Lessons Learned from Lasair-ZTF

WP2.3 Deliverable D2.3.1

Project Acronym	LUSC-B
Project Title	UK Involvement in the Legacy Survey of Space and Time
Document Number	LUSC-B-09

Submission date	28/10/2020
Version	1.1
Status	Final
Author(s) inc. institutional affiliation	R.D.Williams (Edin) K.Smith (QUB) S.Smartt (QUB) A.Lawrence (Edin) G.Francis (Edin)
Reviewer(s)	Cosimo Inserra (Cardiff), Raphael Shirley (Southampton)

Dissemination level
Public

Version History

Version	Date	Comments, Changes, Status	Authors, Contributors, Reviewers
0.9	19/08/20	First full draft	Williams, Smith, Smartt, Lawrence, Francis
1.0	21/08/20	Submitted for internal review	
1.1	28/10/20	Revised following reviewer comments	

Table of Contents

1 Executive Summary

The LSST:UK project has built *Lasair*, (`https://lasair.roe.ac.uk`), a community broker for the LSST alerts that will encourage a variety of users in a variety of science investigations. The objective is twofold: to give UK and other scientists access to the LSST stream in near-real-time, and also to add value to those alerts.

In 2018, we built a web-database prototype broker for the Zwicky Transient Facility (ZTF)[1], that has a similar structure to what is expected for LSST, using Kafka to deliver alerts to their brokers. Both LSST and ZTF send an alert whenever a source is significantly (5σ) brighter than it was in an earlier reference sky. This paper reports on experiences of running this prototype.

In each section, we highlight the relevant **Lessons Learned** by the use of a bold red heading.

2 Scope of document

The original title of this deliverable in the Phase B plan was "Lessons learned on setup of ancillary catalogues for cross-matching ZTF alert stream". However, during Cycle-2 of WP2.3 and 3.2, we felt that it would be useful to assess and review the lessons more generally from running the ZTF prototype. To reflect this broadened scope, the deliverable has been renamed to "Lessons learned from Lasair-ZTF". This is meant to include the scope implied by the original deliverable title.

3 Introduction

In the following, we use the term Lasair-ZTF to mean the prototype we have built for ZTF alerts, and Lasair-LSST for the future system that will handle LSST alerts, expected to be 30-60 times the rate and bulk of ZTF. The Lasair-ZTF prototype, built by the Edinburgh-Belfast collaboration, clarifies the technological requirements for a Lasair-LSST broker. In particular, interacting with real science users has been invaluable, giving us insight to what they want to achieve, and driving the development.

Lesson learned: Relational databases and SQL became part of astronomy some 20 years ago, and astronomers are now effective users. Part of our task is to modify the paradigm from one of immediate execution of a query, to one of inserting a query or filter into the stream, with results seen later.

Users have been keen on using the website, building watchlists, streaming queries, and Jupyter notebooks. They like that Lasair allows such entities to be made public.

Lesson learned: Users would also like a more fine-grained permission, so that they can share these entities and their results with a group of colleagues.

During the development of Lasair-ZTF, several other prototype brokers have also been under development. The LSST project has facilitated discussions between these projects, which has been very useful.

Lesson learned: We should engage more systematically with other broker projects, especially Antares, Alerce, and Fink, and learn lessons from their strengths, weaknesses, and technological approaches.

4 User review and feedback

We hosted a 2 day workshop in Edinburgh, created a Slack workspace for users to ask us questions, a GitHub repository for bug fixes and feature requests, and held a Community User Requirements Review on 17th June 2020.

The top level **Lessons Learned** can be summarised as follows:

- Users require a well maintained and reliable database for querying, either through SQL or Jupyter.
- Users will require an API for accessing the database to pull information (e.g. to feed follow-up infrastructure such as ESO telescopes).
- The added value from Sherlock is viewed as high quality — although we must ensure that new large catalogues can be integrated rapidly (particularly photometric and spectroscopic redshift catalogues).
- Reliability and speed of return of the queries are high priority — returning results on the order of 60s from shutter closure (the LSST science requirement) is generally not a hard requirement for us. However, the database must respond rapidly to a query (in seconds), be reliable (no downtime at any time of day or night), and make the alerts visible within 5 (goal) to 15 (hard requirement) minutes of the exposure finishing.
- The sole result of Lasair can not, and will not, be a stream of annotated Kafka alerts. There is no strong demand from users for this to be the single method with which they interact with the data. Power users may emerge who want this, but it is unlikely to be the main method of data interaction.
- The timescale of variability of an object drives the timescale on which a user requires to be alerted. A star with a characteristic variability period of 60 days, for example, does not need a 60 sec alert; but a kilonova, with a decline timescale of 24hrs, requires immediate data access and a way of alerting the user (again, 5–15 minutes).
- There has been little visible engagement by the variable star community with Lasair functionality — either because their science is made possible through the ZTF data releases, or they are working but not engaging, or we do not (or perhaps the data can not) produce what they require.
- Introducing "tagging" is a high priority — GW maps, Fermi and HAWC maps, neutrino error circles.
- Lightcurve classification and annotations (which evolve over time) are a high priority.
- The User Interface requires improvement and we deal with that below.

5 User Interface

5.1 Static and Streaming SQL

Lasair-ZTF offers an interface based on SQL, a rich and expressive language. Users create and run SQL queries in the website to find the kinds of alerts relevant to their science, and these

queries can be run with the click of a button. Queries can also be stored, and kept private or made public, so that other users can see them. In addition, a query can be made *streaming*, meaning that it will be run in near-real-time as the alerts stream in.

Lesson learned: Some control is needed of the scope of the SQL, including time and volume constraints, as well as tutorials and sample queries, showing how to build a query that will execute quickly.

A static SQL query is semantically different from a streaming query[3]. Suppose, for example that a star S brightens considerably from the reference sky, and stays that way. A static query can be made to find sources brighter than some threshold, and it will find S; but if it is converted to a streaming query, then S will be "found" every time it is observed, and so there will be a repeat every 3 days or so.

Lessons learned: Users will want a way to have such repeats removed: they will want to click on the website to say "never again".

5.2 Nearby Sources and Sherlock Classification

Another feature of Lasair is the classification of the alert based on the position in the sky. The Sherlock[2] system cross-matches the position against over 40 large published catalogues — totalling about 5 TByte — and uses the results to make an intelligent decision about the natures of the alert.

Lessons learned: The Sherlock project will need to be more flexible in that there should be a well-defined procedure for adding new catalogues.

5.3 Watchlists

Users can create and upload a set of sky positions, each with an angular radius, that are called *cones*. The set of cones is known as a *watchlist*, and can be used in the SQL queries described above.

Lessons learned: Users like the watchlist facility. It is currently limited to about 20,000 cones; in Lasair-LSST we should strive to allow much larger watchlists.

5.4 Transient Name Service

Lasair-ZTF continuously cross-matches all the ZTF alerts with the list of supernovae that is maintained by the TNS[6]. This enables users to know if a suspected supernova has been already reported to the central clearinghouse.

Lessons learned: The TNS cross-match is popular, and we will need to cross-match with many more such services in the future.

5.5 Annotations

The objective of our LSST broker is twofold: to give scientists access to the LSST alert stream, but also to add value, which is added by means of what we shall call *annotations*. At first,

there was only one annotation, Sherlock, but as time passes, a variety have appeared and more types of annotation will appear in the future, that can be attached to any given object. Some examples of annotations are in the following list:

- Sherlock: a set of cross-matches with published catalogues, and a summary statement, based on position on the sky.
- Light curve classification: a system to assign an object's light curve to one of several classes, or to give probability estimates for the object being in the classes. One such example, which we have already investigated is RAPID[11]. Other such tools will also be investigated.
- Citizen science classification: Lasair-ZTF is collaborating with the citizen science project Zooniverse, resulting in classifications of alerts.
- Transient Name Server: Lasair-ZTF already has a near-real-time cross-match with the TNS[6], so users can discover if an explosive transient is already known. In the future, there will be other such services.
- Watchlist: The watchlist system defined above can be viewed as an annotation to an alert, when the object position falls inside a watchlist cone.
- Geometry: In the future, we expect to allow users to submit general sky geometry as a filter, for example: *"Alerts that fall in the SDSS footprint area"*. This will be encoded as an annotation.
- User-made tags: in the future, we expect to give users the ability to "tag" their favourite alerts in various ways.

Lesson learned: Annotations should be handled in a general, flexible way, rather than making handmade code and schema. In this way, it will be easy to add new kinds of annotation as the project progresses.

6 Lasair Architecture

6.1 Kafka-Database-Web system

Lasair-ZTF was developed as five linked machines:

- `lasair`: an Apache/Django webserver VM, acting as the user interface, that can execute queries on `lasair-db`, either through Django or directly. Users can get a login to this system — only a live email is needed — and thus save queries and watchlists under that login. The webserver also has NFS mounts of the data directories on `lasair-head` that have non-database files from the ingestion. This is the only node with an external address (`https://lasair.roe.ac.uk`).
- `lasair-db`: a MySQL alert database VM. After two years of operation, there are 100 million detections that define the light curves for 8 million objects, as well as 807 million non-detections, totalling 630 Gbyte.
- `lasair-head`: The data-ingestion VM checks the Kafka every few minutes for new alerts, and ingests a batch of up to 80,000 alerts using 8 execution threads. Images are written

to disk, detections and non-detections inserted to the database, object records updated, and streaming user queries run. The thumbnail images (FITS and jpeg) total 5 Tbyte.

- `lasair-sherlock`: The sherlock machine and database, and the Sherlock code, that reads and writes into `lasair-db`. Sherlock uses 41 standard catalogues that total 7 Tbyte.

- Apache Mirrormaker[4] cache: makes a replica of the ZTF alert stream. Deployed on the Edinburgh Openstack of the UK cloud "IRIS"[5].

Data streaming by Kafka was new to us in 2018, but now we have learned its idiosyncrasies: how to run multi-threaded and multi-processor, which exceptions to ignore and which are important, which versions of the client code have which bugs, how to avoid memory leaks.

Lesson learned: Kafka is a high-speed, real-time data streaming system that performs well for those that understand it.

The webserver is built with Apache and Django, which has proven to be a combination that is resilient and flexible, open-source, well-supported, and with all the extra plugins available that developers find they need.

The MySQL database is a key user interface — the users write queries in the MySQL dialect — so it is the centre of the architecture. In Lasair-ZTF, all information goes into the database on ingest, and only then is available for streaming queries. The deep experience of the Belfast group with MySQL has been invaluable, optimising it with indexes and settings, understanding why some queries run fast and some slow. When data is ingested, light-curve "features" are computed that include (for each filter) the mean, median, and standard deviation of the magnitudes, as well as moving averages with different lag times[10].

Lessons learned: A problem with MySQL is that it is is limited in its scalability. As the LSST stream is expected to be 30 or 60 times the ZTF stream, we will need parallel processing of the ingest and streaming queries, and moving away from the idea of "everything in the database" to more of a cost-benefit analysis — how much benefit will the users gain from the cost of storing this in the relational database?

Users can build SQL essentially freely, joining the object and detections tables, as well as the Sherlock cross-match tables, and their personal watchlists. In the early version of Lasair-ZTF, users could also use subqueries, as in `SELECT ... FROM (SELECT ...)`. Some of the queries that users would run would take a long time, and even longer as the detections database grew from millions to tens of millions. A new "query-builder" interface was introduced, so that queries are restricted in form to a single `SELECT` and a single `WHERE` clause, with a set of joined tables. While queries on a single table are fast enough, it is those with `objects JOIN detections` that are slow, i.e. those that query the individual points of the light curve rather than its features. We think to a future where the database will be much larger, with a lot more users.

Lessons learned: The Lasair-ZTF schema was changed, so that attributes are copied from the detections to the object table, and features computed, to allow many of the `JOIN` queries to be rewritten as queries on a single table.

A later addition to the Lasair architecture has been the Kafka mirror[4], that reads alerts from the ZTF source in the USA, and caches the Kafka stream as a replica. It allows the alerts to be pulled across the Atlantic with technology devoted to maximising bandwidth, then processed by the Lasair ingestion which is focused on computing and databases. It allows tests to be run at high bandwidth — alerts that took hours to read delivered in minutes — and it allows controlled, repeatable tests with the same data each time.

6.2 Jupyter

In addition to the tightly connected system outlined above, we also built a Jupyterhub interface running on IRIS[5], with a firewall tunnel to allow those jupyter processes to connect to `lasair-db` on the MySQL port. The login to this system has been confusingly different from the Lasair-ZTF login mentioned above, using the EGI checkin mechanism[7]. There have been certain frustrations with various types of interruptions to the service, and questions from users about how to extend the notebook to allow code to run asynchronously, either when a condition is satisfied, or at specified times by `cron`. Users have also asked for a way to share notebooks with colleagues that is easier that pushing and pulling to GitHub.

Lesson learned: Perhaps a more effective way to allow users to compute on Lasair data would be to provide an API, so they can run on their own systems, with Jupyter, sharing with whomever they like, storing data on their own system, and running analysis code whenever they like. An API can enable more general queries, including asynchronous queries for larger samples. It is possible that the need for an API should be added to the LSST:UK Science Requirements Document.

6.3 Hardware platform

The Lasair-ZTF prototype has been built on this equipment: 2 x Intel Xeon 2.1GHz 8 Core with 128GB memory and 12x4 TByte disk, split into four virtual machines, as above. As mentioned above, there is an Apache Mirrormaker cluster which is on on separate hardware under Openstack, and a Jupyterhub cluster on a different Openstack cloud (both part of IRIS[5]). This testbed made it easy to build the first version of Lasair-ZTF.

However, the eventual Lasair-LSST will run on a shared cloud architecture based on Openstack, that was being built at the same time as the first version of Lasair-ZTF. In the last year, we have learned to work with building and manipulating virtual machines through a web interface, rather than dedicated "bare-metal" machines described above.

7 Operations and Reliability

Over two years of operation, Lasair-ZTF has had very little downtime. Astronomers have devoted their time to it because it was sufficiently reliable that they could extract science rather than being overburdened by technical difficulties. The Palomar observatory has kept the ZTF stream running almost every clear night, even through the covid pandemic of 2020. Having said that, it has required continuous attention from the developers, and occasional "catch-up" for one or more days after glitches.

Lessons learned: The few times when the Lasair-ZTF was in technical trouble, dedicated staff have reacted promptly even at weekends. In the future, it would be good to have more formal plans for disasters. Instead of a single "production" machine, it would be better to have two equal systems, which can be quickly hot-swapped in case one of them is in difficulty. This is in addition to the "development" system.

Data flows every day, and the days when it doesn't flow are unpredictable. This makes it difficult to do operations such as OS upgrades, security patches, firewall maintenance, and adjusting the virtual machine configurations. We have generally run with three live services, with their own web, database, and sherlock VMs:

`lasair`: the production machine with the preferred domain name. We have tried to keep this running at all times — it has the entire 2 years of ZTF alerts.

`lasair-dev`: for software development. Fully functional, but no users are directed here, and it may not be working at any given time.

`lasair-iris`: running on the IRIS Openstack cloud, a test for differences between the bare-metal testbed implementation and the cloud implementation.

Lesson learned: While it is critical to run these other systems, it requires dedicated staff effort to maintain 3x4 = 12 virtual machines, plus the Mirrormaker. Use of automated setup tools such as Ansible[12] would make this job easier.

As the database grows in size, it takes longer and longer to make a backup. We have been making nightly backups of the Lasair-ZTF database starting at 9pm UK time each evening, and in the beginning this would be finished in an hour. However, with two years of data, it now takes five hours to do the backup, leaving only three more hours until observing starts again at Palomar.

Lesson learned: In the future Lasair-LSST we will need to pay attention to this problem of database backup and recovery.

8 Future

Sherlock needs to be a request-response system rather than the existing configuration where a single system reads and writes directly to the Lasair-ZTF database. In order to set up multiple Sherlock servers for speed, it should to be easy to install Sherlock on a new machine — not just the software, but also the 5 TByte database.

In Lasair-ZTF, all stored data is in either a relational database or a file-system, where the first can be queried in a high-level language, and the files only downloaded whole. As the data quantity and complexity increases it will be more difficult to sustain this model, and new kinds of storage and querying used. We are trialling Cassandra[13] and ElasticSearch[14] as data storage which is both scalable and queryable. We are investigating SparkSQL[15] and Dask[16] as ways to allow users to do large data-mining projects.

Another kind of data access is to not store the data at all, but to make it available on-demand. Instead of storing the voluminous output from an annotation system, we store only the conclusion, the tip of the iceberg, and give a URL link for the full data to be computed on demand.

As noted in the introduction, it is much easier to deal with a previously submitted streaming query than an ad hoc web query. The streaming query can be handled in a scalable way by multiple ingestion nodes, and a set of results accumulated for the user, whereas the ad hoc query must be handled by a database immediately — a database running on a single node that may be already working hard on ingestion at the same time.

It will be critical for Lasair-LSST to develop an effective programming interface (API) to provide access to the data. In this way, users can run their own Jupyter, Python, Cron, Kafka, and other services on their own machines, in whatever way they want, without the difficulty of a separate authentication system. The API should be based on web protocols (REST), and ideally in a formal way that allow automated documentation and maximum ease of use. Each API call would submit a token that identifies the user, and allows throttling if the usage becomes excessive.

A companion project to Lasair, in the same funding framework, is the LSST Data Access Centre (DAC), that will allow UK and other scientists to work with the LSST data releases that will appear after the first year or so of operation. The Lasair and DAC projects should be working closely to ensure that users can gain the most science from the combination of the two.

9 References

[1] F. J. Masci, R. R. Laher, B. Rusholme, et al. 2018, *The Zwicky Transient Facility: Data Processing, Products, and Archive*, PASP, 131, 995. Also https://www.ztf.caltech.edu/.

[2] https://lasair.roe.ac.uk/sherlock

[3] E. Begoli et. al, *One SQL to Rule Them All* https://arxiv.org/abs/1905.12133.

[4] https://kafka.apache.org/documentation.htmlbasic$_o ps_m irror_m aker$

[5] Digital research infrastructure of the UK STFC, https://www.iris.ac.uk/

[6] The Transient Name Server (TNS) is the official IAU mechanism for reporting new astronomical transients such as supernova candidates https://wis-tns.weizmann.ac.il/.

[7] https://www.egi.eu/services/check-in/

[8] K. W. Smith, R. D. Williams et. al., *Lasair: The Transient Alert Broker for LSST:UK* Research Notes AAS, 3,26 (2019). https://doi.org/10.3847/2515-5172/ab020f

[9] R. D. Williams, *The Lasair Transient Broker*, 2019, https://lsst-uk.atlassian.net/wiki/download/attachments/1146928/lusc-a-08_lasair_transient_broker.pdf

[10] R. D. Williams, *Outbursts with Exponential Moving Averages*, https://lasair.roe.ac.uk/lasair/static/EMA.pdf.

[11] D. Muthukrishna, G. Narayan, K. S. Mandel, R. Biswas, R. Hloæek, *RAPID: Early Classification of Explosive Transients using Deep Learning*, https://arxiv.org/abs/1904.00014, https://github.com/daniel-muthukrishna/astrorapid.

[12] https://www.ansible.com/

[13] https://cassandra.apache.org/

[14] https://www.elastic.co/

[15] https://spark.apache.org/sql/

[16] https://dask.org/