Storage Technologies

Lasair Cycle-1 Technology Review

18 March 2020

Problem Statement

- Each alert comes with 3 cutout images that need to be stored in an archive of some kind
- If we're going to store light curves as single objects, then the requirements are very similar to images
- Objective: identify the requirements, and evaluate candidate technologies, for a blob store

Use Cases

- Used to derive requirements:

1. Keep up with the alert stream
2. Retrieve a few images for display on a web page
3. Retrieve 1M items for analysis in a notebook or externally
4. Large scale (>> 10^6) data mining (stretch goal)

Requirements

- Write 15KB images at a rate of at least 900 per second and preferably 2000 per second.
- Write 500KB light curves at a rate of at least 300 per second and preferably 700 per second.
- Store 150TB per year and scale to an ultimate size of around 1.5PB.
- Read arbitrary items with low latency (<1s).
- Read items at a rate of at least 300 items per second.
- Robust service with low downtime.
- Recovery in the event of a failure.
- Minimise staff effort required for both development and maintenance.

Types of technology

- Conventional filesystems
- Distributed filesystems
- Object stores
- Relational databases
- NoSQL databases

Long list of technologies

- 1a. Single filesystem directly attached to ingest node
- 1b. Dedicated storage node with multiple filesystems
- 1c. Cluster of storage nodes
- 2a. CephFS (+ Manila?)
- 2b. HDFS (or similar)
- 3a. Ceph + Swift
- 3b. Ceph/RADOS directly
- 3b. MinIO
- 4a. Store as records in the object database
- 4b. Store as blobs in a (separate?) MySQL database
- 5a. MongoDB
- 5b. Cassandra

Rejected entirely

- 1a. Single filesystem directly attached to ingest node
  - Unlikely to meet requirements
- 1c. Cluster of storage nodes
  - Reinventing the wheel

Deferred

- 1b. Dedicated storage node with multiple filesystems
  - Could potentially meet requirements, but likely requires more effort for less benefit than a more off-the-shelf solution.
- 2b. HDFS (or similar)
  - Unlikely to make sense as a primary datastore and need for a secondary system for data mining not clear.
- 3a. Ceph/RADOS + Swift
  - Could look into as a possible performance enhancement of 3a.
- 3b.MinIO
  - High, and potentially duplicated, effort.
- 5a. MongoDB
  - High effort
- 5b. Cassandra
  - High effort

Not evaluated

- 4a. Store as records in the object database
  - Effectively the null option
- 4b. Store as blobs in a (separate?) MySQL database

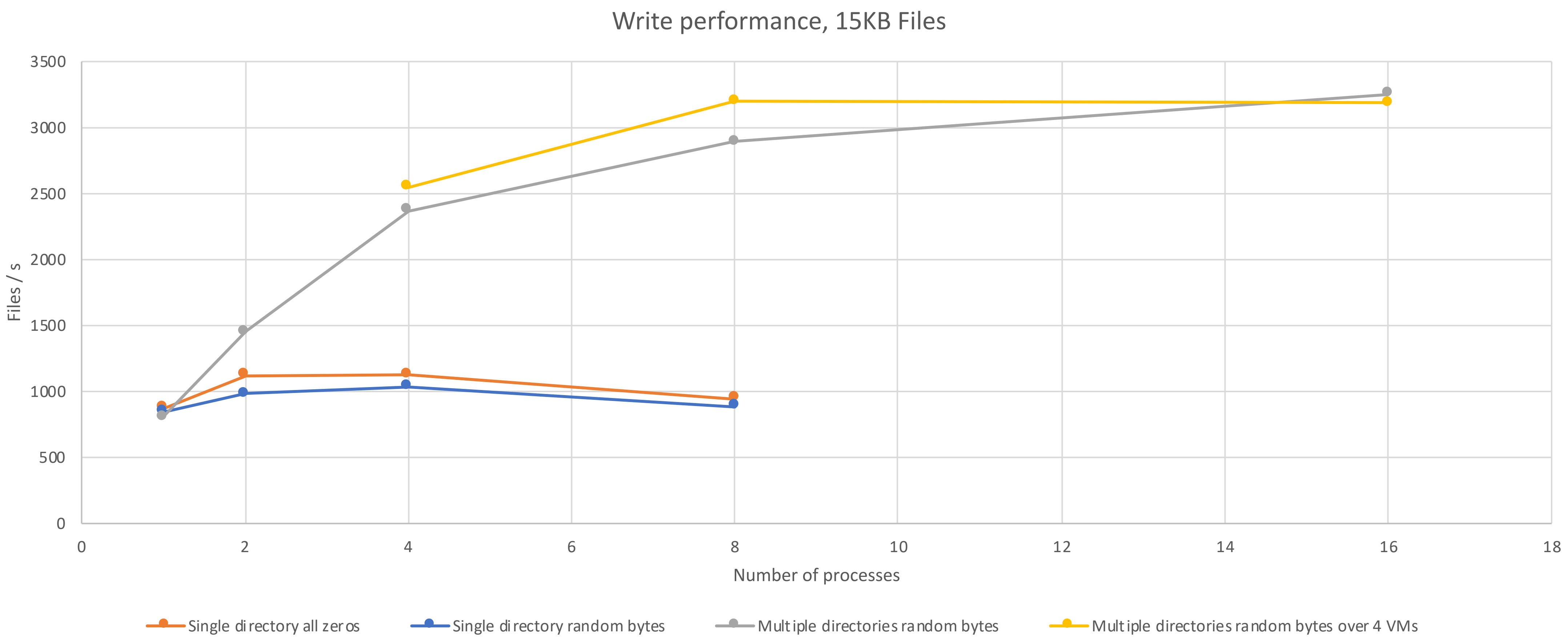
Shortlisted options

- 2a. CephFS
  - Likely to have some performance advantage over Swift at the cost of slightly higher, but still reasonable, admin effort.
- 3a. Ceph/Swift based Object Store
  - Looks like the lowest effort option. Should have excellent scalability and robustness. Still need some experiments to ascertain what sort of performance we can expect.

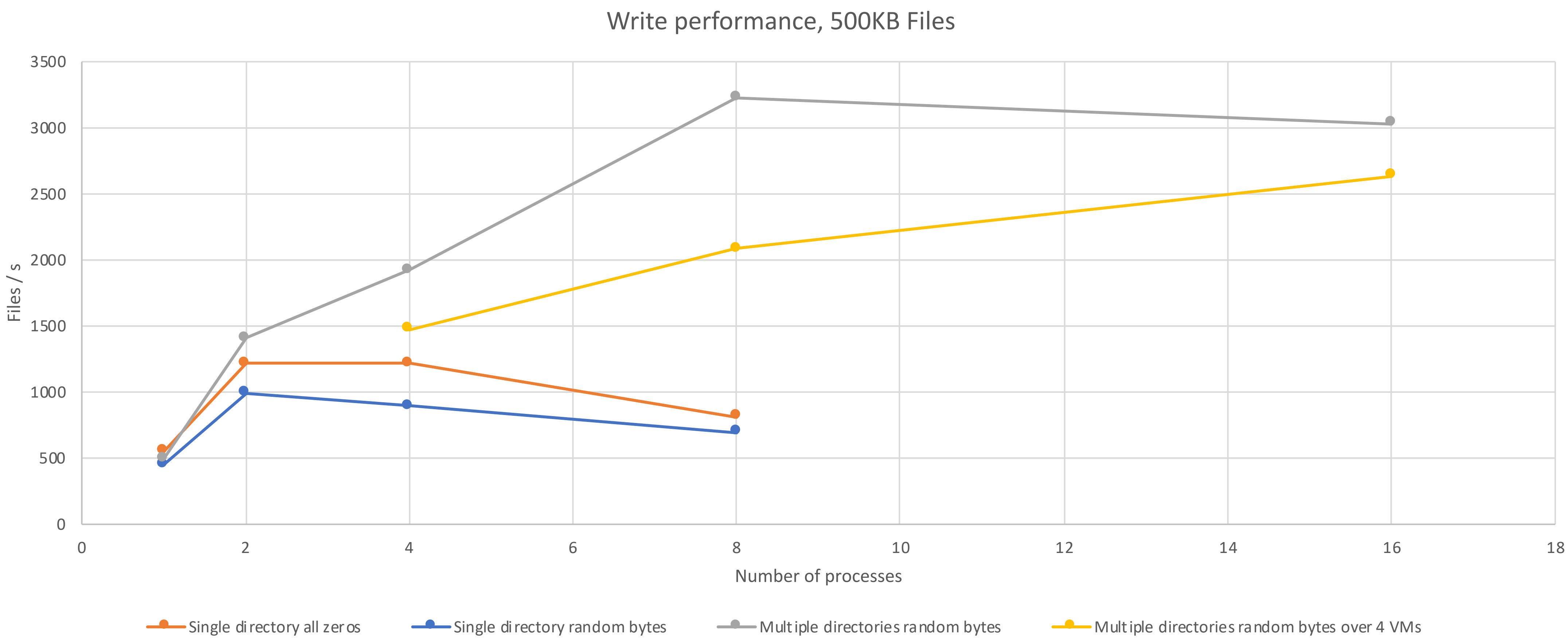
Experiments

- Set up test CephFS volume and Swift service on the OpenStack cluster (thanks to Mark and Teng)
- Performance benchmarking of CephFS
- Two proof of concept demos for Swift
- Swift is functional, but haven’t had time to fully benchmark
- Not getting good performance on initial tests – need to investigtate

CephFS: Writing small files



CephFS: Writing larger files

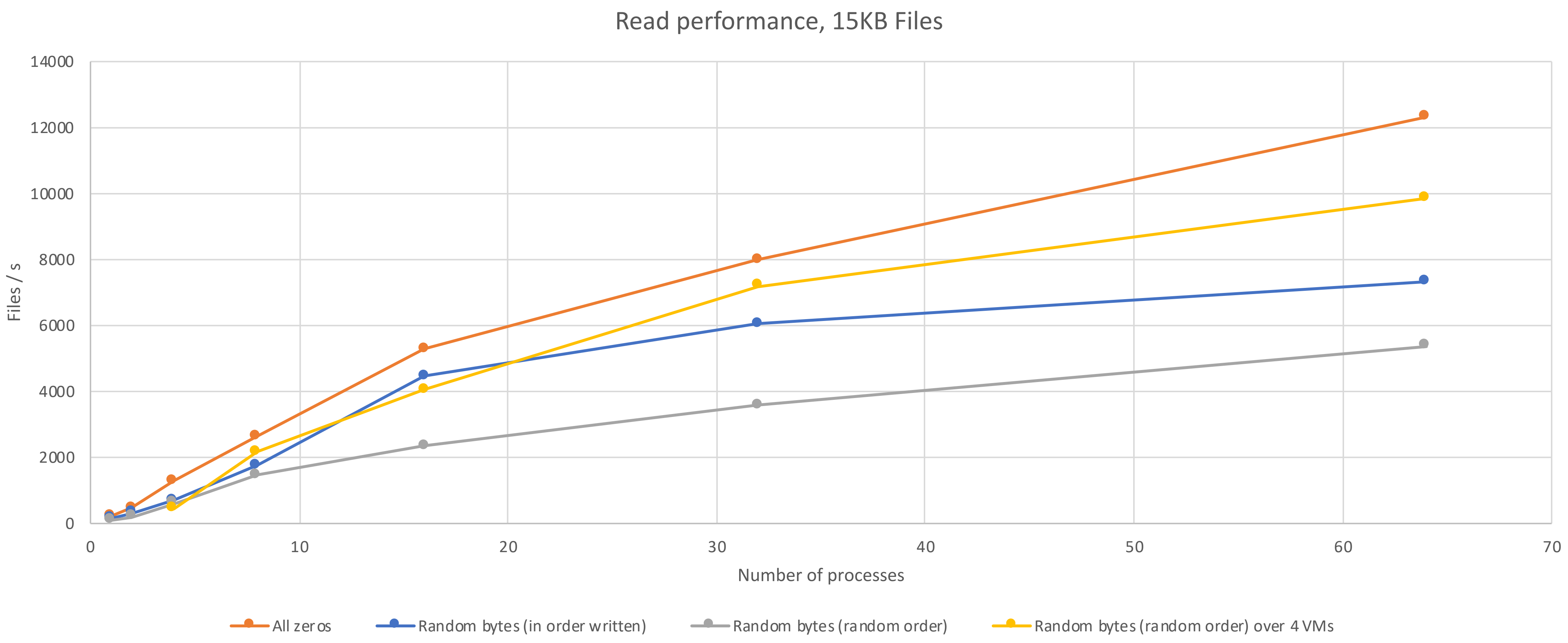


CephFS: Overwriting

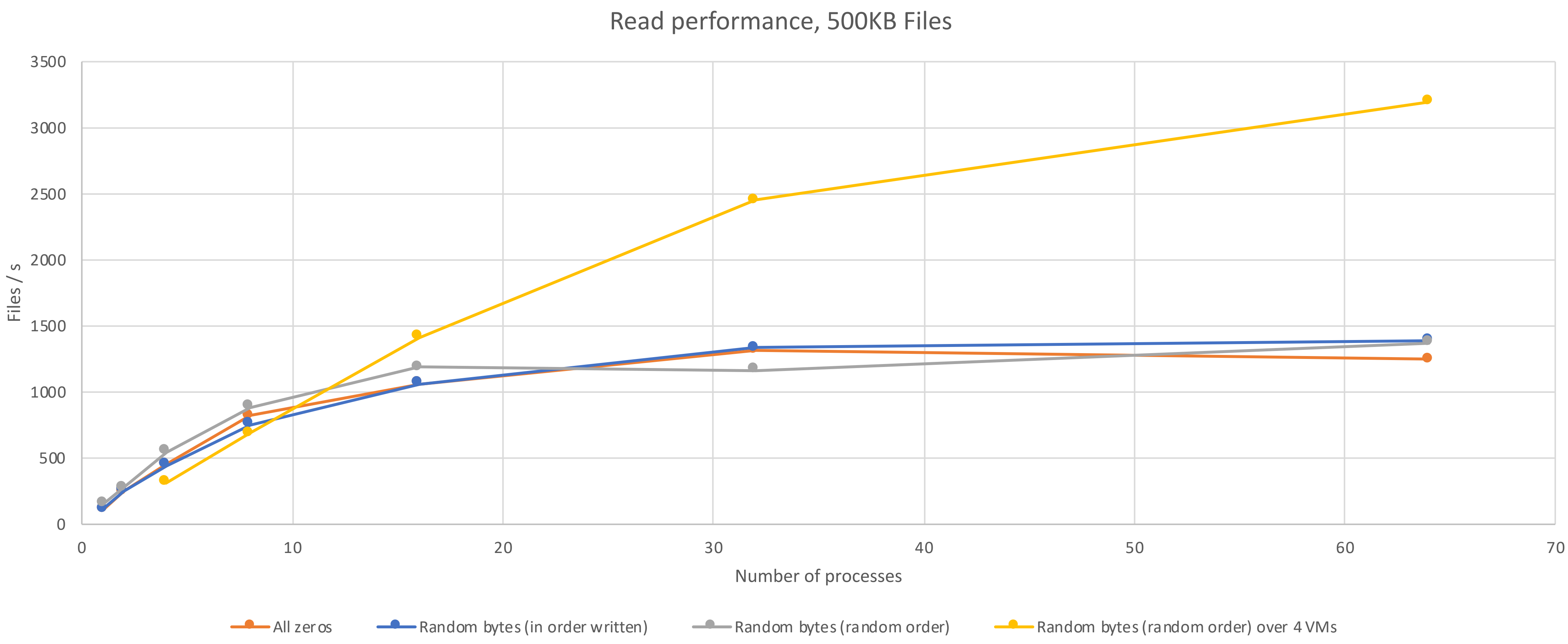
- Overwriting a file appears to be significantly slower than creating a new file. Deleting the existing file first as a separate operation actually appears to be quicker.

		Write			Overwrite			Delete followed by write		
Size (KB)	N files	Time (s)	Files/s	MB/s	Time (s)	Files/s	MB/s	Time (s)	Files/s	MB/s
15	33334	32.64	1021.24	15.31	171.37	194.51	2.91	75.37	442.26	6.63
100	5000	7.49	667.55	66.75	23.72	210.79	21.07	9.28	538.79	53.87
500	1000	1.79	558.65	279.32	5.19	192.67	96.33	2.24	446.42	223.21

CephFS: Reading small files



CephFS: Reading larger files



CephFS: Development and Admin

- Minimal work would be required to modify existing code
- Need to ensure FS gets mounted when deploying VMs
- Requires that all VMs use the same UIDs and GIDs
- Requires that we deploy a separate HTTP interface
- Filesystem has a fixed size, but can be expanded when required
- Could consider using OpenStack Manila to provision/manage

Swift

- Two proof of concept demonstrations
  - Query an archive of ZTF light curves stored as JSON files over an HTTP interface
  - Extract image files from a Kafka stream of alerts and writing them to a Swift object store
- Have not yet been able to do performance benchmarking
- Appears to have issues with performance in initial tests

Swift: Development and Admin

- Some code modification required, but very easy as we already have demo code from POC
- Uses Keystone for AAI so no requirement to deploy anything additional
- No fixed size – well suited to unbounded data
- Built-in HTTP interface for read access.

Key Points

- Swift looks slightly better – if it can be made to perform adequately
- CephFS looks like a good alternative if not and does perform well enough (although marginally so for upper write figure)
- Need lots of parallelism to get good throughput
- Some potential pitfalls to avoid

Open questions

- Lower figures for required performance based on 10 M alerts per night; upper figures based on 50x ZTF. Which one is correct?
- Do we need to go back and look at MySQL?
- Do we agree that Swift is preferable?
- How much more effort should we spend on this?