – Safe regardless of failures
  – Progress if majority is alive
Metasync: Simulate Paxos

• Use an *append-only list* to log **Paxos messages**
  – Client sends normal Paxos messages
  – Upon arrival of message, service appends it into a list
  – Client can fetch a list of the ordered messages

• Each service provider has APIs to build append-only list
  – Google Drive, OneDrive, Box: Comments on a file
  – Dropbox: Revision list of a file
  – Baidu: Files in a directory
Metasync: Passive Paxos (pPaxos)

• Backend services work as passive acceptor
• Acceptor decisions are delegated to clients
Paxos vs. Disk Paxos vs. pPaxos

- **Disk Paxos**: maintains a block per client
  - Gafni & Lamport ’02

### Diagram

**Paxos**
- Proposer
- Acceptor
- Computation
- Propose
- Accept
- Requires acceptor API
- # msgs: O(acceptors)

**Disk Paxos**
- Proposer
- Acceptor
- Disk blocks
- Propose
- Check
- O(clients x acceptors)

**pPaxos**
- Proposer
- Acceptor
- Append-only
- Propose
- Check
- O(acceptors)
Overview of the Design

MetaSync

- Object Store
- Synchronization
- Backend abstractions

3. Replicate objects

Replication

Local Storage

Remote Services

Dropbox
Google Drive
OneDrive

USENIX ATC '15
Stable Deterministic Mapping

• MetaSync replicates objects R times across S storage providers (R<S)

• Requirements
  – Share minimal information among services/clients
  – Support variation in storage size
  – Minimize realignment upon configuration changes

• Deterministic mapping

\[ \text{map} : H \rightarrow \{ s : |s| = R, s \subset S \} \]

  – E.g., map(7a1...) = Dropbox, Google
Implementation

• Prototyped with Python
  – ~8k lines of code
• Currently supports 5 backend services
  – Dropbox, Google Drive, OneDrive, Box.net, Baidu
• Two front-end clients
  – Command line client
  – Sync daemon
Evaluation

• How is the end-to-end performance?

• What’s the performance characteristics of pPaxos?

• How quickly does MetaSync reconfigure mappings?
Evaluation

• How is the end-to-end performance?

• What’s the performance characteristics of pPaxos?

• How quickly does MetaSync reconfigure mappings?
## End-to-End Performance

Synchronize the target between two computers

<table>
<thead>
<tr>
<th></th>
<th>Dropbox</th>
<th>Google</th>
<th>MetaSync</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linux Kernel</strong></td>
<td>2h 45m</td>
<td>&gt; 3hrs</td>
<td>12m 18s</td>
</tr>
<tr>
<td>920 directories</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15k files, 166MB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pictures</strong></td>
<td>415s</td>
<td>143s</td>
<td>112s</td>
</tr>
<tr>
<td>50 files, 193MB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Performance gains are from:

- Parallel upload/download with multiple providers
- Combined small files into a blob
Latency of pPaxos

Latency is not degraded with increasing concurrent proposers or adding slow backend storage service
Latency of pPaxos

Latency is not degraded with increasing concurrent proposers or adding slow backend storage service
Conclusion

• MetaSync provides a secure, reliable, and performant files sync service on top of popular cloud providers
  – To achieve a consistent update, we devise a new client-based Paxos
  – To minimize redistribution, we present a stable deterministic mapping

• Source code is available:
  – http://uwnetworkslab.github.io/metasync/