CS848 Project Presentation
Cloud Personal Data Management

Robert Robinson

Personal File Management

- Proliferation of files individuals have to manage at home
  - Tax information
  - Family photos
  - Plus many others
- Increasing number of Internet-connected devices at home
  - Desktop, Media PC, Smartphone, iPad (?)
  - Access files remotely
- Effective backup strategies are rare
  - Folder copies are common

Existing Solutions

- Dropbox
  - Effectively used like an online USB key
- Time Machine
  - Designed to back up one machine at a time
  - Not designed for network access
- Network file system + backup script
  - Provides reliable network file access with backups
  - $$$ (2+ machines, 3+ drives)
  - Complex to configure

Proposed Solution

- Networked file system hosted in the cloud
- Support transparent file system use
  - Utilize the FUSE framework for typical file access
  - Specialized admin tools expose additional features
- Utilize S3 as a storage backend for personal data
  - Focus on small – medium size files, ignore videos for now
  - Support single-use workloads
- Enable access from multiple computers
  - Interface directly with S3, all logic on-client

Major Features

- Versioning of files
  - Simplifies retrieving archived file contents
- Redundant storage of files
  - Utilize S3’s built-in redundancy to prevent file loss
- Simplified file hierarchy
  - Users can find hierarchical file systems confusing
  - Trend towards flat file systems
- More flexible file management
  - Support Gmail-like tagging for files

Challenges in Using S3

- Eventual consistency the only guarantee provided
  - Possible to provide POSIX-style guarantees?
- Cross-system locking effectively impossible
  - Using an intermediate tier/SimpleDB possible for locks
  - Avoided to minimize dependences
- Metadata stored at the file level
  - Potentially inefficient to search on
    - Depends heavily on Amazon's implementation
    - Opaque to users, can't depend on its efficiency

Proposed Solution

- Versioned file system built on S3
  - Every update written to a unique file
- Maintain file system metadata in a log structure
  - Log records are write-one, just like files
  - Cache log records at client for improved performance
  - Potentially perform checkpointing
- Enable users to tag files arbitrarily
  - Tags will not be versioned
  - A tag will point to the most recent version of a file
  - Expose files based not only on name but also by tags

Proposed Design

- Each file modification creates a new file
  - Write **sampleFile.txt**
  - File named **sampleFile.txt.r7-workstation1** created
  - Append version and hostname to the file for uniqueness
- File system metadata modifications stored in a log file
  - sampleFile.txt => sampleFile.txt.r7-workstation1
- Concurrent writes will create different files
  - File name updates applied in a most-recent-wins order
  - Updates are never lost

File System Metadata

- Structured as a log of file name mappings
  - Modifications create a new log record
- Metadata is generated by scanning through all records
  - Cache metadata at client to improve performance
- Checkpointing problematic
  - Empirically determine worst-case convergence time?
- Log records can arrive before the corresponding file
  - Verify file exists before applying log

Tag Metadata

- Tags map a user-specified keyword to a file name
  - Tags either exist for a file or don’t – can’t tag versions
  - Simplicity a key factor
- Structured as a series of log records much like metadata
  - All log records for a file contained in a “directory”
- Tags aren’t part of the standard FS API
  - Will require additional tools to add/remove tags
  - Viewing tags can be integrated into the FS API…

File System Structure

- /files
  - Contains the most recent copy of each file
  - Flat hierarchy, directories not supported
- /tags/<**tagName**>
  - Virtual directory – exposes files stored in /files
  - All files that have been tagged with **tagName**
- /history/<**fileName**>
  - Virtual directory – exposes file history found in /files
  - Every version of the specified file

S3 Bucket Structure

- /files
  - Contains all of the file versions created by the user
  - Each file exists as a "directory" with all revisions stored inside
    - /files/sampleFile.txt/sampleFile.txt.r7-workstation1
- /metadata
  - Contains the file system log records & checkpoints
- /tags
  - Contains information about all the created tags
  - All logs pertaining to a given tag exist in their own "directory"
    - /tags/sampleTag/2010-03-29_10-01-42

Implementation Work

- Modifying existing FUSE driver for S3
  - s3fs currently provides basic read/write access
  - Add support for reading/writing file modification logs
  - Expose history & tags via `readdir()`
- Creating utility to tag files
  - Tagging not part of the traditional POSIX file system API
  - Requires additional user interaction

Conclusions

- Building a file system over eventual consistency is difficult
  - User is not guaranteed to see written changes immediately
- Eventual consistency works well with file versioning
  - Users never lose data
  - Cost of duplicate data ignored at present
    - S3 is cheap, and files aren't huge
- Fixed file system hierarchy easy to understand
  - "Tag directories" simplify viewing tagged data