Gen3 Middleware Progress Report, 2019-06

Outline

- Milestone statement
- Butler with Oracle Registry
- RC2 Data Ingest
- QuantumGraph Generation Optimization
- Ongoing Database Concerns
- Batch Production Service v0.1
- Running RC2 with Gen3
- Planned vs. Completed

Overview Gen3 Milestone FY2019-2

- FY2019-1 Milestone:
  - Running ci_hsc via Gen3 + Pegasus + sqlite3 + global repo

- FY2019-2 Milestone:
  - Running RC2 via Gen3 + Pegasus + Oracle + global repo
  - More details:
    - https://confluence.lsstcorp.org/display/DM/Gen3+Milestone+FY2019-2

Making Butler more RDBMS-agnostic

Overall, this is a success story for our architecture: this involved a small number of actual code changes, with most of the time going into tracking down quirks.

- Used SQLAlchemy machinery for using sequences for auto increment fields, on backends that need it.
- Change how we disable connection pooling (SQLAlchemy treats SQLite unusually).
- Customized how views are created, serving as a template for extending to other databases as well.
- Slightly modified SQLAlchemy statements such that they produce more standard SQL statements.
    - Often there are multiple similar ways to do things with SQLAlchemy that at first glance seem to do the same thing, but they produce different commands which only have similar results.
    - This may be a clue for future optimization paths: minor statement changes may lead to more efficient DB access patterns.
- Modified our few hard-coded SQL statements to use more standard SQL syntax with broader DB support.

Oracle Backend for Butler

- Modified table names to be case-insensitive
  - SQLite and Oracle may be on extreme ends of RDBMs handling of case sensitivity (SQLite is extremely permissive).
  - While case-sensitive table names worked in Oracle with quoting, that isn't the normal Oracle behavior, and it causes red-herring problems when humans are using non-Butler tools to look at data in the database.
- Created Oracle Registry subclass (trivial; most behavior shared).
- View customization for Oracle.

Updating the connection strings and customizing the view creation also allowed the Butler to work with PostgreSQL. Similar minor changes should be all that is necessary to make Butler work with MySQL as well.

RC2 Data Ingest

Raw science images:

- Ingested directly into a Gen3 repo, given a list of filenames.
- Needs to be upgraded in the future to do vectorized inserts.

Master calibration images:

- Transferred from a Gen2 repository; until we can generate master calibrations in Gen3 directly, we have to pull validity ranges from the Gen2 `calibRegistry.sqlite3` database.
- No filtering based on what raws were ingested.
- *Really* needs to be upgraded to do vectorized inserts.

RC2 Data Ingest

SkyMap (tract/patch) registration:

- Runs directly against Gen3.
- Already upgraded to vectorized inserts (the difference between a runtime of multiple days vs. ~4 minutes).

Reference catalog ingest:

- Runs directly against Gen3.
- Only the shards overlapping ingested raws are ingested.
- No vectorized inserts (but not a bottleneck due to filtering).

RC2 Data Ingest

Bright Object Mask ingest:

- Converted from Gen2 repo, just to avoid hard-coding the filename template in the script (i.e. easy to make Gen3 native).
- Only patches overlapping ingested raws are ingested.
- No vectorized inserts (but not a bottleneck due to filtering).

RC2 Data Ingest

- A new Gen3 SQLite repo can be bootstrapped in ~4 hours, with most of the time spent ingesting master calibrations.
- Oracle is much slower (18 hours!), but we understand why and how to fix it. 85% of database workload due to overzealous use of savepoints. Additional overhead from row-by-row processing. Likely other optimization opportunities.
- There are high-level Python interfaces for all of these steps (`obs.base.gen3.BootstrapRepoTask`), but no general command-line interface - there's just a lot of highly-structured input information that's hard to map to command-line arguments.

QuantumGraph Generation Optimization

The QuantumGraph Generation algorithm starts with a big, complex (50 KB!) SQL query, followed by lots of small, simple, follow-up queries.

We started with a very naive system - no indexes in the database, no caching on the Python, and no previous attempts to optimize.

Predictably, it was *really* slow: we killed the first runs after waiting over 30 hours.

Unless otherwise noted, all benchmarks are for generating a QuantumGraph for all Tasks, in all 5 HSC bands, in just tract=9615, in Oracle.

Sources of QGraph Generation Slowness

Step 1: we didn't have any indexes, and knew we needed some.

- Added a small set of 13 indexes, identified from database trace while query was running (more can be added later; these were just the most important/obvious).
- Brought "big initial query" runtime from unbounded down to ~25 minutes.
- Remaining time (still unbounded) was in Python result-processing code and follow-up queries.

Sources of QGraph Generation Slowness

Step 2: duplication in follow-up queries (identified in DB traces)

- Two queries were being run ~3 million times each with the same arguments.
- Others were being run 1-2 million times with only <10k distinct arguments.
- We added caching for the results of these queries in Python.
- This brought the total time down to ~25 minutes (about 10% seems to be in Python rather than DB, but profiler overhead is now significant enough to make the exact amount unclear).

Sources of QGraph Generation Slowness

Step 3: inefficient ordered-search subquery

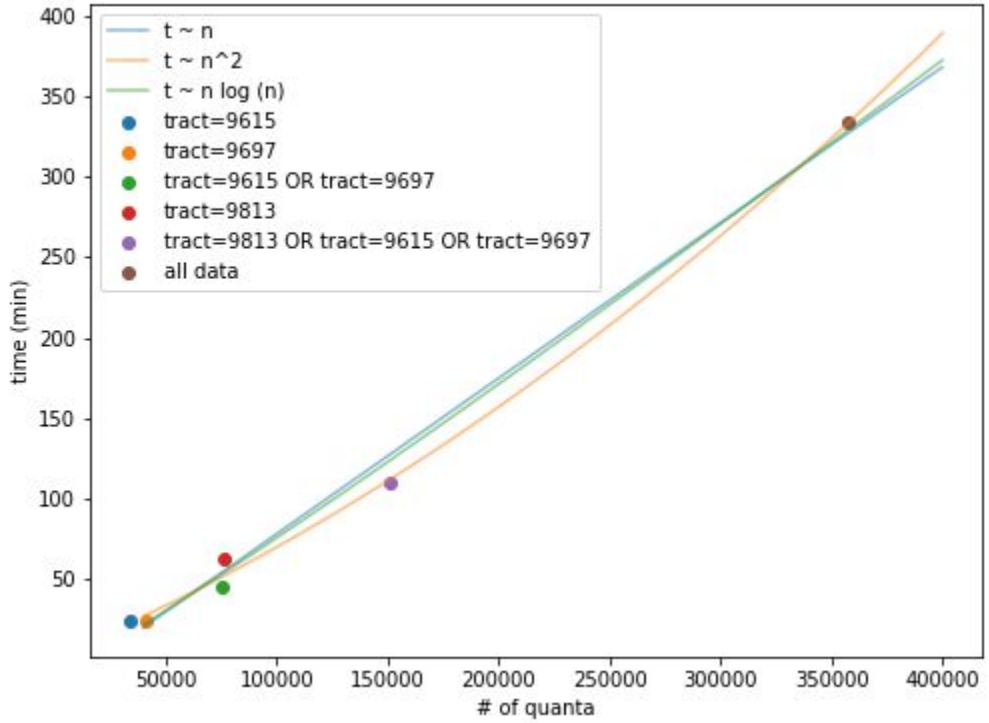
- A common piece of the big initial query is a variation on a subquery that searches an ordered sequence of collections for a particular dataset.
- This was originally written with a pair of nested CASE subqueries, and was rewritten to use a single CASE in a window function.
- This actually slowed things down slightly in our initial tests, but we're still working on the details. Ran out of time for now.

Sources of QGraph Generation Slowness

Step 4: Explore additional optimization opportunities

- Views are used to simplify SQL but we suspect they are causing issues in execution plan generation for a number of reasons.
- As execution plans change with iterative improvements, revisit overall indexing.
- Investigate causes of incorrect cardinality estimates by optimizer.
- **Possible schema design changes and data volume reduction.**

QuantumGraph Generation timing



Don't read too much into this; I expect it to change completely once we have had a chance to act on all of what we've learned already.

We should also be reporting time in terms of LIOs (logical I/O i.e. memory accesses), not wall-clock time.

Ongoing Database Concerns

- Ensuring predictable response times for dynamically generated, large join count SQL w/ inclusion of multi-registry functionality
- Connection management for highly concurrent pipeline processing
- Scalability limitations due to schema design
- Workload management / adequate service levels / partitioning of database resources as other workloads are added to the database
- Row-by-row transaction processing overhead

Batch Production Service v0.1

- Previous version consisted of a hardcoded shell script written specifically to run either demo or full ci_hsc pipeline after resetting repo.
- v0.1 Improvements:
  - BPS portion is all python (https://github.com/lsst-dm/ctrl_bps)
    - resetting repo is not part of BPS and still shell script driver
  - BPS takes an execution config
    - Allows setting memory requirement for pipetasks
  - Due to long wallclock of initial versions of QuantumGraph Generation, added feature to start from existing QuantumGraph
  - ctrl_exec/allocateNodes.py changed to create partitionable HTCondor slots

Running RC2 with Gen3

- Skip tasks skyCorrection, jointcal, ForcedPhotCcd for now
- Stack w_2019_21 & daf_butler tickets/DM-19808 + DM-19851
- RC2 repo creation using stack w_2019_20 and a calibration fixup
  - As mentioned in previous slide, this took 18 hrs
- Some manual fixups such as adding dataset types
- To run on multiple nodes using BPS:
  - lsst-dm/ctrl_bps branch tickets/DM-19846
  - Generate quantum graph for the full tract
  - Run by BPS via Pegasus
- Run using partitionable HTCondor slots created by ctrl_exec's allocateNodes.py in a Slurm reservation
- Working towards a successful run of a full tract. The most recent blocking bug in QuantumGraph generation: DM-19988

RC2 Tract 9615 Quantums

Count	Abbrev	pipelineTask
5067	isr	lsst.ip.isr.isrTask.IsrTask
5067	cit	lsst.pipe.tasks.characterizeImage.CharacterizeImageTask
5067	ct	lsst.pipe.tasks.calibrate.CalibrateTask
3351	mwt	lsst.pipe.tasks.makeCoaddTempExp.MakeWarpTask
405	cwact	lsst.pipe.tasks.assembleCoadd.CompareWarpAssembleCoaddTask
405	dcst	lsst.pipe.tasks.multiBand.DetectCoaddSourcesTask
81	mdt	lsst.pipe.tasks.mergeDetections.MergeDetectionsTask
405	dcsst	lsst.pipe.tasks.deblendCoaddSourcesPipeline.DeblendCoaddSourcesSingleTask
405	mmcst	lsst.pipe.tasks.multiBand.MeasureMergedCoaddSourcesTask
81	mmt	lsst.pipe.tasks.mergeMeasurements.MergeMeasurementsTask
405	fpct	lsst.meas.base.forcedPhotCoadd.ForcedPhotCoaddTask

Running RC2 with Gen3

Summary from a (failed) test run tract=9615

```
20739 Quantum = 20739 Quantum jobs,
11 pipelineTasks = 11 init-only jobs (e.g., create any science schema files)
Total jobs created by BPS = 20750

------------------------------------------------------------------------------
Type           Succeeded Failed  Incomplete  Total     Retries   Total+Retries
Tasks          20443     37      270         20750     37        20517
Jobs           22525     37      270         22832     37        22599
Sub-Workflows  0         0       0           0         0         0
------------------------------------------------------------------------------

Workflow wall time                                      : 4 hrs, 34 mins
Cumulative job wall time                                : 34 days, 9 hrs
Cumulative job wall time as seen from submit side       : 34 days, 10 hrs
Cumulative job badput wall time                         : 1 hr, 10 mins
Cumulative job badput wall time as seen from submit side : 1 hr, 10 mins
```

Failures tract=9615

37 job failures reported.
Currently BPS runs until can’t run any more.
Not yet determined if errors are due to same problem in DM-19988 or are separate issues.

- 1 cit - lst::psex::exceptions::InvalidParameterError: 'Only spatial variation (ndim == 2) is supported; saw 0'
- 19 ct - RuntimeError: Unable to match sources
- 1 ct - RuntimeError: No matches to use for photocal
- 2 cwact - RuntimeError: Unable to determine PSF to use for detection: no sigma provided
- 14 mmt - ValueError: Error in inputs to MergeCoaddMeasurements: source IDs do not match

Running RC2 with Gen3

Summary from a (failed) test run tract=9615

```
# 470446d1-2186-4fc9-ae01-45de57d452ab (G3M19c_000001_015)
Transformation           Count      Succeeded Failed  Min       Max       Mean          Total
cit                      5069       5067      2       8.021     295.507   97.334        493384.826
ct                       5087       5047      40      3.863     122.164   41.5          211111.006
cwact                    389        385       4       2.817     686.533   483.046       187904.728
dagman::post             22599      22525     74      0.0       18.0      2.919         65974.0
dcsst                    366        366       0       2.169     386.266   221.964       81238.867
dcst                     385        385       0       3.105     176.216   69.395        26717.263
fpct                     296        296       0       10.553    3207.465  2340.74       692859.124
isr                      5068       5068      0       4.061     49.942    27.261        138156.663
mdt                      74         74        0       2.675     29.446    19.753        1461.692
mmcst                    366        366       0       3.797     4418.415  2519.319      922070.802
mmt                      88         60        28      2.724     24.742    17.278        1520.433
mwt                      3329       3329      0       3.079     104.835   58.186        193702.604
pegasus::dirmanager      1          1         0       2.158     2.158     2.158         2.158
pegasus::transfer        2081       2081      0       3.438     23.554    9.616         20010.862
```

Planned vs Completed

Planned Deliverable	Status
Results of running Gen3 DRP pipeline on RC2 HSC data on lsst-dev ● Using Oracle as backend for registry of shared butler repository ● Pegasus for workflow	● No successful runs (blocked by DM-19988) ● Oracle: Working, but SQL inefficiencies need work ● BPS: improvement, but large gap to production system.
Gen3 run as part of the monthly HSC-RC2 reprocessing runs ● Incorporate into procedures with lower expectations than Gen2	Not ready at this time to be run regularly without extra effort.
Instructions for friendly-user developers using Gen3 Butler with Oracle	https://confluence.lsstcorp.org/display/DM/Oracle+with+Gen3 ● User access to Prod outputs delayed
Maintainable Registry code	Some reorganization deferred to later