A Guide to the FENIX Storage Infrastructure

Offerings, Architectures, and Best Practices

Thorsten Hater

Jülich Supercomputing Centre

10 March 2020

The ICEI project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement No 800858.

Agenda

- Introduction
- FENIX Storage Offerings
- Parallel Filesystems
  - Architecture
  - Usage (libraries)
  - Best Practices
- Object Stores
  - Architecture
  - Usage (Python, CLI)
  - Best Practices
- Comparison
- Conclusion, and Q&A

Introduction

STORAGE OFFERINGS

Introduction

- Two environments plus associated storage
  - HPC environment: parallel filesystems
    Active Data Repositories (ACD)
  - Cloud environment: object stores
    Archival Data Repositories (ARD)
- Goals of this talk
  - storage architectures
  - methods of access
  - short guide on performance
- Disclaimer: vast generalisations

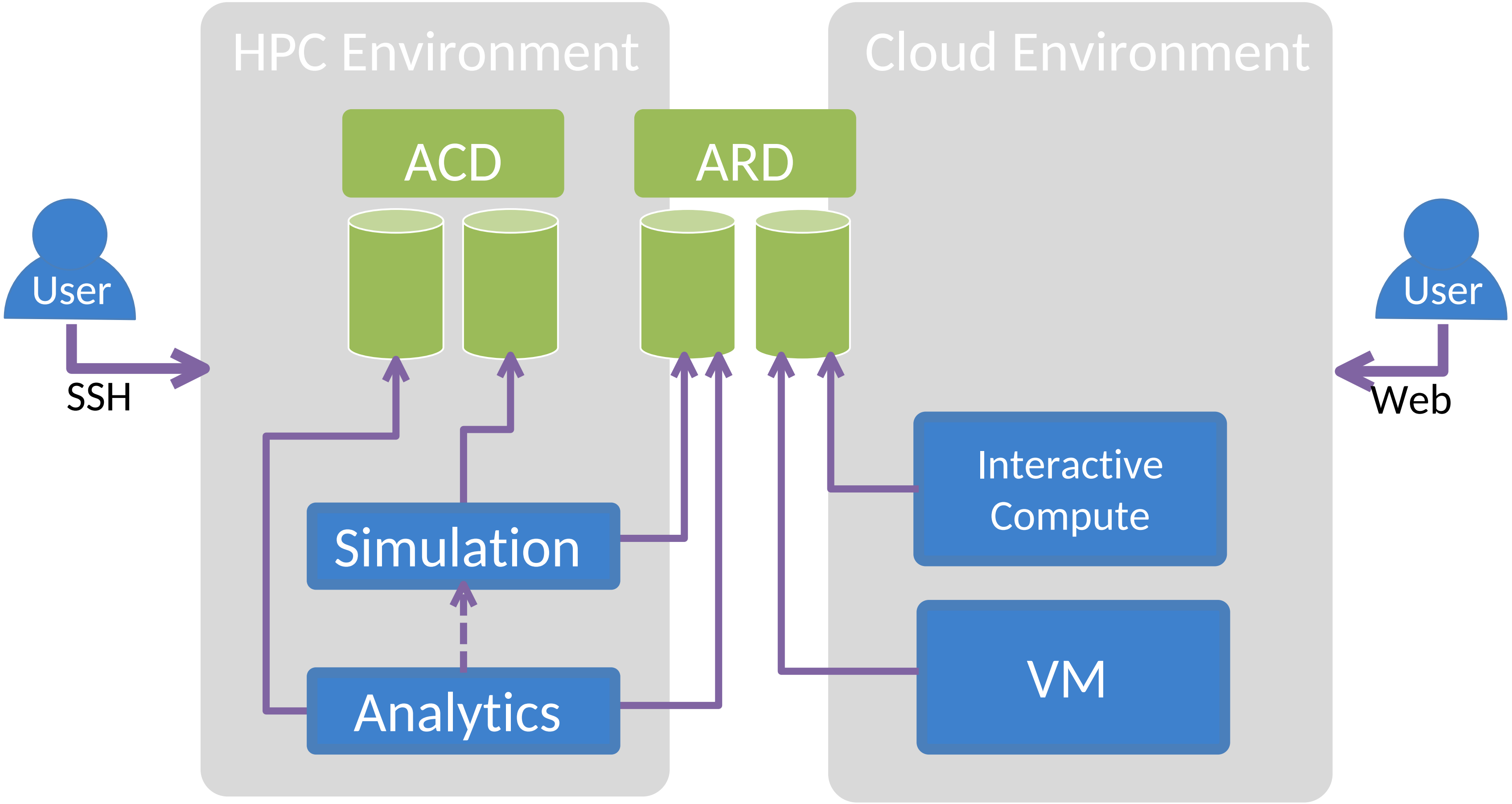
Active Data Repositories (ACD)

- Scalable Compute Services
- ACD usually built as Parallel Filesystems
- "Workhorse" of HPC storage
- Well-known interface: POSIX
  - emulates local storage
  - handles parallel access
  - strong consistency semantics
- Highly optimised for bandwidth
- Examples: SpectrumScale, Lustre, BeeGFS

Archival Data Repositories (ARD)

- Interactive Compute/Virtual Machine Services
- Implemented as Object Stores
  - very popular in Cloud and Web contexts
  - less common in HPC
- Simpler than (parallel) filesystems
  - no guarantees on ordering, but: atomicity
  - put/get semantics
  - flat hierarchies
- Examples
  - Amazon S3
  - OpenStack SWIFT (used in FENIX)

3.048km View

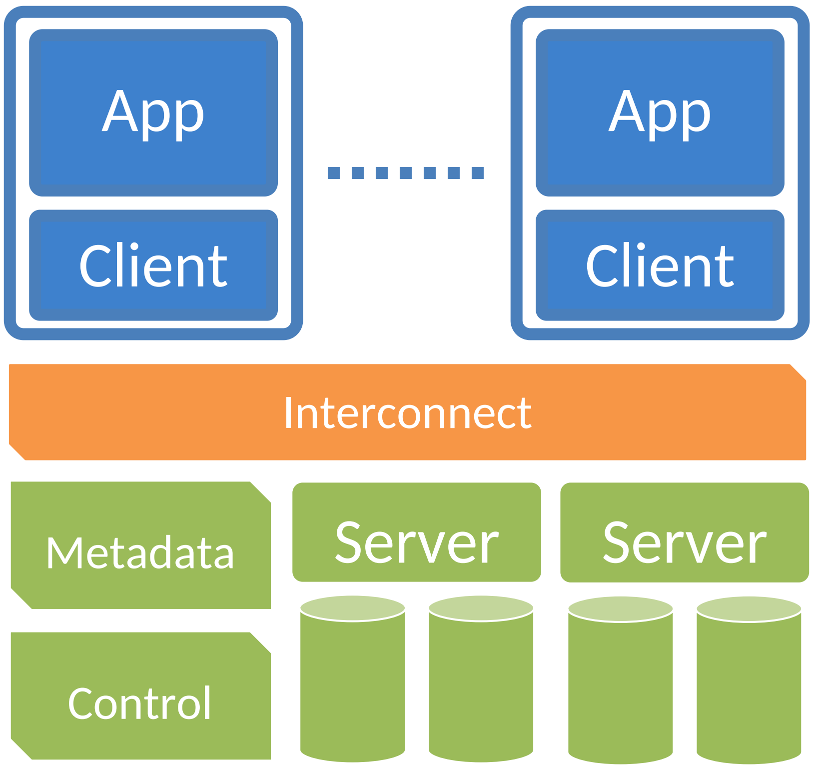


FENIX RI

Part I

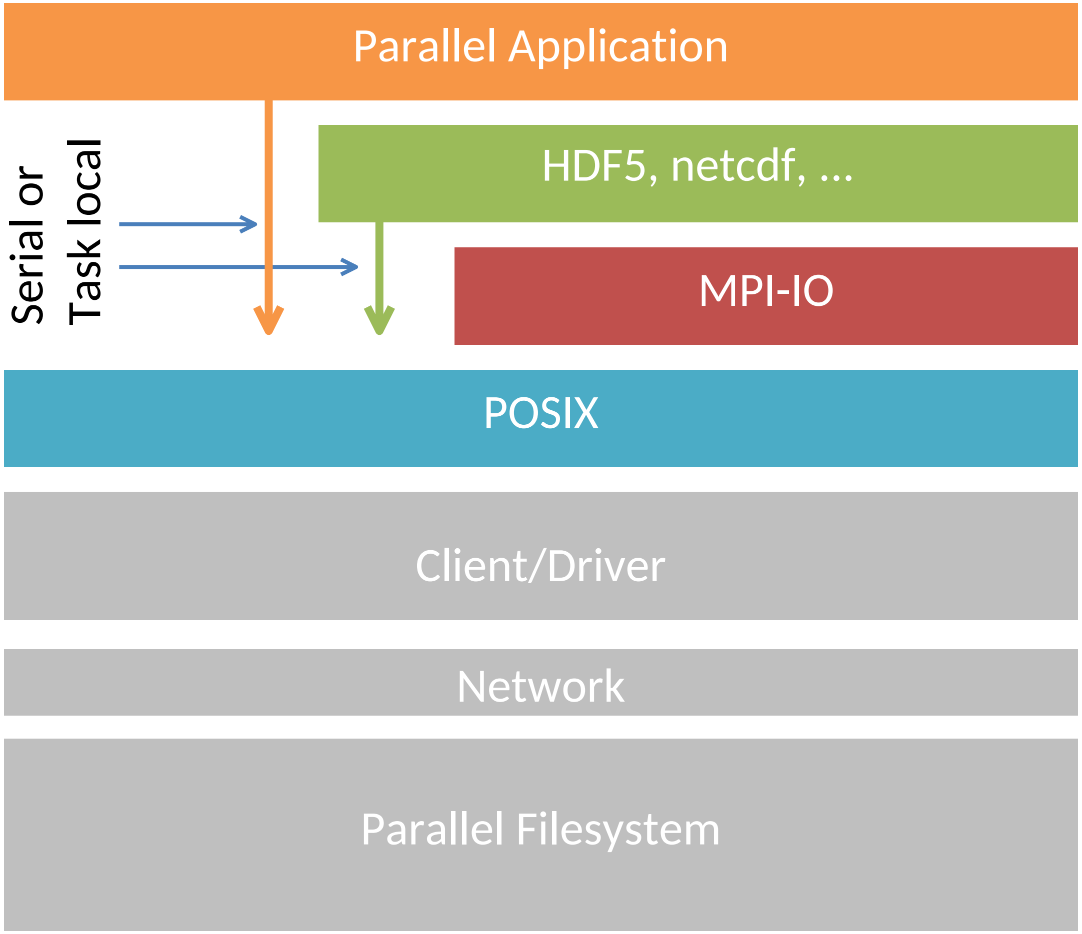
ACTIVE DATA REPOSITORIES

Typical Organisation



- Files are distributed in blocks
  - striping for performance
  - RAID for resilience
- Global metadata
  - directory contents
  - Size, RWM times, ...
  - can be a source of contention
- Consistency model options
  - relaxed: not POSIX
  - using locks: scalability issues
  - various trade-offs in between

Methods of Access



High-level libraries

- can expose MPI-IO

Low-level libraries

- pFS-aware
- tuning knobs

File-level access

- not pFS-aware
- same for serial I/O

User Interface

- POSIX: `open, write, read, close, ...`
  - Files: byte-level access
  - Directories: group files and other directories
- Strong guarantees on Ordering and Visibility
  - eg Read after Write
  - hold in multi-process environment
  - especially: exclusive ownership
  - hard to implement performantly
- Reachable inside a site
- Access control: owner/group, permissions

Do's and Don'ts

Common Pitfalls

- Many small files
  - metadata
  - false sharing
- Concurrent use of files
  - false sharing
  - locks
- Random access
  - caches
  - pre-fetching
- Redundant accesses
  - bandwidth

Optimised Use

- A few, **not** one, files
  - one per node
  - at least block sized
- Contiguous access
  - aligned to FS blocks
  - block granularity
  - exclusive to a process
- Reduce operation count
  - read once, broadcast
  - coalesce writes
  - especially within nodes

```python
import numpy as np
import h5py as h5
from mpi4py import MPI

# Setup parameters
N = 4
comm = MPI.COMM_WORLD
size, rank = comm.size, comm.rank

# Generate a block of data containing our rank
data = np.zeros((N,), dtype=np.float32) + rank

# Create a pHDF5 file and a chunked dataset inside
fd = h5.File("ranks.h5", "w", driver="mpio", comm=comm)
fd.create_dataset("R", dtype=np.float32,
                  shape=(N*size,), chunks=(N,),)

# All tasks coordinate, each writes one block
with fd["R"].collective:
    fd["R"][N*rank : N*(rank + 1)] = data[:]
```

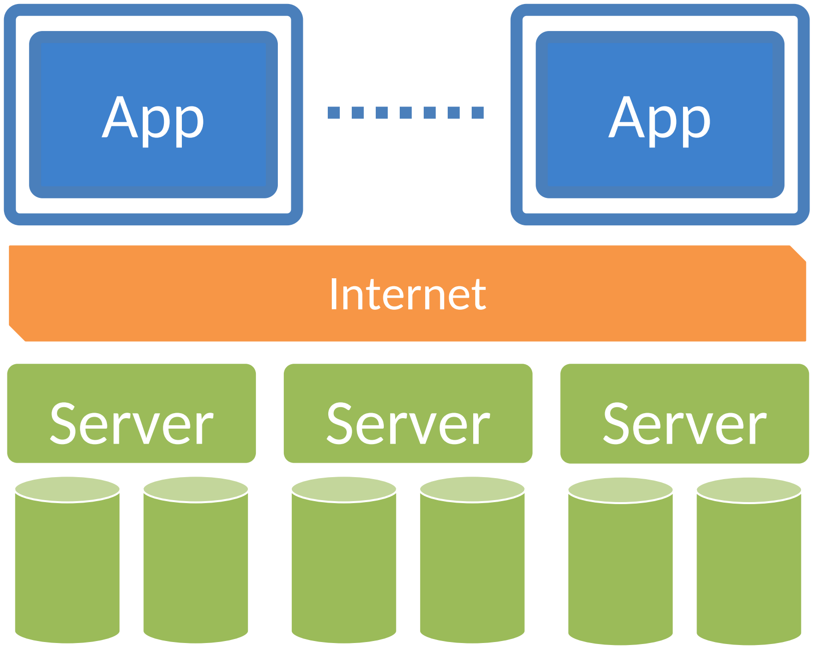
```
$ mpirun -n 4 python3 ranks.py
$ h5dump ranks.h5
HDF5 "ranks.h5" {
  GROUP "/" {
   DATASET "R" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 16 ) / ( 16 ) }
      DATA {
      (0): 0, 0, 0, 0,
           1, 1, 1, 1,
           2, 2, 2, 2,
           3, 3, 3, 3
      }
    }
  }
}
```

FENIX RI

Part II

ARCHIVAL DATA REPOSITORIES

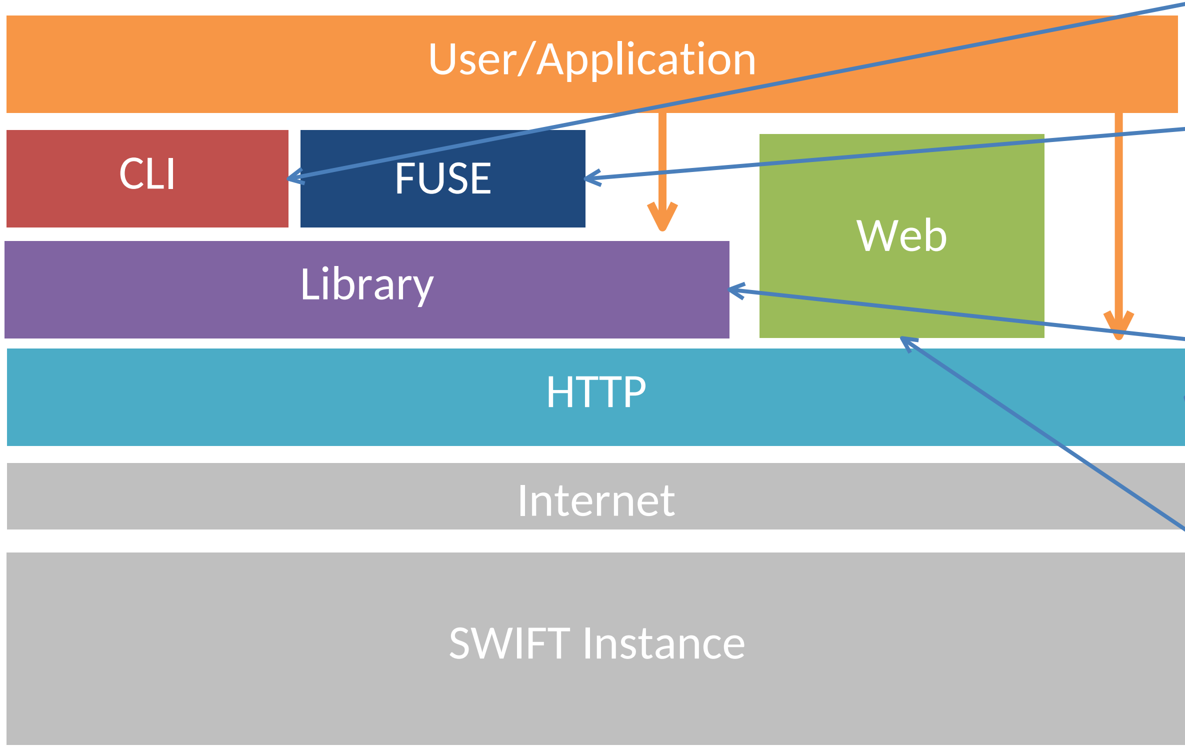
Typical Organisation



- Data model: abstract objects
  - arbitrary size
  - metadata inline
  - resilient storage
  - (optionally) content addressable
- Atomic operations
  - all-or-nothing
  - no ordering guarantees
- Performance
  - (usually) slower than pFS
  - high latency/low IOp/s
  - HTTP overheads
  - not a HPC network

FENIX RI

Methods of Access



Official client

FS on SWIFT
- works well for RO
- may break POSIX

`swiftclient`
- Python
- parallel
- advanced features

Raw REST

Browser as UI

Firewalls

FENIX RI

User Interface

- HTTP REST
  - Objects: atomic units of data
  - Containers: group objects
  - Flat hierarchy
- Atomic operations on objects
  - no guarantees on order
  - no byte-granular access
- Exposed on a public endpoint
  - access protected by authentication
  - traffic encrypted
- Access Control Lists

Introduction to REST

REpresentational State Transfer

- Pattern for Web API design
- Client-Server models
- Stateless, queries contain all context
- HTTP verbs express operations
- Resources identified by URL
- Atomic transitions between states

Verbs

- GET: Read content/List items
- HEAD: Get partial information, headers
- PUT/POST: Write to location
- DELETE: Remove item
- Less common: PATCH, TRACE, ...

URL

https://endpoi.nt/project/account/container/obj/ect

Endpoint
SWIFT installation

User/Project
Identification

Container
group of objects

Object
atomic data unit

SWIFT Objects

Object: Any byte stream

- POST: Write metadata
- GET: Content + metadata
  - Can ask for most recent version (expensive)
  - Can use eTag for caching
- PUT: Write data
- HEAD: Retrieve Metadata
- DELETE: Remove object

Metadata

- Key-Value pairs of form X-Object-Meta-<name>
- Mime-Type
- Encoding
- Expire objects at/after
- Checksums

HTTP

Metadata

OpenStack

Content

```
$ curl https://endpoi.nt/project/account/container/obj/ect \
       -X GET                                              \
       -H "X-Auth-Token: $token"
HTTP/1.1 200 OK
Date: Thu, 16 Jan 2014 18:51:32 GMT
Accept-Ranges: bytes
Content-Length: 14
Last-Modified: Wed, 15 Jan 2014 16:41:49 GMT
Etag: 451e372e48e0f6b1114fa0724aa79fa1
X-Timestamp: 1389804109.39027
X-Object-Meta-Orig-Filename: goodbyeworld.txt
Content-Type: application/octet-stream
X-Trans-Id: tx8145a190241f4cf6b05f5-0052d82a34
X-Openstack-Request-Id: tx8145a190241f4cf6b05f5-0052d82a34
Goodbye World!
```

FENIX RI

SWIFT Containers

Containers

- POST: Write metadata
- GET: Metadata + List of objects
  - Paging/Sorting
  - Pseudo-directory operations
- PUT: Create container
- HEAD: Retrieve Metadata
- DELETE: Remove containers

Metadata

- Key-Value pairs of form `X-Container-Meta-<name>`
- Access control lists (ACLs)
- Quota
- Versioning
- Synchronisation

HTTP

Metadata

OpenStack

Content

```
$ curl https://endpoi.nt/project/account/container \
       -X GET                                       \
       -H "X-Auth-Token: $token"
HTTP/1.1 200 OK
Accept-Ranges: bytes
Date: Wed, 15 Jan 2014 16:57:35 GMT
Content-Length: 341
Content-Type: application/json; charset=utf-8
X-Container-Object-Count: 2
X-Container-Meta-Book: TomSawyer
X-Timestamp: 1389727543.65372
X-Container-Bytes-Used: 26
X-Trans-Id: tx26377fe5fab74869825d1-0052d6bdff
X-Openstack-Request-Id: tx26377fe5fab74869825d1-0052d6bdff
Chapter/1
Chapter/2
```

Do's and Don'ts

Common Pitfalls

- Incremental updates
  - redundant
  - consistency with other writers
- Small objects
  - Performance suffers
- Redundant operations
  - individual operations are slow
- Concurrent modifications
  - no locking, one writer wins

Optimised Use

- Use a few MB per object
  - see benchmarks later on
- Consider bundling data
  - HDF5, "tar", ...
- Investigate compression
  - HDF5, "zip", ...
- Cache RO objects
- Update via RMW cycle
  - GET, update, PUT
  - condense updates
- Consider versioning objects
- Leverage atomicity

```
# setup a virtual environment
$ python3 -mvenv swift
$ cd swift
$ source bin/activate

# install required software
$ pip install python-openstackclient lxml oauthlib \
              python-swiftclient python-heatclient
$ git clone https://github.com/eth-cscs/openstack.git

# authenticate against SWIFT
$ source openstack/cli/pollux.env
> User:     ******
> Password: ******

# upload to a new container `test`
$ swift upload test openstack

# check container
$ swift list test
... list of all files in `openstack` folder ...
```

```python
# Now we download the object back again, in python.
from swiftclient.service import SwiftService
from os import environ as env

# Setup SWIFT client
swift = SwiftService(options={'auth_version':  '3',
       'os_project_id': env['OS_PROJECT_ID'],
       'os_auth_url':   env['OS_AUTH_URL'],
       'os_auth_token': env['OS_TOKEN'],})

# List container and pull out names
lst = swift.list(container='test-new')
nms = [i['name'] for p in lst
                for i in p['listing']]

# Download one object into ByteStream and re-assemble
dwn = swift.download(container="test",
                     objects=["openstack/cli/pollux.env"],
                     options={"out_file": "-"})
obs = [b"".join(d["contents"]) for d in dwn]
```

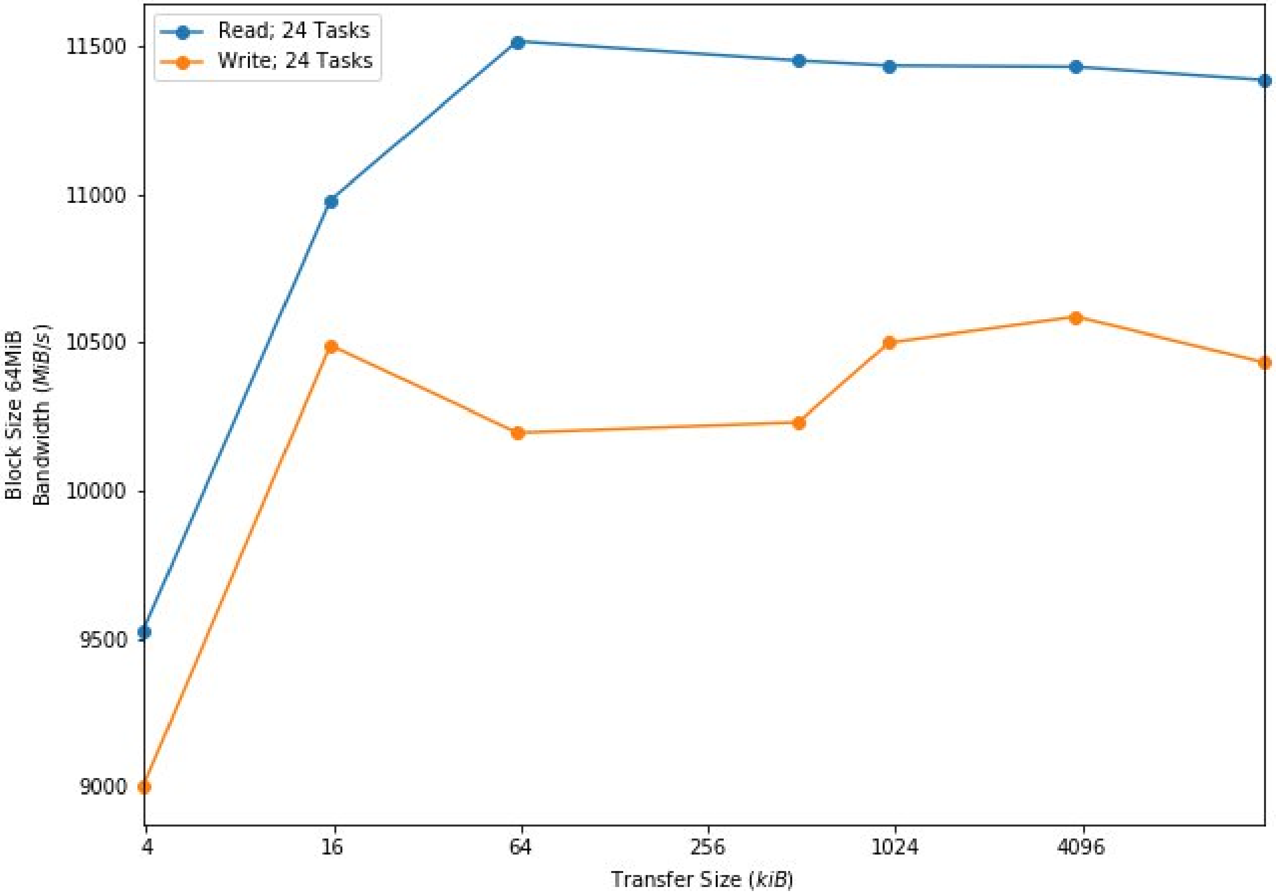
No errors checked!

FENIX RI

Evaluation

PERFORMANCE

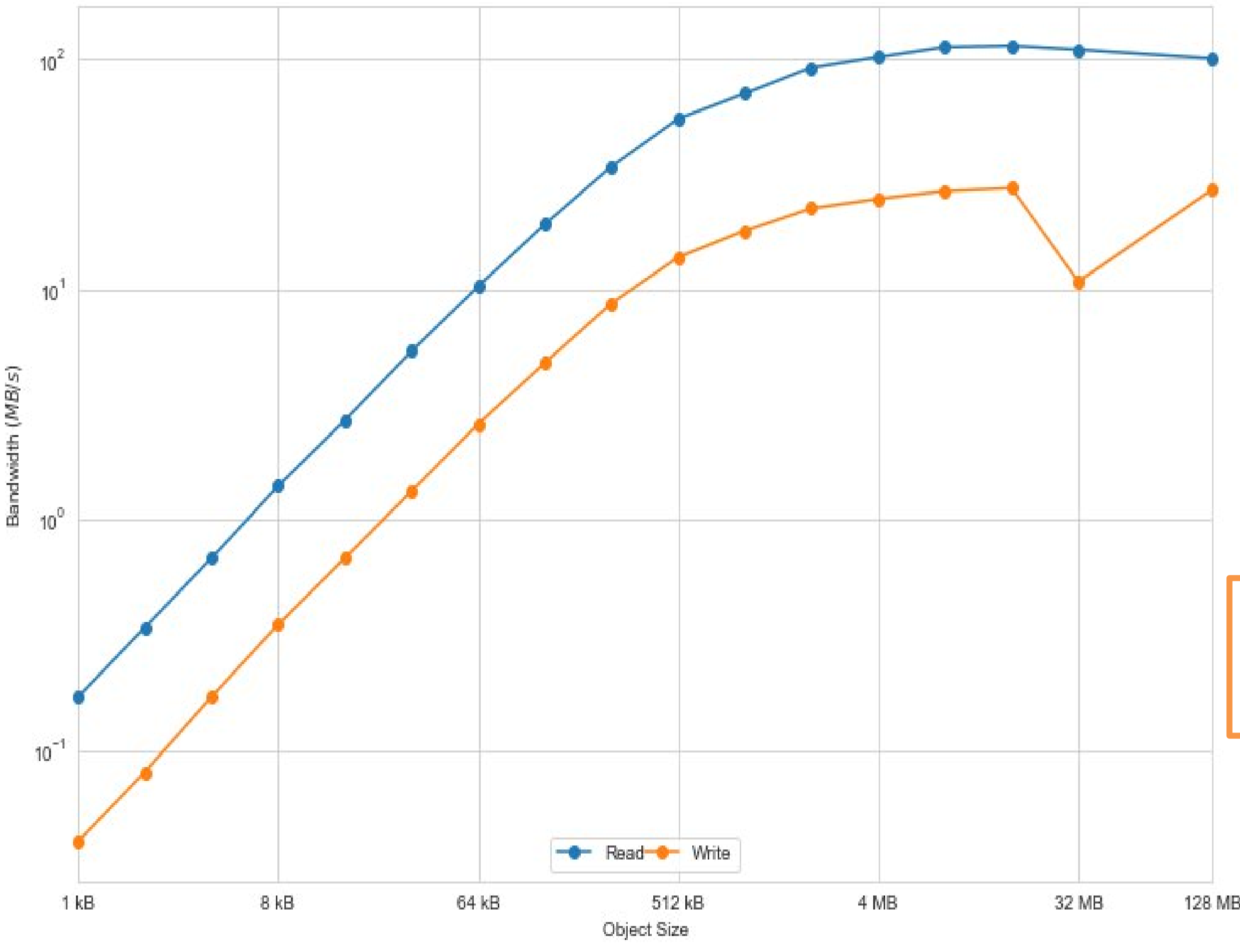
Performance Example: ACD



- IOR (3.3.0)
  - 1 file/task
  - no page cache
  - MPI-IO backend
- Juwels@JSC
  - GPFS
  - 4 nodes x 24 tasks
  - storage network per node
- Use >16k blocks

FENIX RI

Performance Example: ARD



- Cosbench (0.4.2)
- Operation mix
  - concurrent
  - 0.8R + 0.2W
- Juron@JSC
  - 1 node x 20 tasks
  - Pollux@CSCS
- Use >4M objects

Much more sensitive to block size than pFS

FENIX RI

Conclusion

KEY MESSAGES

Take-home Messages

- Two breeds of storage with unique strengths
  - HPC + Active Data Repositories (ACD)
  - Cloud + Archival Data Repositories (ARD)
- ACD for high-performance I/O on-site
  - Fast, byte-oriented
  - Strong guarantees for coordinated access
- ARD for long term, federated storage
  - Slower, object-based
  - Atomic put/get
  - Data can be made accessible as a plain http link

Learn More

- General information on SWIFT and I/O
  - CSCS Object Storage user.cscs.ch/storage/object_storage
  - HBP T7.2.3 I/O Guides wiki.humanbrainproject.eu/bin/view/Collabs/how-to-data-access-and-efficient-io
  - OpenStack SWIFT docs.openstack.org/swift/latest
- Talk to us
  - HBP Support support@humanbrainproject.eu
  - Fenix User Forum fenix-ri.eu/infrastructure/fenix-user-forum
- Get Access to FENIX fenix-ri.eu/access

Thanks!

QUESTIONS?

References

- ior github.com/LLNL/ior
- cosbench github.com/open-io/cosbench
- https://www.nextplatform.com/2017/09/11/whats-bad-posix-io/
- High Performance Parallel IO (2019; Prabhat, Koziol et al)

FENIX RI

POSIX Specification (2008)

- After a **write** to a regular file has successfully returned
  - Any successful **read** from each byte position in the file that was modified by that **write** shall return the data specified by the **write** for that position until such byte positions are again modified.
    Meaning: reads sync writes and are ordered relative to writes
  - Any subsequent successful **write** to the same byte position in the file shall overwrite that file data.
    Meaning: Writes are ordered relative to each other.

Consequence
Parallel FS must either break POSIX compliance or very carefully perform synchronisation, which is expensive.

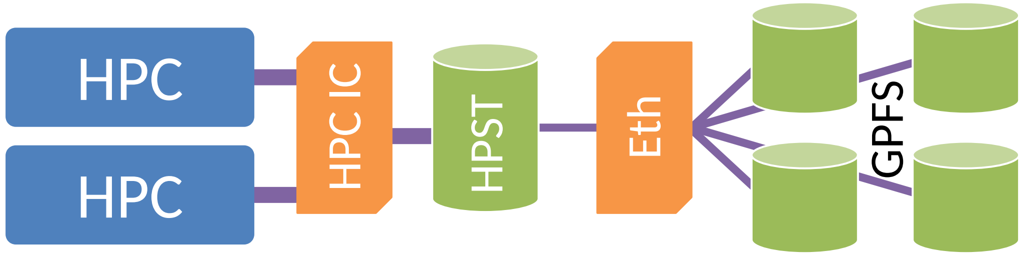
FENIX RI

High Performance Storage Tier at JSC

- NVMe-based storage cluster at JSC
  - Tied into HPC interconnect
  - Small capacity
  - High bandwidth
- Part of FENIX effort
- SLURM integration for Co-Scheduling

Usecases

- Burst Buffer
- Check pointing
- Pre-staging
- Data processing campaigns
- Live visualization
- Complex workflows



FENIX RI