CDMI™ and Cloud Federation Year 4

David Slik
NetApp, Inc.

Session Agenda

- A Brief Overview of CDMI
- Federation Overview
- Local Device “Mini-clouds”
- Demo
- Data Synchronization
- Federation and Versioning
- Application-Specific Conflict Resolution

A Brief Overview of CDMI

- CDMI is an RESTful API for accessing cloud storage. The major cloud storage APIs are:
  - Amazon S3
  - CDMI
  - Microsoft Azure
  - Swift API (part of OpenStack)
- CDMI is widely implemented
  - >30 server implementations
  - CDMI gateway for S3, OpenStack support

A Brief Overview of CDMI

- 2009: SNIA Cloud Technical Working Group founded to explore API standardization
  - Published TWG Charter and Use Cases
- 2011: CDMI 1.0 ratified as a US Technical Architecture
  - CDMI 1.0.1 errata released in late 2011
  - CDMI 1.0.2 errata released in mid 2012
- 2012: CDMI 1.0.2 becomes ISO/IEC 17826
  - An international standard for Cloud Storage
- 2013: CDMI 1.1 under active development
  - 13 Extensions submitted
  - Spec draft in public review

A Brief Overview of CDMI

- Why does CDMI Matter?
  - Simple and easy to implement
    - Start with HTTP and add functionality, few mandatory parts
  - Advanced functionality not found in other APIs
    - Provides a foundation for next generation cloud services, such as federation
  - Open industry standard
    - Not controlled by any one vendor, protection against patents
  - Well defined formal standard
    - Enables interoperability, testing, and cross-vendor support
  - Widespread government support and adoption

A Brief Overview of CDMI

- CDMI Standardizes:
  - CRUD operations (Create/Read/Update/Delete)
  - Data, Container, Queue and Domain objects
  - Identity and access control model
  - Metadata (including client and vendor extensibility)
  - Query and Notifications
  - Versioning
  - Serialization and Deserialization
  - Interoperability with other NAS and cloud protocols

A Brief Overview of CDMI

CDMI AJAX Client Demonstration

Federation Overview

- Federation is the process by which two (or more) separate systems can work together to act as a single system while still retaining autonomy
- Examples of federated systems:
  - The United States (States federated into a country)
  - The World Wide Web (Sites federated via links)
  - Apple iCloud (Devices federated via the cloud)
- CDMI enables storage Federation
  - Proxying & Peering, see Federation Year 2 & Year 3 presentations for more details

Local Device "Mini-Clouds"

- Industry trends are moving towards a model where multiple semi-connected mobile devices revolve around Internet-based cloud storage
  - Devices include phones, tablets, laptops and desktop computers
  - With the "Internet of Things", the number of devices we interact with will dramatically increase
- Devices typically have local storage, and allow data to be created, viewed, updated and deleted
- User data is synchronized via the cloud

Local Device "Mini-Clouds"

- One architectural model is to consider each device as a "mini-cloud" that is federated with devices and clouds to provide a global view
  - This enables device autonomy, disconnected operation, and the ability to freely change providers and federation topologies
- However, information synchronization has traditionally been a "hard problem"
  - CDMI makes this much simpler, as every object has a globally unique object and version identifier

Local Device “Mini-Clouds”

HTML5 Local Storage Mini-Cloud Demonstration

Local Device "Mini-Clouds"

- In the demo, the object and associated metadata is stored locally in the browser
- Locally stored versions can be accessed even if cloud connectivity is unavailable
- When the browser refreshes the listing, it uses the CDMI identifiers to determine if an object is locally stored, and if the stored version is up to date
- This is sufficient information for synchronization

Data Synchronization

- With CDMI, the client can tell if an object is stored local only, remotely only, or both, even if:
  - The object is renamed (locally or remotely)
  - The object is moved into another container (locally or remotely)
  - The object is updated (locally or remotely), and the relationship between the parent and the child is preserved
- This is accomplished by using CDMI Identifiers

Data Synchronization

- Renames:
  - Object is created as /original_location.txt
    - It is assigned object ID 00007ED900104E…
  - Object is renamed to /new_location.txt
    - It still has ID 00007ED900104E…
- Thus, a client can tell that these are the same object, and decide which object needs to be renamed to be in sync

Data Synchronization

- Moves:
  - Object is created at /original/location.txt
    - It is assigned object ID 00007ED900104E…
  - Object is renamed to /new/location.txt
    - It still has ID 00007ED900104E…
- Thus, a client can tell that these are the same object, and decide which object needs to be moved to be in sync

Data Synchronization

- Updates:
  - Object is created with value “41”
    - It is assigned object ID 00007ED900104E…
    - It is assigned version ID 00007ED900103D…
  - Object is updated to have the value “42”
    - It still has ID 00007ED900104E…
    - It has a new version ID 00007ED90010DF…
    - Parent version ID is 00007ED900103D…
- Thus, a client can tell that these are the same object, that the value is changed, and know which direction to synchronize

Federation and Versioning

- Federation client logic, part 1/3:
  - Subscribe to notification feed, perform a query or walk the namespace to get current state of objects of interest
    - For updates from a notification feed, when reconnecting after disconnected operation, optionally coalesce overlapping updates
  - For each cloud originated update, determine if there are any pending local updates by comparing the cloud version parent ID with the local version ID
    - If matches, apply cloud originated update to local storage
    - If it does not match, perform reconciliation step

Federation and Versioning

- Federation client logic, part 2/3:
  - When a local update is performed, add to a local update queue
  - When connected, for each local update, determine if there are any pending cloud updates by comparing the local version parent ID with the cloud version ID
    - If matches, apply local originated update to cloud storage
    - If it does not match, perform reconciliation step

Federation and Versioning

- Federation client logic, part 3/3:
  - Perform application-specific logic to reconcile
    - Merge (requires knowledge of object format, and only works for some data types)
    - Prompt user to determine which one is kept
    - Preserve both (by renaming)
    - Preserve both (by versioning)
  - When reconciliation is required, walk back through the parent IDs on both the cloud and local copy until a common ancestor is identified
    - Allows changes from common ancestor to be determined

App-Specific Conflict Resolution

- Examples:
  - Microsoft Word Document Compare
  - Source Code Management merge
  - Log interleaving
  - Sensor data
- Typically, any unordered or time-ordered record-oriented write-only structure is trivially merged

In Summary

- CDMI enables active-active multi-party bi-directional synchronization (federation) between different storage clouds
- CDMI allows unambiguous eventual consistency
- CDMI makes recovering from disconnected operations simple
- Conflicts that can not be resolved by the storage system can be delegated to an application-level resolver
  - Even if multiple resolvers are active, the system will be eventually consistent

Thank You!

Questions and Answers

Contact Info:

dslik@netapp.com