Device-Transparent Personal Storage

Jacob Strauss, Justin Mazzola Paluska, Chris Lesniewski-Laas
Bryan Ford, Robert Morris, Frans Kaashoek

Quanta Research Cambridge    MIT    Yale

June 17, 2011
Personal Data Management: Point-to-point Synchronization

1. Take photos
2. Go home, sync new photos to desktop

• Good Properties:
  – Local connection: fast & inexpensive
  – Simple to use
Synchronization Among Multiple Devices

- Single server to hold & organize entire collection
- Requires hub be reachable
Store & Fetch from Cloud

- More flexibility than a single hub
- Not always reachable, can be slow
Ad-hoc Manual Management

• Push manually to nearby device for more storage

• Upload to cloud later when connected

• Problem: user must track where objects are
Ideal: Device Transparent Storage

Same global view of data collection from each device
Device-Transparency: Impossible?

- Limited Storage Capacity
  - Can’t put everything everywhere
- Devices might be disconnected
  - Can’t use files stored on unreachable devices
Approach: Split Metadata from Content

• Fully replicate all metadata
  – Small: fits everywhere
• Partially replicate all content
  – Not small: place where needed
Metadata is Useful Alone

- When disconnected & without content:
  - View complete collections of objects
  - Move objects between collections
  - Identify devices that do hold the content
Device-Transparent Storage Approach

• Separate Metadata from Content
  – Global Metadata Replication
  – Partial Content Replication

• Peer-to-peer Continuous Synchronization
  – Approximate global store as connectivity permits

• Automate Conflict Resolution
  – Eventually consistent metadata collection
Eyo Overview

Music Player
Photo Editor
Mail Client
User Application

Overlay Network: UIA [OSDI’06]

Local storage

Personal Data & Device Collections
Eyo API Design

• Challenge: Automated Conflict Resolution

• API Properties
  – First-class version history
  – Explicit metadata and content split
  – Placement policy

• Borrows mechanisms from existing work
  – Distributed File Systems, Optimistic Replication, Version Control Systems
Using the Eyo Storage API

No naming hierarchy

Attribute queries:
List of objects ← lookup(has key ‘content-type’ with value ‘image’)
Using the Eyo Storage API

Eyo manages network transfers
Notifies interested applications when updates arrive
Content Store

• Content block per object
  – Immutable after writing

• Device holds subset of all content
  – Guided by placement rules [Cimbiosys, Perspective]
  – Application specified query mapping objects to set of devices
    • Ex: songs with tag “top-rated” → ipod
Metadata Store

• Metadata includes:
  – everything users need to name and find objects
  – album, song name, artist, location, etc.

• Eyo replicates metadata store to all of user’s devices
  – Each device knows about all objects

• Small enough to store everywhere
  – Small updates: quick, frequent transfers

{ Content-type: audio
  Size: 1234
  Artist: U2
  Album: The Joshua Tree
  Playlists: 80’s
  Rating: 4/5 }
Concurrent Updates to Metadata

-Disconnected changes lead to conflicts
-When and where should these be resolved?

create file A

unreachable

Edit A → B

Edit A → C

?
Handling Conflicts

- Track common ancestor
- Eyo provides version history to applications
- Applications specify predecessors when replacing old versions
- Compare to branches in version control systems
- Permits many concurrency strategies
Handling Conflicts

- Do nothing: fork history
- Pick arbitrarily (based on timestamp)
- Let the user pick a version
  - Media player: song title: A→B and A→C
  - Write a new version that replaces both B and C
Handling Conflicts Automatically

- Use application-specific knowledge
  - Media Player:
    - play song in two places,
    - increment playcount on each
    - Merge to total sum
  - Photo Editor:
    - Tag photos concurrently; merge to include both
  - User never aware a conflict occurred
## Storage API Summary

### Eyo Objects

<table>
<thead>
<tr>
<th>Object ID: 12</th>
<th>Object ID: 34</th>
<th>Object ID: 56</th>
</tr>
</thead>
</table>

### Version 87 Metadata

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-type</td>
<td>Image/jpg</td>
</tr>
<tr>
<td>Content-length</td>
<td>5000</td>
</tr>
<tr>
<td>Aperture</td>
<td>f/5.6</td>
</tr>
<tr>
<td>Resolution</td>
<td>1024x768</td>
</tr>
<tr>
<td>ISO equiv</td>
<td>400</td>
</tr>
<tr>
<td>Name</td>
<td>dog.jpg</td>
</tr>
<tr>
<td>Date</td>
<td>4/27/10</td>
</tr>
<tr>
<td>Predecessor</td>
<td>Version 21</td>
</tr>
<tr>
<td>Content ID</td>
<td>Content 41</td>
</tr>
</tbody>
</table>

### Object ID: 56

- Version ID: 34
- Version ID: 56
- Version ID: 78
- Version ID: 21
- Version ID: 87
- Version ID: 65

### Content Store

ID: 41 Value:
API Implementation Challenges

• Device to Device Connectivity
  – Which devices?
  – Where are they?
  – Secure communication

• Continuous Synchronization
  – Approximates device transparency
  – Send updates between all reachable peers
  – How to do so efficiently?

Provided by UIA [OSDI’06]
Separate Synchronization Protocols

• **Metadata**
  – Fast, frequent, small changes
  – Result in identical collections
  – Use metadata to track content

• **Content**
  – Can be big, slow to move
  – Place objects where they belong
Metadata Synchronization

• Find and send only changed objects from large set of unchanged objects
• Group updates into an immutable Generation
• Single Generation Vector describes set of updates each device has seen
• Single lookup identifies state to send
Passing Content Responsibility

• Exchange responsibility for storing objects
• Does not rely on correct placement rules
• Guarantees at least one live copy
  – Assuming no lost or failed devices
Passing Content Responsibility

- Exchange responsibility for storing objects
- Does not rely on correct placement rules
- Guarantees at least one live copy
  - Assuming no lost or failed devices

Camera can safely delete photo
Eyo: Implementation

• Python per-device daemon
  – RPC for metadata sync
  – http for fetching content (no swarming yet)

• Python and C client libraries
  – Sqlite for metadata storage & queries
  – D-bus for event notifications

• UIA for group identity and communication
  – Users create a group of “my” devices
  – Tracks current locations, builds overlay network
  – Authenticates & Encrypts communication
Evaluation Questions

• What can we do with Eyo that we couldn’t do otherwise?

• Is Eyo’s API a good fit for real applications?
  – How difficult is adapting applications?
  – Usability of explicit version histories?

• Is the metadata-everywhere model feasible?
  – Storage costs?
  – Bandwidth overhead?
Evaluation Approach

• Modify applications to use Eyo
  – Rhythmbox & Quodlibet media players
  – gPodder podcast manager
  – IMAP email gateway
  – Rawstudio photo editor

• Examine new features & scope and types of changes needed
New Ability: Device Transparency

• From a disconnected device

• Browse and organize the entire collection
  – Search for tags
  – View thumbnail images
  – Add and edit tags for all images
  – Show which devices hold objects

• View and edit locally-cached full size image originals
Few Application Changes Needed

• Rawstudio photo editor (C & C++):

<table>
<thead>
<tr>
<th>Original #lines of code</th>
<th>Eyo Version #lines</th>
<th>#lines added</th>
<th>#lines removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>59,767</td>
<td>59,851</td>
<td>1851 (~3%)</td>
<td>1596 (~3%)</td>
</tr>
</tbody>
</table>

Different ‘line’ definitions

– No User Interface changes in these values

• Remaining example applications:
  – Changes limited to <10% of codebase
Applications already have Metadata split

User Interface
- Object A
- Object B
- Object C

Application Core
- Content A,B,C
- Metadata A,B,C

Local Filesystem:
- File A, File B, File C
- File: Metadata DB

Filesystem API
Eyo API Makes Split Explicit

- **User Interface**: Object A, Object B, Object C
- **Application Core**: Reads & Writes, Queries & Notifications
- **Eyo**: Content Store A,B,C, Metadata Store A,B,C
- **Local Storage**: Network Updates

---

33
### Metadata Storage Cost

- **How much metadata?**
- **Look at one personal collection:**

<table>
<thead>
<tr>
<th></th>
<th># objects</th>
<th>total size</th>
<th>Metadata per object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td>724,230</td>
<td>4.3 GB</td>
<td>245B</td>
</tr>
<tr>
<td>Music</td>
<td>5,278</td>
<td>26 GB</td>
<td>511B</td>
</tr>
<tr>
<td>Photos</td>
<td>72,380</td>
<td>122.8 GB</td>
<td>328B</td>
</tr>
</tbody>
</table>

Not very different
Storage Costs: Reasonable for portable devices

- Store collections in Eyo
- Look at resulting metadata size

<table>
<thead>
<tr>
<th></th>
<th># objects</th>
<th>total size</th>
<th>Eyo metadata store size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td>724,230</td>
<td>4.3 GB</td>
<td>529 MB</td>
</tr>
<tr>
<td>Music</td>
<td>5,278</td>
<td>26 GB</td>
<td>5.8 MB</td>
</tr>
<tr>
<td>Photos</td>
<td>72,380</td>
<td>122.8 GB</td>
<td>53 MB</td>
</tr>
</tbody>
</table>

Total: <600MB, mostly from email
Related Work

• Optimistic Replication
  – Cimbiosys, Perspective
  – Coda, Ficus, Bayou, PRACTI, EnsemBlue, Tierstore, Podbase, Ivy

• Point-to-point replication: Rsync, Unison

• Version Control Systems
  – Git, SVN

• Centralized Cloud Topologies
  – MobileMe/iCloud, Gmail/Gears, LiveMesh
Summary

• Device Transparency
• View and manage complete collection
  – From disconnected, storage limited devices
• Eyo
  – Storage API with explicit version histories
  – Continuous peer-to-peer synchronization
  – Good fit for existing applications