NRC·CNRC

Cavern: a different VOSpace prototype

Patrick Dowler
Canadian Astronomy Data Centre

prototype: Patrick Dowler, Brian Major, Adrian Damian

Cavern: a different VOSpace prototype

- current CANFAR VOSpace is operational:
  - ~100 users, ~250TiB, ~175 million files
- 3-part implementation:
  - web service front end (REST API)
  - RDBMS for node metadata
  - distributed object store for data node bytes (CADC data archive)
  - highly integrated with CADC/CANFAR authentication and authorization - users control permissions
- command-line clients for users
  - familiar unix style (vls, vcp, vrm, …)
- mountable via FUSE (vofs)
  - uses REST API, negotiated transfers, etc
  - complex implementation (FS + multi-threaded + caching)
  - it works...
  - performance not sufficient: primarily due to overhead

Cavern: a different VOSpace prototype

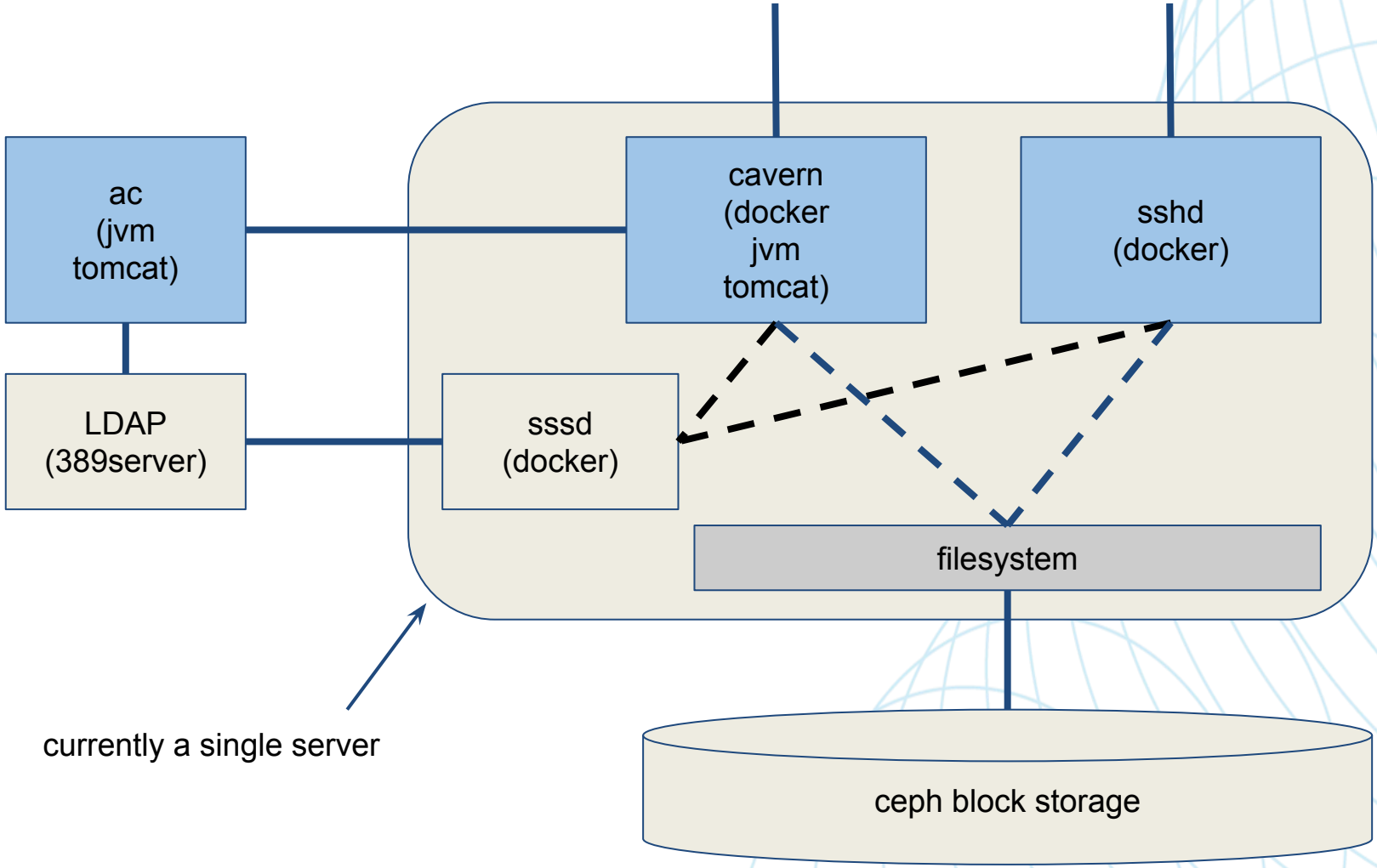
- new VOSpace prototype
- 2-part implementation:
  - web service front end (REST API)
  - scalable filesystem backend
  - complete integration with A&A system - users control permissions
  - allow clients to mount using off-the-shelf tools

- consistency between REST API and mounted access
- scalable to support processing work loads

Cavern: a different VOSpace prototype

- easy bits:
  - use java.nio package to map VOSURI <-> Path (on disk)
  - DataNode <-> file
  - ContainerNode <-> directory
  - LinkNode <-> symbolic link (VOSpace internal copy: hard link)
  - owner of the node <-> owner of the file/directory/link
- tricky bits:
  - file owner
  - store node properties in the filesystem
  - VOSpace permission properties don't match basic POSIX permission model
  - link node behaviour
- always keep in mind: what happens when a user moves a large directory from one place to another?

Cavern: a different VOSpace prototype



Cavern: a different VOSpace prototype -- Owner

- owner of a node: a known CADC/CANFAR user
- owner of a file: POSIX user known to the system
- the CANFAR access control (ac) service built on LDAP backend
  - translation to POSIX user
  - java.nio.file.attribute.UserPrincipalLookupService
- nodes created through VOSpace API must be assigned ownership:
  `java.nio.file.attribute.PosixFileAttributeView`
- requires web service with **chown**
  - run tomcat process as root (ack!)
  - use setcap to give tomcat cap_chown (ummm… scope?)
  - exec a small external program with cap_chown (whack-a-mole)
  - tomcat in container: run as root (ummm… OK?)
  - definitely introduces security concerns….
- worked: ext4, xfs, zfs failed: NFSv3, NFSv4

Cavern: a different VOSpace prototype -- Node Properties

- users can store arbitrary key=value pairs on any kind of node
  - in practice, keys are URIs (length to ~32 chars)
  - in practice, values are very short
  - expect values of ivo://ivoa.net/vospace/core#title to be long-ish
- some node properties in use are not set by users and provided as basic POSIX attributes: length, modification timestamps, etc
- Solution: POSIX extended attributes to store others
  `java.nio.file.attribute.UserDefinedFileAttributeView`

- worked: ext4, xfs, zfs failed: NFSv3, NFSv4

Cavern: a different VOSpace prototype -- Permissions

- VOSpace permission properties don't match basic POSIX permission model
  - ivo://ivoa.net/vospace/core#ispublic == POSIX world-readable
- basic POSIX group permissions could support one of these
  - ivo://ivoa.net/vospace/core#groupread
  - ivo://ivoa.net/vospace/core#groupwrite
  - BUT collaboration use cases require both w/ different groups
- solution: POSIX extended access control lists (ACLs)
  `java.nio.file.attribute.AclFileAttributeView`
  - not implemented in Linux JVM for any file system
  - wrote code to exec getfacl/setfacl (simple and robust)

- worked: ext4, xfs, zfs failed: NFSv3, NFSv4
- pro: enforces permissions when filesystem accessed by user
- con: users cannot necessarily see the permissions that grant access

Cavern: a different VOSpace prototype -- Link Nodes

- in VOSpace API, LinkNode target is a URI
- in filesystem, symbolic link target is a path
  - restrict target to another local node
  - absolute URI of LinkNode target made relative with respect to the root of the vospace
  - relative links work via mount if the mount point is close enough to root to include the target
  - broken links are harmless but users have to grok how mounting and relative links interact (~sysadmin concern)
- ironically, users of the production VOSpace want relative links
  - magic absolute <-> relative not a great solution
  - maybe concept of relative links should be added to standard?

- worked: everywhere
- somewhat complex & magical

Cavern: a different VOSpace prototype -- Mount

- initial prototype: SSHFS
- client uses transfer negotiation to get the mount details
  new Direction: **ivo://cadc.nrc.ca/vospace#biDirectional**
  new Protocol: **ivo://cadc.nrc.ca/vospace#SSHFS**
- resulting endpoint:
  sshfs:pdowler@proto.canfar.net:/blah/blah/pdowler/foo
  - sshfs scheme (because xsd)
  - posix user name
  - ssh server + impl-specific path (/blah/blah)
  - node path to the container being mounted

- works? parts tested, deployment underway
- pros: permissions enforced servers-side
- cons: users cannot see or manipulate ACLs or extended attrs

Cavern: a different VOSpace prototype -- Mount

- future work:
  - performance testing of sshfs
  - put a file server between disks (ceph) and user-facing services (lustre, glusterfs, ???) so we can scale
  - look at alternative user-space network filesystem mount: client+server or direct to low level file server (glusterfs?)

https://github.com/opencadc/vos.git

Cavern: future deployment goal