VOSpace implementations in OpenCADC and VOSpace-next

Patrick Dowler
Canadian Astronomy Data Centre

IVOA InterOp 2023b

VOSpace-2.1

- What is VOSpace?
  - REST API (abstraction) to hierarchical storage: nodes
  - permissions using GMS
  - annotate nodes with properties (simple metadata tags)
  - transfer negotiation with extensible set of transport protocols
  - async server-to-server transfers (client: fire and forget?)

- 2.1 became a REC in 2018, relatively minor updates
- 2.0 became a REC in 2013
- most of the OpenCADC code from early in the CANFAR project
  - 10 year old standard
  - 10 year old code

- CADC/CANFAR: we think this is for user storage…that's really just what we use it for, not what it is

OpenCADC VOSpace implementations

- **libraries**:
  - java libraries for most (>95%) of server implementation
  - python library and client tools

- **vault**: RDBMS for “nodes”, object store for “bytes”
  - distributed storage using CADC archival storage system
  - robust long term storage and preservation
  - currently 230 million files, 900 TiB used

- **cavern**: POSIX filesystem for “nodes” and “bytes
  - CEPH-fs back end
  - mountable into Science Platform containers
  - uses fs attributes (node properties) and fs ACLs (permissions)
  - currently 900 TiB capacity, 325 TiB used
  - **posix-mapper** : helper service to manage local posix uid/gid

OpenCADC VOSpace implementations

- what works well?
  - users organize their own data
  - users control permissions (public flag, GMS groups)
  - enables collaboration
  - transfer negotiation enables robust transfers (with a good client)
  - many opportunities to extend with custom features

- what works poorly?
  - busy data node
  - quotas are hard
  - paging
  - transfer negotiation is too complex, introduces needless latency
  - views are never the best solution to any problem
  - easily shareable link to a file not part of standard
  - recursive operations poorly specified (update nodes, deletion)

Proposal #0

- paging container nodes with uri and limit
    - no overflow/truncation indicator so client makes 1 extra request
    - implementation not very usable and not scalable in posix filesystem

- proposal: make paging optional
    - still require support for limit=0
    - allow InvalidArgument fault? or define UnsupportedOption?

    - how to tell clients what to expect? TBD

VOSpace transfer negotiation

- get data out of a vospace:
  - pullFromVoSpace: give URLs(s), client initiates connection
  - pushFromVoSpace: server initiates connection

- put data into a vospace:
  - pushToVoSpace: give URL(s), client initiates connection
  - pullToVoSpace: server initiates connection

- also internal copy, move, rename

VOSpace transfer negotiation

- general purpose /transfer endpoint (UWS async)
  - overhead: POST job info, redirect to job, POST to start job, poll job, get job result(s), GET transfer details document (with usable URLs)
  - minimally: 5 https requests

- slightly optimised /synctrans endpoint (sync)
  - overhead: POST job info, redirect to job, GET sync job, redirect to transfer details, GET transfer details document (with URLs)
  - minimally: 3 https requests
  - predicated on making sync a layer over async

proposal #1: VOSpace files endpoint

- GET node metadata: /srv/nodes/path/to/a/node

- GET file data: /srv/files/path/to/a/node
  - container node: fail (400?)
  - data node: deliver bytes or redirect to a URL that can

- makes view=data obsolete: deprecate
- makes /synctrans with parameters obsolete: deprecate
- GET only: still need to negotiate for PUT (cannot redirect)

- implemented in **cavern** and **vault**:
  - other mechanisms almost never used but still add complexity
  - users can predictably create URLs to share with collaborators or put in web pages
- makes the simplest thing actually the simplest thing!!

Proposal #2: simplified transfer negotiation

- /transfers for transfers that are inherently async
  - pullToVoSpace
  - pushFromVoSpace
  - internal copy, move

- /synctrans for transfers that are initiated by client
  - pushToVoSpace aka PUT
  - pullFromVoSpace aka GET
  - decouple from async and jobs
  - minimally: POST and read response (implementation may use redirects, clients should just follow)

- rename as a simple sync request? TBD

Proposal #3: explicit recursive operations

- delete node: DELETE /srv/nodes/path/to/a/node
  - currently: delete is implicitly recursive – dangerous!
  - **propose**: delete non-empty container node: FAIL
- **propose**: UWS async job for recursive delete
  - details about behaviour w.r.t. permissions: TBD

- update node (set properties): POST /srv/nodes/path/to/a/node
  - set node properties on that node only
- **propose**: UWS async job for recursive set node props
  - details about behaviour w.r.t. permissions: TBD

- both of these take time to execute that scales with how many nodes they encounter
- service implementations can balance/constrain the load
- users can monitor a job and potentially kill it

Proposal #4: alternate format for API payloads

- VOSpace “node” document: XML, input and output
- VOSpace “transfer” document: XML, input and output
- UWS “job” document: XML, output only

- implemented**: JSON documents with equivalent DOM
  - add JSON format to standard(s)
  - client selected via HTTP accepts header
  - probably: XML format is the default

- child node listing is currently a “node” document with all the child nodes
  - not scalable
  - if paging optional: need list format that is easily streamed and consumed
  - TODO: see if TSV would suffice

VOSpace implementation details

- ready-to-use docker images:
  - images.opencadc.org/platform/cavern
  - images.opencadc.org/storage-inventory/vault (SOON)

  - generally: libraries are published in maven, but we prefer to deliver and support use of pre-built images

- plans for near future:
  - finish new **vault** implementation and migrate content
  - move as much client code to PyVO as possible
  - incremental improvements, support additional types of back end storage

VOSpace implementation details

- open source code in OpenCADC
- core libraries (java):
  https://github.com/opencadc/vos
- **cavern** (posix filesystem):
  https://github.com/opencadc/vos
- **vault** (db + storage-inventory) – IN PROGRESS
  https://github.com/opencadc/storage-inventory
- client tools (python):
  https://github.com/opencadc/vostools

VOSpace-next summary

- paging optional
- files endpoint
- simplified transfer negotiation
- first class JSON support