LARGE SYNOPTIC SURVEY TELESCOPE

Large Synoptic Survey Telescope (LSST)

QA Strategy Working Group Report

Bellm, E.C., Chiang, H.-F., Fausti, A., Krughoff, K.S., MacArthur, L.A., Morton, T.D., Swinbank, J.D and Roby, T.

DMTN-085

Latest Revision: 2018-07-06

DRAFT

Abstract

Abstract.

Change Record

Version	Date	Description	Owner name
9ec9484	2018-07-06	Unreleased draft.	Bellm et al.

Contents

QA Strategy Working Group Report

1 Introduction

Assignee

John

This report constitutes the primary artefact produced by the DM QA Strategy Working Group (QAWG), addressing its charge as defined in LDM-622.

2 Approach to the Problem

Assignee

John

The QAWG addressed its charge by sub-dividing the problem space into three separate areas:

- Addressing the needs of developers writing and debugging algorithms on the small scale;
- Developing tooling to address the *drill down* use case;
- Providing the infrastructure needed to support automatic testing and verification.

Each of these areas were assigned to a separate sub-group within the WG for brainstorming and developing approaches, with each sub-group regularly reporting progress to overall working group meetings.

When each sub-group had developed a strong concept for the tooling needed to address their particular part of the charge, the whole working group reviewed each design in detail, identifying and developing specifications for common components or activities that enable one or more of the designs.

In §§2.1, 2.2 and 2.3, we provide details about the charge provided to each sub-group. In §§3.1, 3.2 and we provide details about the charge provided to each

2.1 Pipeline debugging

Assignee

John

What tools do we need to help pipeline developers with their every-day work? Specifically:

- How do you go about debugging a `Task` that is crashing?
- Is `lsstDebug` adequate?
- Do we need an afwFigure, for generating plots, to go alongside `afwDisplay`, for showing images?
- What additional capabilities are needed for developers running and debugging at scale, e.g. log collection, identification of failed jobs, etc.
- What's needed from an image viewer for pipeline developers? Is DS9 or Firefly adequate? Is there value to the afwDisplay abstraction layer, or does it simply make it harder for us to use Firefly's advanced features?
- How do we view images which don't fit in memory on a single node?
- How do we handle fake sources? Is this a provenance issue?

2.2 Drill down

Assignee

John

How can we provide developers and testers with the ability to "drill down" from high level aggregated metrics to explore the source data and intermediate data products that contributed to them? Specifically:

- What sort of metrics should be extracted from running pipelines[1]?
- How can those metrics be displayed on a dashboard? Is a simple time-series adequate, or do we need other types of plotting?

[1] Scalars, vectors, spatially binned quantities, etc.

- By what mechanism can the user drill-down from those aggregated metrics to identify the sources of problems? Do they click through pre-generated plots, or jump straight into a notebook environment?
- Assuming the user ends up in an interactive environment, what are its capabilities?
- What do the above tell us about the data products that pipelines need to persist (both in terms of metrics that are posted to SQuaSH, and regular pipeline outputs, Parquet tables, HDF5 files, etc)?

2.3 Datasets and test infrastructure

Assignee
John

What infrastructure must we make available to enable testing and verification of the DM system? Specifically:

- Are any changes needed to the way that DM currently handles unit testing?
- How are datasets made available to developers? Git LFS repositories? GPFS?
- What is the appropriate cadence for running small/medium/large scale integration tests and reprocessing of known data?
- How is the system for tracking metrics managed? — how are the metric calculation jobs run? By whom? How often?
- How run-time performance of the science algorithms be tracked?

3 Design sketch

3.1 Pipeline debugging

Assignee
John

3.2 Drill down

Assignee
Tim

3.3 Datasets and test infrastructure

Assignee
John

4 Core components

4.1 Updated pipeline debugging system

Assignee
Simon

Derived from §3.1.

i.e. redesigned `lsstDebug`.

4.2 Logging

Assignee
Simon

Derived from §3.1.

4.3 Capability for developers to run pipelines at scale

Assignee
Lauren

Derived from §3.1.

4.4 Guidance on visualization

Assignee
Lauren

Derived from §3.1.

We're requesting a set of guidelines for developers here, not a new framework — but that's still a concrete deliverable (it's just documentation, rather than code). We might suggest that these guidelines be developed by a new WG, per Simon's suggestion[2].

4.5 Image viewer

Assignee
Trey

Derived from §3.1.

As of 2018-06-12 we haven't converged on a solid recommendation here.

Key considerations:

- Firefly is the annointed solution being provided by DM to external stakeholders (commissioning, operations, etc). It feels right to everybody that we should be dogfooding it, and also benefitting from development being carried out for those stakeholders.
- Currently, Firefly is unappealing to developers (primarily, I think, because of slowness of user interface, and perhaps also due to installation issues). Can we resolve these issues?
- We'd want to support visualization in a number of different environments, for e.g.:
  - Inside a Jupyter notebook;
  - As a standalone tool, à la DS9;
  - Embedded in a dashboard, à la JS9, Aladin-Lite, etc.
- Do we lose flexibility by mandating the use of a backend-agnostic API (`afwDisplay`) rather than going "all-in" on e.g. a custom Firefly interface?

[2] `https://confluence.lsstcorp.org/display/DM/Pipeline+Debugging+Design`

- We'll need to do full focal plane visualization, which none(?) of the current tools support well.

Options include:

- Do nothing; continue as we are, which means most people will use DS9 and a few will drift to Firefly as commissioning ramps up.
- Issue some sort of edict that pipelines developers have to use Firefly.
- Encourage the use of some other tool (Ginga?) instead of or as well as some of the above.
- Probably others.

Sounds like we need somebody from the QAWG to actually write some requirements — or a wishlist set of features we want — here.

4.6 Catalog visualization tools

Assignee
Lauren

Derived from §3.1.

For visualizing bigger-than-memory catalogs. May include e.g. the capability to spin up Dask clusters on demand, combined with Holoviews/Datashader/whatever. Somebody who knows about this stuff needs to write a summary...

4.7 Provenance

Assignee
Hsin-Fang

Derived from §3.1.

This section should note:

- That provenance is an immediate issue impacting QA work, so a solution is a priority;
- Some requirements as to the granularity at which provenance tracking is necessary for QA.

4.8 Documentation content updates

Derived from §3.3.

Assignee
John

- Clearer guidance on unit tests.
- Clearer guidance on code review, with requirements for test coverage etc.

4.9 Testing for documentation

Derived from §3.3.

Assignee
John

- Examples.

4.10 CI system updates

Derived from §3.3.

Assignee
John

- Test coverage.
- Tighter control of the environment.

- Better notifications.
- Better descriptions of which jobs do what.
- Clear description of what Developers are required to do before merging to master (see also §4.8).

4.11 Metrics Dashboard / SQuaSH

Derived from §3.3.

Assignee

Angelo

To date, SQuaSH has been used to follow a subset of KPMs computed by `validate_drp` for tracking performance regressions due to pipeline changes by regularly reprocessing test datasets in Jenkins/CI.

The following recommendations would enhance SQuaSH capabilities for DM developers.

Recommendation

SQuaSH should be used by developers for tracking metrics on their particular projects.

Developers can instrument their science pipeline Tasks using `lsst.verify` and create new verification packages to be tracked in SQuaSH (see e.g. `jointcal`). It would be interesting to send results to SQuaSH when testing development branches, so that developers can compare the new metric values with the previous values *before* merging to master. Any metric defined in `lsst.verify.metrics` should be uploaded to SQuaSH including, for example, computational metrics like code execution time.

Recommendation

SQuaSH should provide automated notification of regressions.

Metric specifications in `lsst.verify` include thresholds that can be used to automatically detect and notify regressions. The notifications could be presented to developers by Slack, for example.

Recommendation

SQuaSH should provide a metric summary display.

Verification packages might have specialized visualizations for displaying metric summary information in addition to the current time series plot. DM developers should be able to extend SQuaSH by creating new visualizations following developer documentation provided in the SQuaSH Documentation (https://squash.lsst.io/)

Recommendation

SQuaSH should support the LDF execution environment in addition to Jenkins/CI.

Pipeline runs on larger datasets (e.g. HSC RC2 weekly reprocessing) require more computation than can be provided in the Jenkins/CI environment. SQuaSH should be flexible to support other environments like the LDF environment.

Recommendation

SQuaSH should be able to store and display metric values per DataIds (e.g. CCD, visit, patch, tract, filter).

Pipeline runs on larger datasets (e.g. HSC RC2 weekly reprocessing) also require to store and display metric values per `DataIds` as opposed to the entire dataset (e.g. test datasets in Jenkins/CI). The ability to identify metric values per filter name or spatially by CCD in a visit or per patch in a tract, would enhance SQuaSH display and monitoring capabilities, turning SQuaSH or its successor into a richer metric dashboard (see also §**??**).

4.12 Standard format dataset package

Derived from §3.3.

Assignee

Hsin-Fang

4.13 Standard test package design

Derived from §3.3.

Assignee

Hsin-Fang

Should address the union of lsst_dm_stack_demo, ci_hsc, validate_drp use cases.

4.14 Updates to guidelines for GPFS-based dataset storage

A Glossary

aggregate metric An aggregation of multiple point metrics. For example, the overall photometric repeatability for a particular tract given multiple observations of each star.

aggregation A single result—e.g., a metric value—computed from a collection of input values. For example, we can sum or average a metric computed over patches to produce an aggregate metric at tract level.

dashboard A visual display of the most important information needed to achieve one or more objectives, consolidated and arranged on a single screen so that the information can be monitored at a glance (Few, 2013).

drill down Move from a higher level aggregation of data to its inputs. For example, given data describing a tract, we might drill down to constituent patches and then to objects; given a visit, we might drill down to CCD and then source. In the context of this document, it refers to the act of identifying an issue in a high-level summary of the data (e.g. an aberrant metric value) and interactively investigating its inputs to find the source of the problem.

GPFS IBM's General Parallel File System; now known as IBM Spectrum Scale. In DM use, this is taken to mean bulk data storage provided through a POSIX filesystem interface at the LSST Data Facility.

KPM Key Performance Metric.

metric We follow the SQR-019 definition of a metric as a measurable quantities which may be tracked. A metric has a name, description, unit, references, and tags (which are used for grouping). A metric is a scalar by definition. We consider multiple types of metric in this document; see aggregate metric, model metric, point metric.

metric value The result of computing a particular metric on some given data. Note that we *compute*, rather than measure, metric values.

model metric A metric describing a model related to the data. For example, the coefficients of a 2D polynomial fit to the background of a single CCD exposure.

monitoring The process of collecting, storing, aggregating and visualizing metrics.

point metric A metric that is associated with a single entry in a catalog. Examples include the shape of a source, the standard deviation of the flux of an object detected on a coadd, the flux of an source detected on a difference image.

QAWG QA Strategy Working Group.

releaseable product A software package or other component of the DM system which is expected to be included in the next tagged release of the system. At time of writing, this implies inclusion in a standard top-level package (e.g. lsst_distrib), but we note that future changes to the release procedure may render that definition obsolete.

SQuaSH Science Quality Analysis Harness; SQR-009; `https://squash.lsst.codes`.

tidy data Tidy datasets have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table (Wickham, 2014).

References

[SQR-009], Fausti, A., 2017, *The SQuaSH metrics dashboard*, SQR-009, URL `https://sqr-009.lsst.io`

Few, S., 2013, *Information Dashboard Design*, Analytics Press, 2 edn.

[SQR-019], Sick, J., Fausti, A., 2018, *LSST Verification Framework API Demonstration*, SQR-019, URL `https://sqr-019.lsst.io`

[LDM-622], Swinbank, J., 2018, *Data Management QA Strategy Working Group Charge*, LDM-622, URL `https://ls.st/LDM-622`

Wickham, H., 2014, Journal of Statistical Software, Articles, 59, 1, URL `https://www.jstatsoft.org/v059/i10`, doi:10.18637/jss.v059.i10