ICRAR

International
Centre for
Radio
Astronomy
Research

pyvospace

VOSpace Implementation in Python 3

Dave Pallot

Curtin University

THE UNIVERSITY OF WESTERN AUSTRALIA

Government of Western Australia
Department of the Premier and Cabinet
Office of Science

pyvospace

Why the need for pyvospace?

- Works out-of-the-box with very few complex dependencies.
- Makes use of current technologies with scalability and throughput in mind.
- Easily extensible:
    - Add new and existing storage platforms with a simple framework.
    - Customisable user management (Authorisation and Authentication)
- Can be easily integrated into existing supercomputer facilities.
- Basic package runnable within minutes.

pyvospace

Basic features

- VOSpace Specification 2.1
- Features:
  - Python 3.6+ using AsyncIO coroutine framework.
  - PostgreSQL 11 support.
  - Docker deploy scripts.
  - Pluggable Authorisation and Authentication.
  - Framework for custom storage backend development.
  - POSIX storage backend out-of-the-box.
  - FUSE front-end client.
- Funded by Australia Astronomy Limited (AAL)

pyvospace

Architecture

- Metadata and storage services are separate (customisable and scalable)
    - Defers deployment architecture to the administrator (load balancing etc)

Metadata Service

- Maintains consistency of VOSpace node tree.
    - PostgreSQL *ltree* data type.
- Provides basic VOSpace operations:
    - create, move, copy, get, delete, metadata requests (views, properties, etc)
- Provides the Universal Worker Service interface for VOSpace transfers.
    - Offloads final phase of *PushToVoSpace* and *PullFromVoSpace* (sync and async operations) to storage service.

pyvospace

Storage Service

- POSIX storage service available out-of-the-box.
    - Doubles as an example for other developers of storage backends.

Customisable Storage Backend:

- Developer implements 2 functions: *download* and *upload*.
    - Functions provide node and job details (node, view, permissions, parameters)
    - Developer free to decide how to deal with storage aspect.
        - POSIX, S3, Azure blobs etc.
    - Framework has a Transaction API that ensures database node and storage consistency on update/create.

pyvospace

```
async def upload(self, job: StorageUWSJob, request: aiohttp.web.Request):
    # upload data to staging area first
    ….

    async with job.transaction() as tr: # lock the target node (busy) for the duration of transaction
        node = tr.target # get the target node that is associated with the data
        node.size = size # set the size of file
        node.storage = self.storage # set the storage back end so it can be found on pull request
        await node.save() # save details to space db
        await move(stage_file_name, real_file_name) # move file from staging to final location on storage.
Rollback all if any operation fails.
```

pyvospace

Extending Storage Protocols beyond HTTP(S)

1. Implement ***get_transfer_protocols.*** Called on a transfer request:

```
async def get_transfer_protocols(self, job: UWSJob) -> List[Protocol]:
    protocols = job.job_info.protocols
    if isinstance(job.job_info, PushToSpace):
        if httpput() in protocols: # out of the box protocol
            return [Endpoint(“http://storage01/push/jobid/”)] # specify http storage endpoint
        if ftpput() in protocols: # new protocol to storage
            return [Endpoint(“ftp://storage01/push/jobid/”)] # specify ftp storage endpoint
        if <other> in protocols:
            return ...
```

pyvospace

Extending Storage Protocols beyond HTTP(S)

2. Write space storage server. Make use of ***StorageUWSJobPool*** that constructs a UWS job and related Transfer object i.e. destination, target node etc based on Job UUID.

```
pool = StorageUWSJobPool(...)

async def upload_request(self, protocol_request):
    job_id = <extract job UUID from request>
    response = await pool.execute(job_id, my_upload, protocol_request)
    await pool.set_completed(job_id)
    return response

async def my_upload(self, job: StorageUWSJob, protocol_request):
    blob = protocol_request.read()
    ...
```

pyvospace

Limitations and Considerations

- Database:
    - Currently “harded code” to PostgreSQL because of ltree.
        - SQL separate from logic, so relatively simple to add new DB flavours.
    - Database does not validate ltree path:
        - A.B is valid even if A does not exist!
        - Path validation must occur in DB using own functions or defer to application.
    - Potentially excessive row locking due to concurrent nature of the implementation.
        - Potentially reduces throughput and responsiveness under heavy load.
        - More testing and tweaks may be required.
- FUSE library behaves very differently on different platforms.
    - Can not open data in a+ mode, does not make sense in a Space.
    - Working on supporting as many flavours as possible.

pyvospace

Thoughts

- Specification should consider a Space to Space replication feature.
  - Allow for trees to be replicated and synchronised.
  - Opens up the possibility of VOSpace federation(s).
    - Gracefully handle of failover.
    - High availability.
    - Scalability.
- Transfer process is particularly onerous for simple data transfers.
  - Polling for job state is antiquated.
    - Consider adding a state callback mechanism. i.e. URL, websocket etc
  - Sync transfer reduces complexity.
- Consider specifying a base user profile in specification.
  - Links users to node operations, ownership, permissions, jobs, views etc.
  - Allows a more standardised approach for interacting with different Spaces.

pyvospace

Work in Progress and Future Plans

- Trial at Pawsey for MWA and ASKAP users.
    - Develop necessary space views given the scientific data requirements.
- Provide an administration console out-of-the-box.
- A comprehensive web-front end.
- Add S3, NGAS and Azure blob storage options to base package.
    - NGAS will be here soon!
- Improved documentation for operators and developers, logging, comments etc.
- Integration with Travis (automated testing).
- Fixing bugs as they are found!

pyvospace

- Open-source repository: https://github.com/ICRAR/pyvospace
- Any contributions, ideas and comments are welcome.
- Questions?