Blob Storage Technologies

Contents

Executive Summary

A study was undertaken to identify the key requirements for storing binary object like items in Lasair:LSST (cutout images and potentially light curves) and to shortlist preferred technology options for implementing such a store.

The key requirements are that such a data store must:

- Be able to store 15KB cutout images at a rate of at least 900 per second and preferably 2000 per second.
- Be able to store 500KB light curves at a rate of at least 300 per second and preferably 700 per second.
- Be able to store around 150TB per year and scale to an ultimate size of around 1.5PB.
- Allow the retrieval of arbitrary items with low latency (<1s).
- Allow the retrieval of items at a rate of at least 300 items per second.
- Allow for a robust service with low downtime.
- Allow for recovery in the event of a failure.
- As far as possible within these constraints, minimise staff effort required for both development and ongoing maintenance.

Two technology options were shortlisted: CephFS and Swift. Both technologies are based on the existing Ceph storage cluster and are designed to scale to multi-petabyte scale with a high degree of resilience.

CephFS

A CephFS filesystem using the existing Ceph storage cluster was tested with the following results:

- Read performance was well in excess of the minimum requirements provided sufficient parallelism is employed.
- Write performance was well in excess of the lower requirement for images, but only marginally better than the upper requirement – we should identify as soon as possible which of the two figures is actually applicable here.

Swift

Unfortunately our testing of Swift is not yet at the stage where we have comparable performance figures. Functionally it has been demonstrated to work and in terms of administration has some (minor) advantages over CephFS.

Objective

The objective of this work is to identify a preferred technology to store binary object like items in Lasair:LSST. These objects will definitely include the cutout images and potentially also light curves. There is no specific requirement that these two cases should use the same technology, although obviously it is desirable to minimise the number of different technologies and systems in use.

Note on terminology: since we are considering file systems, object stores and databases, which all use different terminology, and also the term object is already used to refer to an astronomical object, we have generally used the term "item" as a generic term for "file, object or blob".

Requirements

Types of item

Cutout images are small (around 15KB for ZTF and expected to be similar for LSST), there are three per alert, they are unique to the alert and never need to be updated or changed.

Light curves are potentially much larger (up to 500KB) and since they map to the object rather than the candidate most operations will be updates.

Ingest rates

The selected technology must be able to store images/lightcurves as fast as they are produced. It is the average rate over minutes to hours that matters – we don't necessarily need to worry about matching brief bursts of higher alert rates, but we do want images (and light curves) available in a reasonable time. We also need to have enough "slack" in the system that it can catch up if interrupted for some reason.

For the LSST alert stream we anticipate[i] that there will be around 10 million alerts per night. This implies around 1 million per hour or around 300 per second.

We have also been using the "50 times more alerts than ZTF" measure. *[Where does this come from?]* On this basis, the maximum number of ZTF alerts in a night is around 500000, which implies maximum rates for LSST (averaged over one night) of around 2.5 million per hour or 700 per second.

This implies target write rates of at least 900 items/s (13 MB/s) up to 2100 items/s (31 MB/s) for images; 300 items/s (150 MB/s) up to 700 item/s (350 MB/s) for light curves.

Requirement: for images, a minimum write rate for 15KB items of at least 900 items/s and preferably in excess of 2000 items/s.

Requirement: for light curves, a minimum write rate for 500KB items of at least 300 items/s and preferably in excess of 700 items/s.

Storage requirements

We expect to retain all cutout images generated in the course of the 10 year survey.

On the basis of 10M alerts per night we would expect around 10B images per year. ZTF has produced 135M images per year; 50 times this implies 7B images per year. This results in a storage requirement of 100 to 150 TB per year or 1 to 1.5 PB over the course of the 10 year survey.

Requirement: for images, the ability to store 150TB per year scaling to an ultimate size of around 1.5PB.

For light curves, we have obtained 1.5 million per year with ZTF. This implies around 75M in the first year with LSST occupying 35TB. Since light curves are typically updated on subsequent visits to the same object

it is not entirely clear how this number will change after the first year. As an upper bound, we assume that it remains the same in which case we get 750M over 10 years requiring 350TB.

Requirement: for light curves, the ability to store 35TB per year scaling to an ultimate size of around 350TB.

Retrieving Items

For reading items we considered the following use cases:

1. Retrieve a few images for display on a web page

- Given an identifier (presumably previously obtained from DB or stream) can retrieve using HTTP.
- Applies only to images.
- Requires reasonable latency (< 1s).

2. Retrieve up to order 1M items for analysis in a notebook or externally

- Given a list of identifiers (presumably previously obtained from DB or stream) can retrieve using some readily available API for processing in a notebook or externally.
- Would expect a small batch (~1K) to retrieve in at most a minute, and a larger batch (~1M) in order an hour.
- Applies to both images and light curves.
- We anticipate O100 active users for small queries of O100 items with the number of users decreasing with increasing size and complexity of query. We therefore don't anticipate a major issue with significant numbers of queries running concurrently, but it is necessary that any long running queries should not unduly interfere with much smaller ones.

3. Large scale (>> 10^6) data mining

- Applies to both light curves and images.
- Considered a stretch goal so no hard requirements, but forward compatibility with systems such as Spark may be something we want to consider.

Requirement: for images, the ability to retrieve any specific image with latency < 1s

Requirement: for images, the ability to retrieve at least 300 item/s (4.5MB/s) each by at least two queries in parallel.

Requirement: for light curves, the ability to retrieve at least 300 item/s (150MB/s) each by at least two queries in parallel.

Other considerations

Although these requirements are not easily quantifiable, they are at just as important for our ability to operate the service in the long term.

- Staff effort and specialised skills required, both to implement and maintain, must be kept as low as practical.
- The service must be robust, that is unlikely to fail, and any failure that does happen should be as easy as possible to detect and recover from.

Technologies

This is a summary of the analysis at https://lsst-uk.atlassian.net/wiki/spaces/LUSC/pages/885227561/Blob+Storage+Technology+Shortlist

Particular technologies do not exist in isolation and are considered here in the context of the existing OpenStack/Ceph cluster. Some options which might otherwise look attractive do not necessarily make sense if they require significant effort to reimplement functionality that already exists here.

Shortlisted Options

Two technology options were shortlisted to move forward to more detailed experiments. These were selected on the basis of having an attractive combination of robustness, scalability and low effort to implement and maintain in the context of the existing OpenStack environment, while still being likely to meet the minimum performance criteria.

2a. CephFS
3a. Ceph/Swift based Object Store

Deferred Options

These options we have decided not to progress with now (generally because they would require significantly more effort than the shortlisted options), but we could return to them if none of the shortlisted options end up being suitable or requirements change.

1b. Dedicated storage node with multiple filesystems
2b. HDFS (or similar)
3a. Ceph/RADOS + Swift
3b. MinIO
5a. MongoDB
5b. Cassandra

Eliminated Options

These options were considered and eliminated due to obviously failing to meet one or more requirements.

1a. Single filesystem directly attached to ingest node
1c. Cluster of storage nodes

Not Considered

Options included for completeness, but not considered due to lack of time/expertise.

4a. Store as records in the object database
4b. Store as blobs in a MySQL database

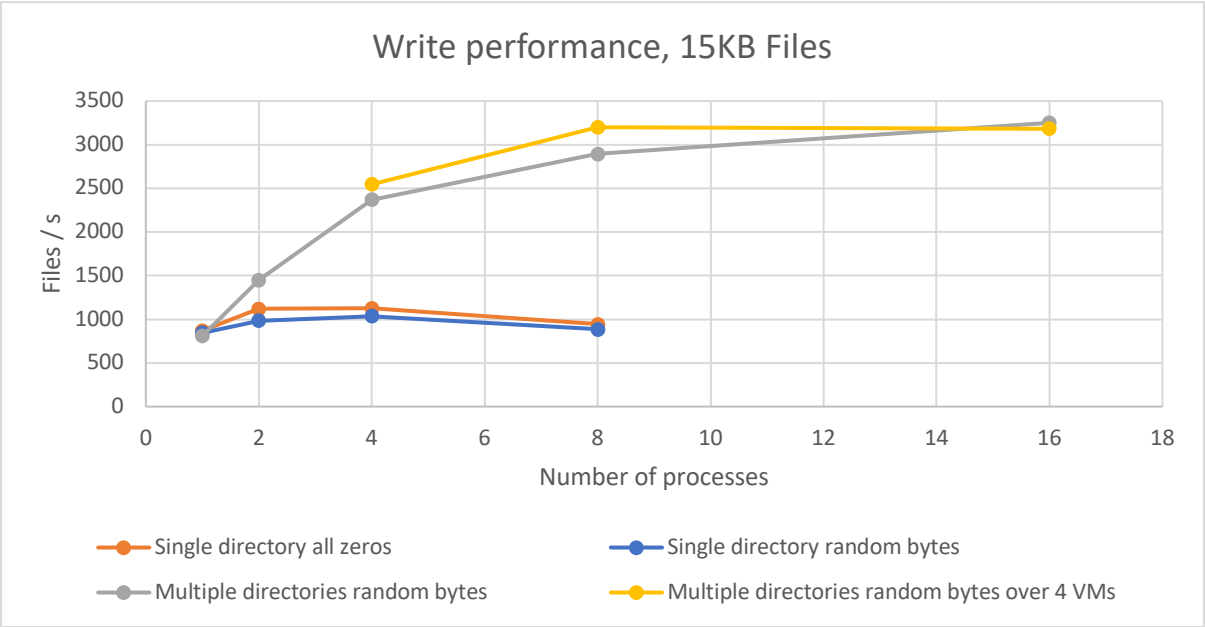
Experiments

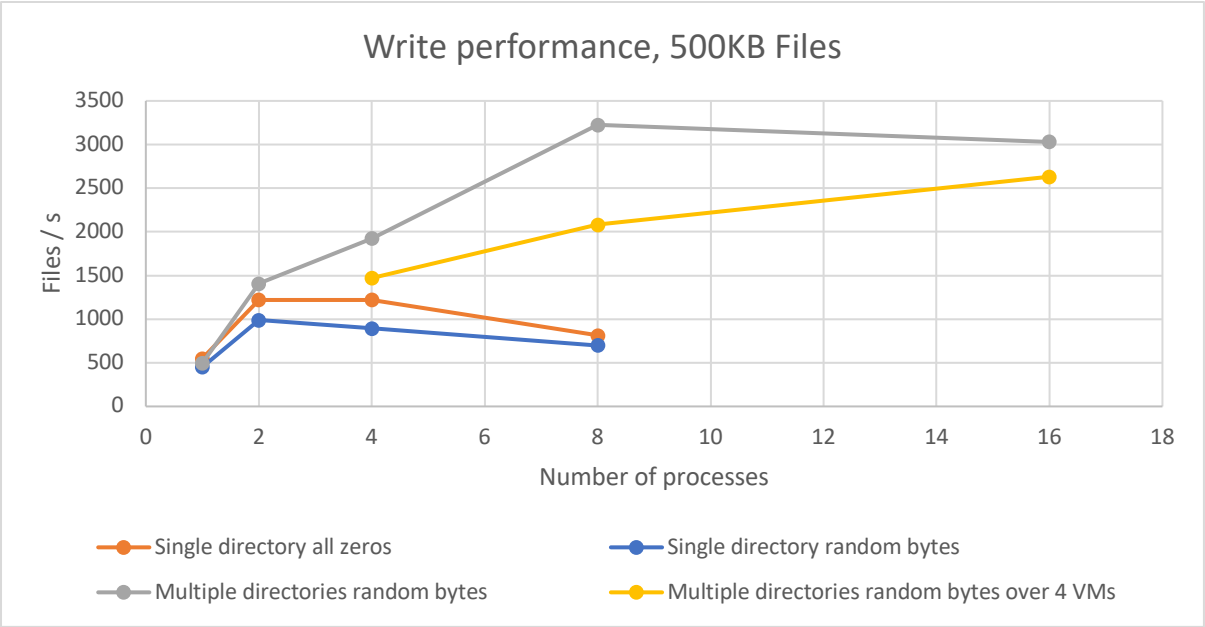
CephFS

Read and write tests were performed using a simple Python script to read/write around 500MB of data to a mounted CephFS filesystem. Local caches on the reading VMs were cleared between benchmark runs.

An important caveat is that these measurements were all taken at times when the rest of the system was relatively quiet. It was observed that at times when the Kafka cluster was active the performance was subjectively significantly lower and more variable. Unfortunately there was not time to explore this in any systematic way.

Write performance

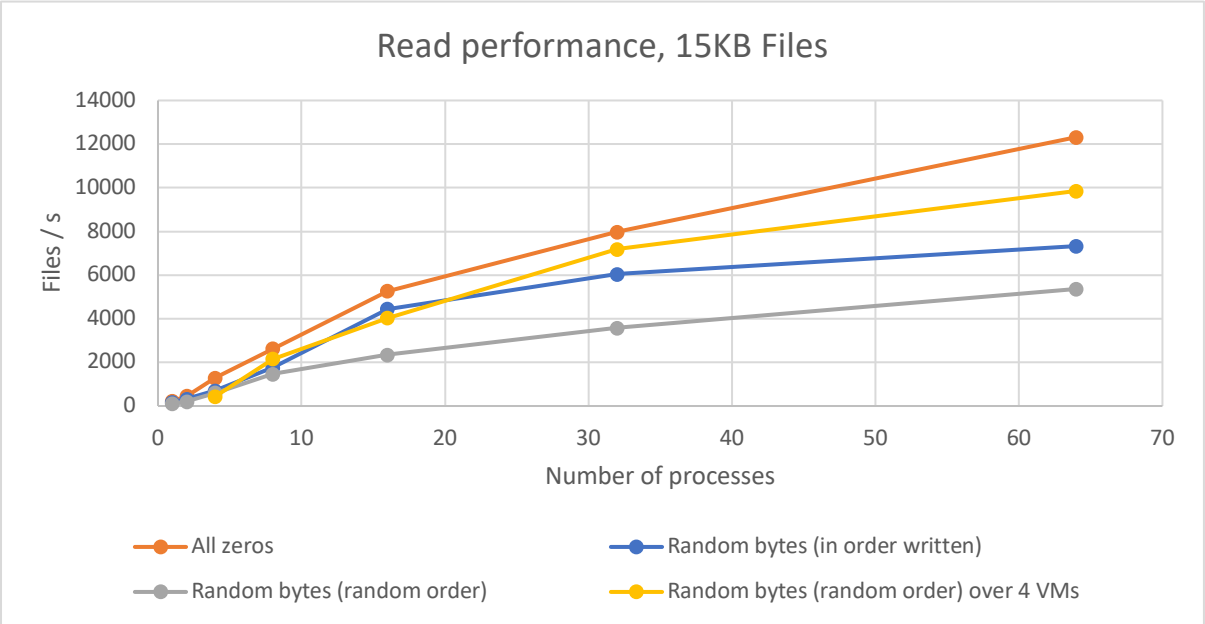


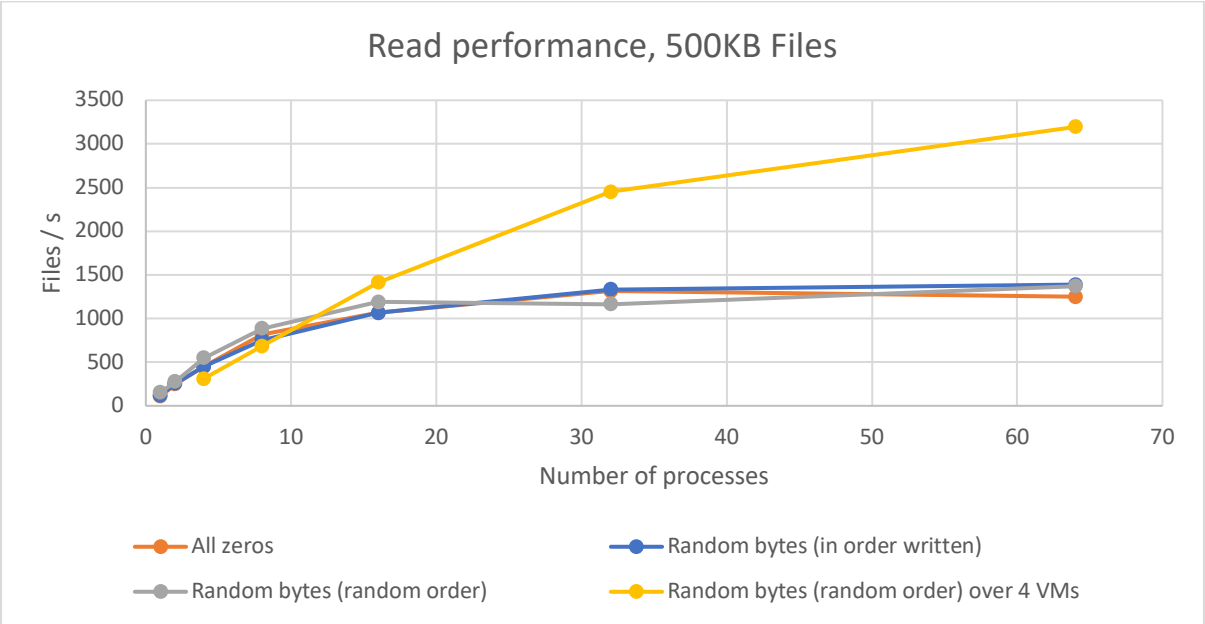


Note: overwriting a file appears to be significantly slower than creating a new file. Deleting the existing file first as a separate operation actually appears to be quicker.

		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

Read performance





It is unclear why compression and ordering appear to make a difference with 15KB files and not with 500KB files (one would expect the opposite if anything). Although local VM level caches were cleared between tests, it is possible that there is some caching taking place within the Ceph storage system that is causing this behaviour. It is also possible that it is simply an artefact caused by unaccounted load on the network and/or storage.

Latency

Latency to read a single randomly selected 15KB file (from a set of around 100,000) is around 0.25s. However, given the relatively small total volume of data involved and the potential for caching (especially of metadata), this is probably a best case scenario and we should consider verifying that latency remains acceptable in a more realistic scenario.

Other issues

CephFS is a shared file system, which has the advantage that minimal work would be required to modify existing code that already uses a filesystem for storage. However, it does require that all VMs mounting the filesystem use the same UIDs and GIDs, at least for those users that need access to it. This can be done using a central identity system (e.g. LDAP etc.) or by arranging for the use of common UIDs and GIDs through whatever deployment and configuration mechanism is used.

For users requiring an HTTP interface, that would need to be deployed separately, although this is not likely to require a large effort.

Summary

- Write performance appears to be adequate for image files at an ingest rate of 900 items/s, but potentially marginal at 2000 items/s.
- Write performance is sufficient for light curves.
- There appears to be a bottleneck when writing to a single directory – achieving useful parallel scaling on write requires writing to multiple directories.
- Read performance appears to be well in excess of minima, provided that multiple files can be read in parallel.
- Latency also appears to be acceptable.
- These performance figures may be slightly optimistic due to contention for network and storage from other workloads and caching effects. This is only likely to be a major issue with regard to writing image files as the other performance figures comfortably exceed minimum requirements.

Swift

It has not been possible to conduct comparable performance testing using the Swift interface to the Ceph storage cluster due to lack of time.

Two proof of concept experiments were undertaken. The first demonstrates querying an archive of ZTF light curves stored as JSON files over an HTTP interface and is described in detail at described at https://lsst-uk.atlassian.net/wiki/spaces/LUSC/pages/862355464/Data+Mining+Light+Curves. The second demonstrates extracting image files from a Kafka stream of alerts and writing them to a Swift object store and is described at https://lsst-uk.atlassian.net/wiki/spaces/LUSC/pages/975503385/Swift+image+store+POC.

Swift does have some advantages over CephFS in terms of ease of administration:

- Uses Keystone for AAI so no requirement to deploy manage anything additional
- No fixed size so no effort required to expand/manage as data volumes grow
- Built-in high performance HTTP interface for read access.

[i] https://confluence.lsstcorp.org/display/LKB/LSST+Key+Numbers