paradigm4

Using flexFS™ for cloud-native, high-throughput, cost-effective HDF data volumes

HUG - October 14, 2021

Speaker Introduction

Gary Planthaber

- V.P. Advanced Product Development @ Paradigm4
- Responsible for flexFS™ and general innovation
- Email: gplanthaber@paradigm4.com

flexFS™ Introduction

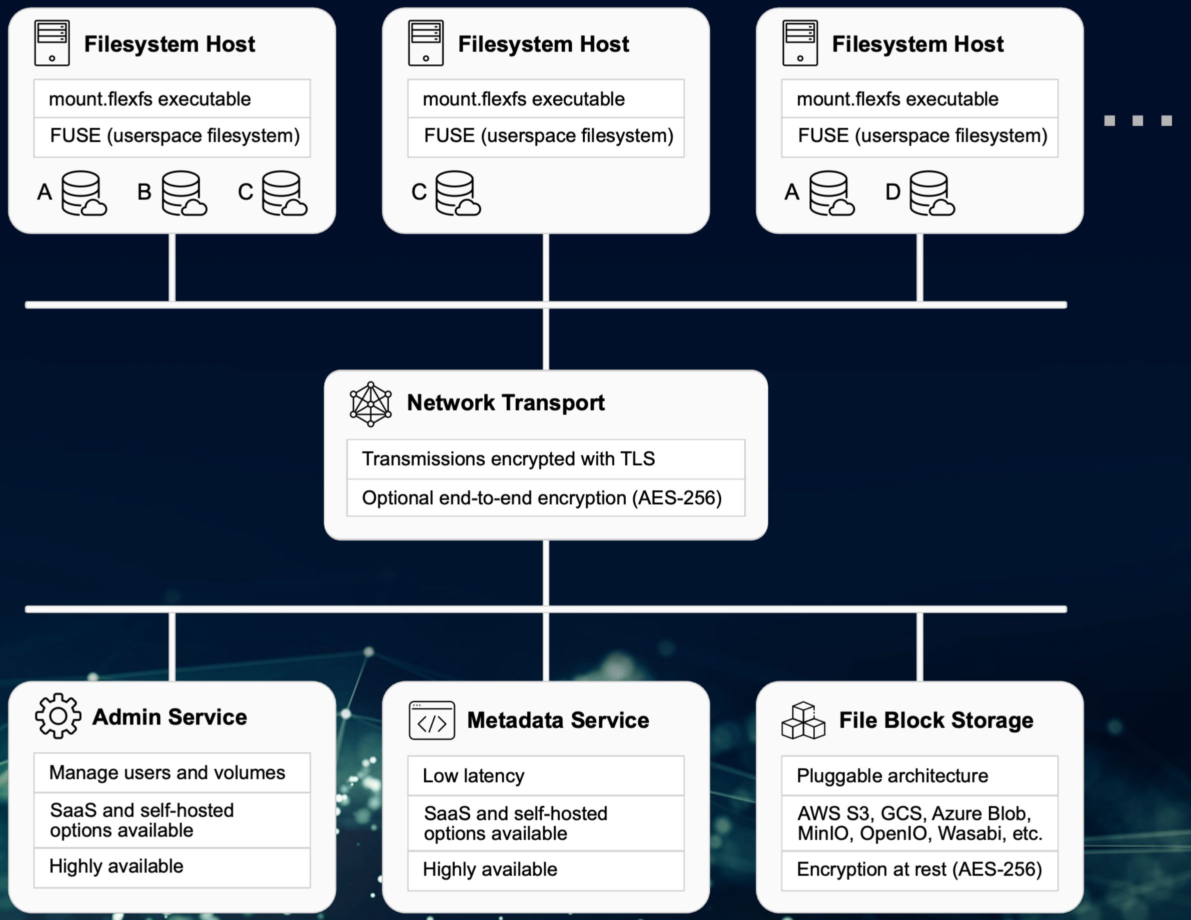
- What is flexFS™?
  - A POSIX-complaint network file system that was built from the ground-up to take full advantage of public cloud object stores (e.g., Amazon S3) to maximize aggregate throughput and minimize total cost of ownership (TCO)
- Who is flexFS™ intended for?
  - Public-cloud users of who:
    - Want an alternative to cloud filesystems (e.g., Amazon EFS)
    - Wish object stores (e.g., Amazon S3) were proper file systems
    - Want to avoid costly R&D projects to code applications to use object stores directly
- Who was flexFS™ NOT intended for?
  - Private-cloud users who:
    - Already have high-performance hardware dedicated to a clustered file system
    - Do not have a general-purpose block storage system (e.g., OpenIO)

flexFS™ Overview

- Scalable and easy to use
  - Trivial to mount shared read / write volumes on Linux hosts
  - Volumes appear and act as normal folders and enforce Linux permissions (even xACLs)
  - Each volume can be shared by 1000s of hosts concurrently
- Core strengths
  - Cheaper, higher throughput, and more secure than alternatives such as Amazon EFS
  - Cloud agnostic: supports portability and multi-cloud solutions
  - Similar or better performance versus using object stores directly
- What it's good for
  - Turning object stores (e.g., S3) into proper POSIX filesystems
  - Streaming data to computing clusters for analytics and machine-learning
  - Data staging and data migration workflows

flexFS™ Architecture

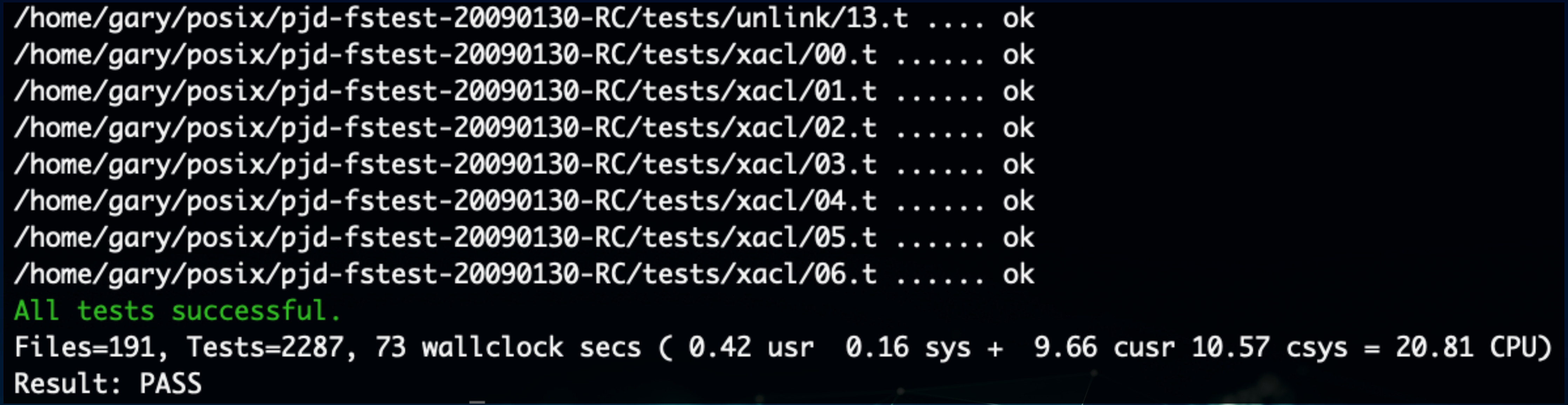
- Low-latency metadata service
  - Typically, 1-3 servers
  - Supports file locking
  - ACID-compliant
- High-throughput file block storage
  - Typically, object stores (e.g., S3)
  - Pluggable for flexibility
- Mounts communicate directly with file block storage to minimize infrastructure and bottlenecks



Filesystem Host
mount.flexfs executable
FUSE (userspace filesystem)
A
B
C
Filesystem Host
mount.flexfs executable
FUSE (userspace filesystem)
C
Filesystem Host
mount.flexfs executable
FUSE (userspace filesystem)
A
D
Network Transport
Transmissions encrypted with TLS
Optional end-to-end encryption (AES-256)
Admin Service
Manage users and volumes
SaaS and self-hosted options available
Highly available
Metadata Service
Low latency
SaaS and self-hosted options available
Highly available
File Block Storage
Pluggable architecture
AWS S3, GCS, Azure Blob, MinIO, OpenIO, Wasabi, etc.
Encryption at rest (AES-256)

flexFS™ POSIX Compliance

- Support for soft links, hard links, advisory file locking, extended ACLs, etc.
- Use your favorite tools and utilities (find, grep, cat, less, etc.)
- Passes extensive Tuxera POSIX test suite

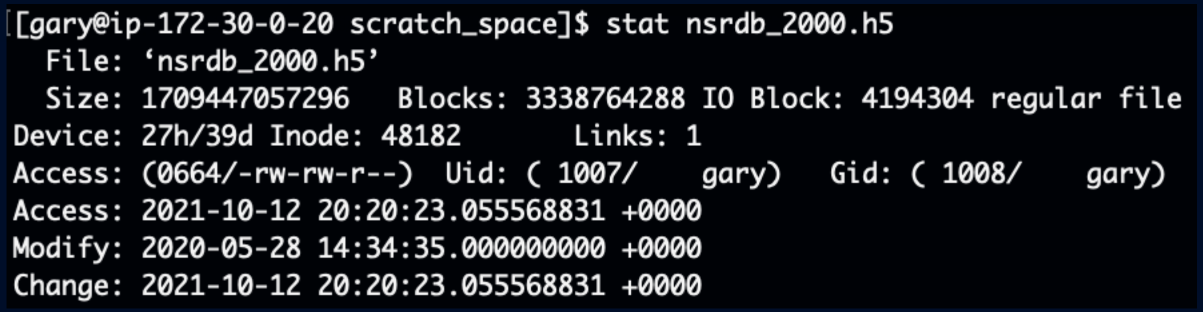


```
/home/gary/posix/pjd-fstest-20090130-RC/tests/unlink/13.t .... ok
/home/gary/posix/pjd-fstest-20090130-RC/tests/xacl/00.t ...... ok
/home/gary/posix/pjd-fstest-20090130-RC/tests/xacl/01.t ...... ok
/home/gary/posix/pjd-fstest-20090130-RC/tests/xacl/02.t ...... ok
/home/gary/posix/pjd-fstest-20090130-RC/tests/xacl/03.t ...... ok
/home/gary/posix/pjd-fstest-20090130-RC/tests/xacl/04.t ...... ok
/home/gary/posix/pjd-fstest-20090130-RC/tests/xacl/05.t ...... ok
/home/gary/posix/pjd-fstest-20090130-RC/tests/xacl/06.t ...... ok
All tests successful.
Files=191, Tests=2287, 73 wallclock secs ( 0.42 usr  0.16 sys +  9.66 cusr 10.57 csys = 20.81 CPU)
Result: PASS
```

flexFS™ Compression

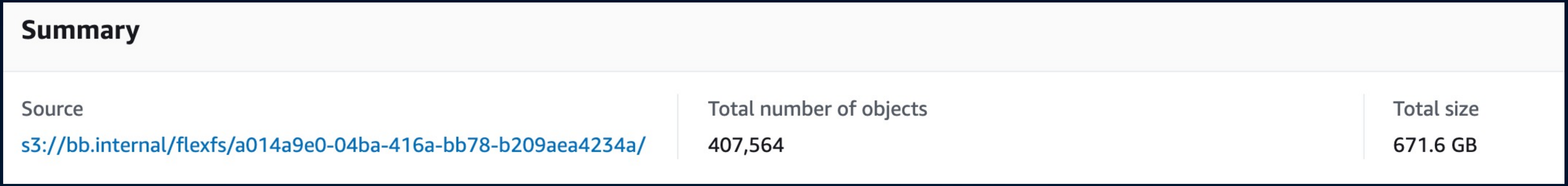
- All data blocks are compressed
- Sparse blocks are not materialized
- Object store usage is minimized
- Storage costs are reduced

Example: 1592 GiB of NREL NSRDB raw file data



```
[gary@ip-172-30-0-20 scratch_space]$ stat nsrdb_2000.h5
  File: ‘nsrdb_2000.h5’
  Size: 1709447057296   Blocks: 3338764288 IO Block: 4194304 regular file
Device: 27h/39d Inode: 48182       Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1007/    gary)   Gid: ( 1008/    gary)
Access: 2021-10-12 20:20:23.055568831 +0000
Modify: 2020-05-28 14:34:35.000000000 +0000
Change: 2021-10-12 20:20:23.055568831 +0000
```

Is losslessly persisted as 671.6 GiB of data in S3



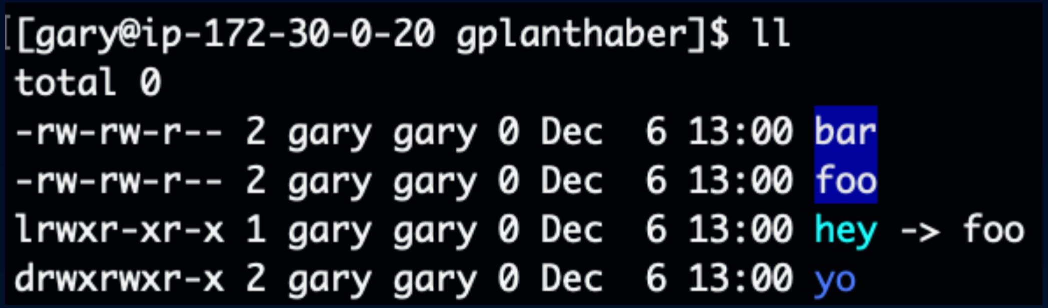
Summary

Source	Total number of objects	Total size
s3://bb.internal/flexfs/a014a9e0-04ba-416a-bb78-b209aea4234a/	407,564	671.6 GB

flexFS™ Privacy & Security

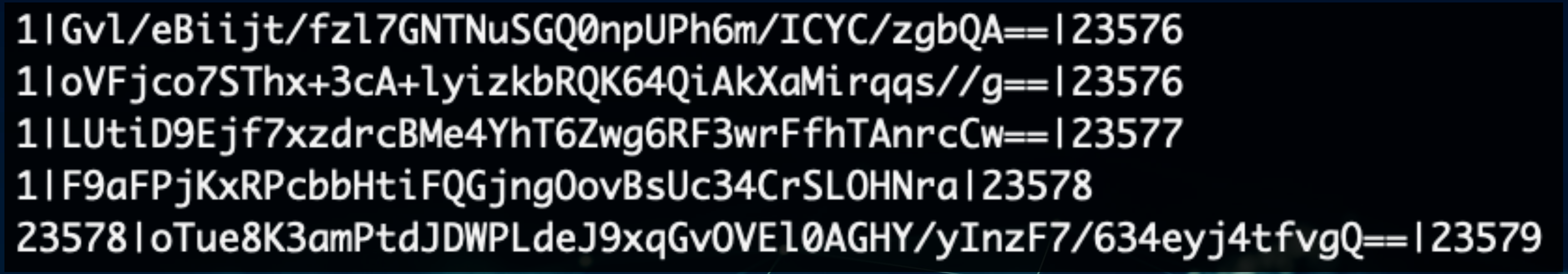
- Always-on encryption during transit (TLS) and at rest (AES-256)
- Optional end-to-end (AES-256) encryption layer

What the client sees



```
[gary@ip-172-30-0-20 gplanthaber]$ ll
total 0
-rw-rw-r-- 2 gary gary 0 Dec  6 13:00 bar
-rw-rw-r-- 2 gary gary 0 Dec  6 13:00 foo
lrwxr-xr-x 1 gary gary 0 Dec  6 13:00 hey -> foo
drwxrwxr-x 2 gary gary 0 Dec  6 13:00 yo
```

What the metadata service sees



```
1|Gvl/eBiijt/fzl7GNTNuSGQ0npUPh6m/ICYC/zgbQA==|23576
1|oVFjco7SThx+3cA+lyizkbRQK64QiAkXaMirqqs//g==|23576
1|LUtiD9Ejf7xzdrcBMe4YhT6Zwg6RF3wrFfhTAnrcCw==|23577
1|F9aFPjKxRPcbbHtiFQGjngOovBsUc34CrSLOHNra|23578
23578|oTue8K3amPtdJDWPLdeJ9xqGvOVEl0AGHY/yInzF7/634eyj4tfvgQ==|23579
```

flexFS™ Limits

The following are some theoretical limits for flexFS™:

- Maximum number of inodes = 2^63 - 1
- Maximum file size = (2^63 + 1) * blocksize
- Maximum file system size = (2^63 - 1) * (2^63 + 1) * blocksize

flexFS™ Sequential I/O Benchmark

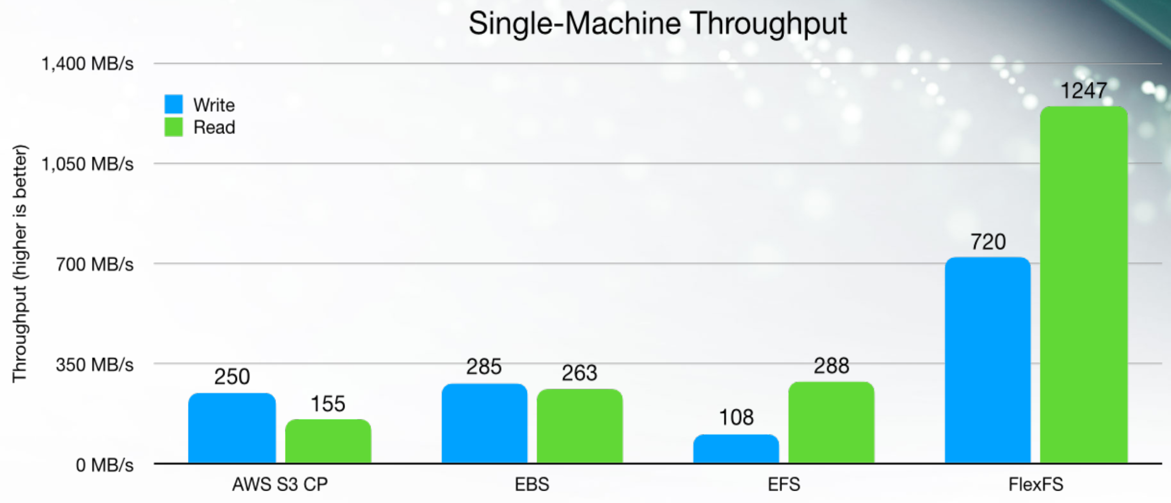
Disclaimer: This benchmark tests a NOMINAL case for flexFS™

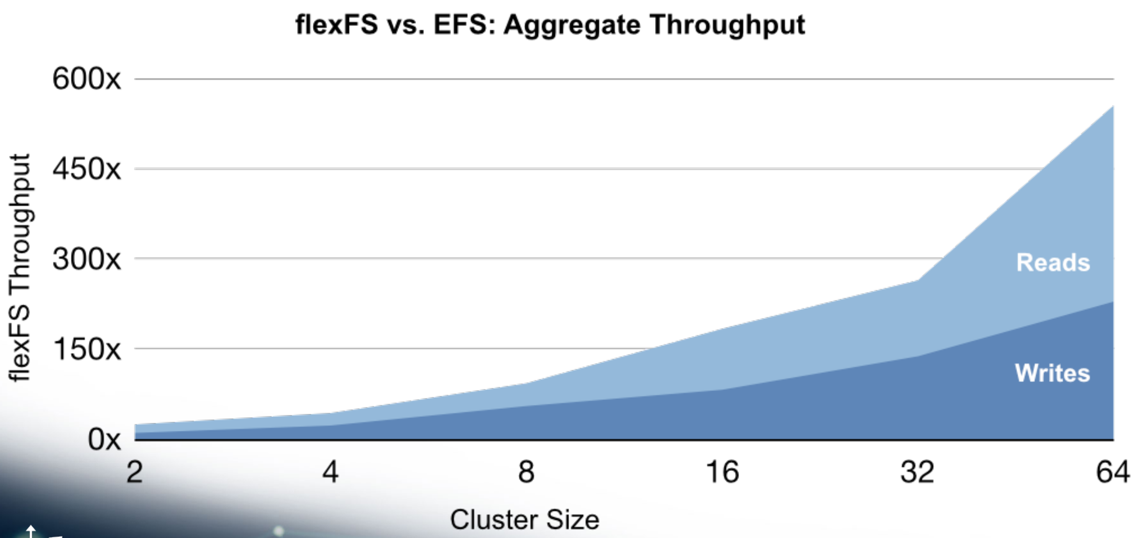
A large file was written from a ramdisk into S3 using ”aws s3 cp”, and into EBS, EFS, and flexFS using ”dd”. After clearing all caches, the same file was read back from S3 using “aws s3 cp” and from EBS, EFS, and flexFS using “dd” into /dev/null.

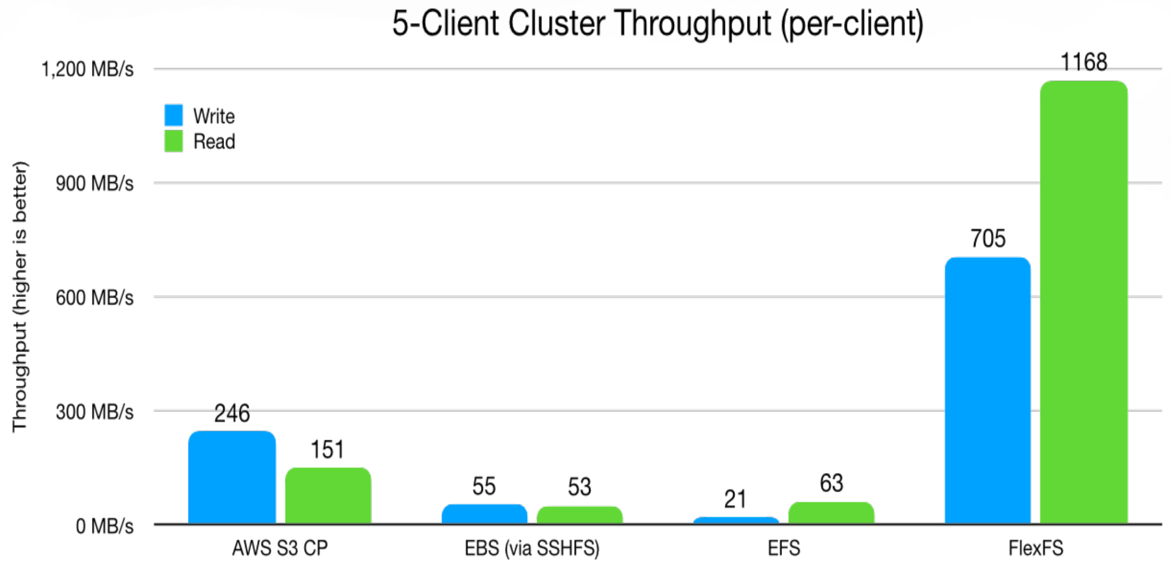
- Instance type: c5n.18xlarge (72 vCPUs, 192 GB RAM, 100 Gbps network)
- EBS volume type: gp2 (16,384 GiB provisioned, 16,000 IOPS)
- EFS volume: default parameters, starting empty
- flexFS volume: default parameters, starting empty
- File Size: 128,830,500,549 bytes (120.0 GiB / 128.8 GB)

flexFS™ Sequential I/O Benchmark

- flexFS aggregate throughput scales near-linearly with cluster size
- EFS and EBS aggregate throughput remains effectively constant







flexFS™ Random I/O Benchmark

Disclaimer: This benchmark tests a WORST case for flexFS™

The same NREL NSRDB file and Python benchmark script used by John Readey in his HSDS benchmarks were run against flexFS for comparison.

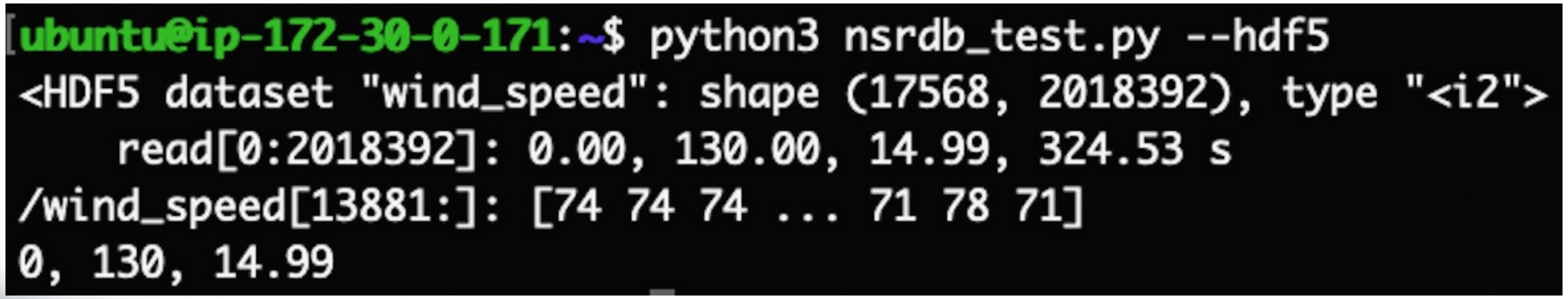
s3://nrel-pds-nsrdb/v3/nsrdb_2000.h5

https://github.com/HDFGroup/hsds/blob/master/tests/perf/nsrdb/nsrdb_test.py

- Instance type: m5.8xlarge (32 vCPUs, 128 GB RAM, 10 Gbps network)
- File Size: 1,709,447,057,296 bytes (1.6 TiB / 1.7 TB)

flexFS™ Random I/O Benchmark

- Raw h5py + flexFS (325 s) performs almost identically to h5py + ros3 (328 s)
- Why? Both flexFS and ros3 must perform around the same number of (sequential) fetches from S3, with high latency
- If h5py could parallelize these fetches (like HSDS does), then performance would increase dramatically



```
ubuntu@ip-172-30-0-171:~$ python3 nsrdb_test.py --hdf5
<HDF5 dataset "wind_speed": shape (17568, 2018392), type "<i2">
    read[0:2018392]: 0.00, 130.00, 14.99, 324.53 s
/wind_speed[13881:]: [74 74 74 ... 71 78 71]
0, 130, 14.99
```

flexFS™ Observations (on AWS)

- Low metadata overhead
  - Metadata overhead is typically ~ 1% of S3 latency
  - i.e., the flexFS™ file system is a “low-cost” abstraction over S3
- Compression reduces latency
  - Experiments show that the compression has a slightly positive reduction on S3 latency
  - i.e., flexFS™ is often faster than using S3 directly with uncompressed objects
- First-class write coordination
  - flexFS only updates blocks that are impacted by writes and provides locking to coordinate
- Performance bounds relative to HDF5 tools
  - Best case: access patterns are sequential and flexFS™ rivals HSDS performance
  - Worst case: access patterns are random and flexFS™ rivals ros3 performance

Questions