VERA C. RUBIN OBSERVATORY

U.S. National Science Foundation

U.S. DEPARTMENT of ENERGY | Office of Science

Rubin Observatory: Scaling Towards Data Release Production using HTCondor

Gregory Daues, Michelle Gower
National Center for Supercomputing Applications
Throughput Computing 2026 Jun 9 – 12

Rubin Observatory Introduction

- Legacy Survey of Space and Time (LSST)
  - LSST: 10 year survey of the Southern sky, full coverage every 3-5 days
  - 30 s per exposure ➡ ~40 M images per year
  - Data transferred to US Data Facility SLAC (USDF), FRDF (CC-IN2P3), UKDF (RAL, LANCS)

- Different Flavors of Processing with Various Tools/Envs
  - Real Time Processing Efforts in Kubernetes Envs (Alerts)
  - Offline Processing using Panda FRDF UKDF
  - Offline Processing using HTCondor :
    - Data Release Production Work SLAC [Data Previews] : Next Slide(s) !
    - Solar System Objects Processing
    - Adaptive Optics System Analysis
    - General User/Developer Batch

Rubin Observatory : Data Release Production

- Annual data releases
  - Large undertaking, running frequently if not all the time
  - Automation / Minimize intervention
- Complex workflow
- ~100 tasks in the full pipeline
  - 100s M instances lasting from minute to hours
- ~500 distinct dataset types
- Data Release 1 (Y1 survey)
  - 50 PB data in total
  - ~3 G files
- Scaling up with Data Previews

Rubin LSST image processing steps

DRP Pipeline stages:

Stage 1: Single visit processing

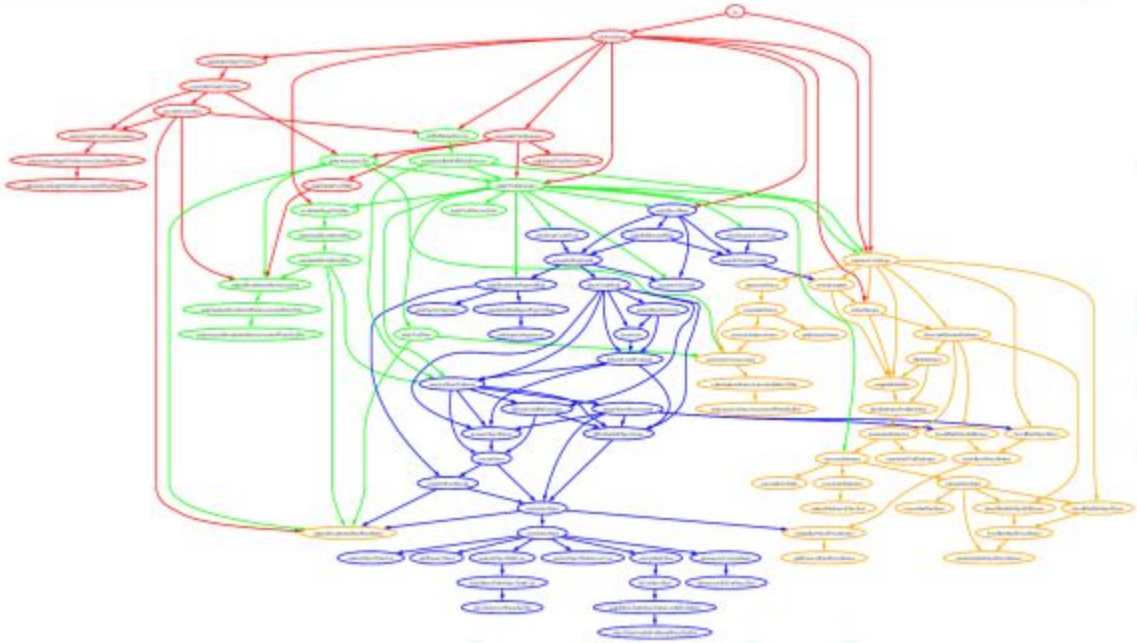
Stage 2: Recalibration

Stage 3: Coadd processing

Stage 4: Measuring Variability

(These have been color-coded to match the processing time-history plots.)

Almost all of these tasks are implemented as single-threaded jobs, and for a given task type, they are embarrassingly parallelizable.



Graph showing dependencies between task types for Rubin image processing. This graph was generated from the DRP pipeline in w_2025_33. More complete graphs, including data products are available with each release here.

Rubin Middleware Describing the Workflow

- Data Butler: Abstracts the data access details from the pipeline developers.
- Quantum : unit of data
- Quantum Node: Work to be done on a single set of inputs
  - A Task and a Unit of data to be processed (Quantum)
  - Example: remove instrument signature on the raw image for detector 3 of exposure 12345.

- Quantum Graph: a directed acyclic graph of QuantumNode objects
  - No runtime information (such as command lines or required memory) that would be needed to run jobs via a workload management system (e.g., Slurm).

BPS - Batch Processing Service

- Layer between Users/Rubin middleware and workflow management systems.
- Allows different workflow management systems to be used via plugins.
  - HTCondor, PanDA, Parsl
- Builds Quantum Graph based on user inputs via yaml
  - Sample inputs
    - Task to run : isr, calibrateImage, transformPreSourceTable analyzeAmpOffsetMetadata, analyzeCalibrateImageMetadata
    - Input data: inCollection: "HSC/RC2/defaults"
- HTCondor context: Converts the Quantum Graph into a DAGMan workflow dag by adding the runtime information.

BPS - Batch Processing Service

- Implements actions on DAGman workflows, e.g., “bps submit …”
  - submit
  - status
  - report
  - cancel
  - restart and more

```
%  condor_q —allusers
 OWNER   BATCH_NAME              SUBMITTED   DONE   RUN    IDLE   HOLD  TOTAL JOB_IDS
  …
lsstsvc1 LSSTCam_runs_DRP_DP2_v30_0_8_DM-55060_stage3_20260607T091744Z.dag+655
                                  6/7  05:12 329910   2889   5969     _ 413257 66250861.0 ... 66446010.0
```

‘Allocating Nodes’ via User Glideins

- SLAC Shared Science Data Facility (S3DF) is shared by experiments
  - Slurm Cluster 15,000 - 20,000 Rubin cores
- We target the HTCondor context for Rubin workflows
- Schedds are run administratively on Rubin login/devel nodes
- Users generate HTCondor EPs via Glidein Slurm jobs using ‘allocateNodes’

  - allocateNodes.py -N 10 -c 32 -q milano … s3df

- Glideins run as the user, accept the users jobs : Start = (Owner == “lsst1”)
- Users can configure BPS to run allocateNodes in a PROVISIONER Node

‘Allocating Nodes’ via User Glideins: Nodesets

- allocateNodes can run in ‘auto’ mode to detect job pressure
  - Generic e.g., 32-core for small jobs; Large jobs receive dedicated
- BPS runs can match specific glideins via “Nodeset”

  - allocateNodes.py -- auto -N 100 -c 32 –nodeset DAILY … s3df

    START= (Owner == "lsst1") && (JobNodeset == "DAILY")
    Nodeset="DAILY"

  - BPS provides jobs with:
    Requirements = (TARGET.Nodeset == "DAILY")
    +JobNodeset="DAILY"

Configuration for short HTCondor jobs

- Many workflows have > 100k small, quick jobs

- BPS / DAGMan will report many READY jobs, but 0 / few PENDING jobs.
  - Address this scenario with:

    DAGMAN_MAX_JOBS_IDLE = 50000

    DAGMAN_AGGRESSIVE_SUBMIT = True
- To ensure Pending/Idle jobs Run without delays in user glideins:

    NEGOTIATOR_IGNORE_USER_PRIORITIES = True

    NEGOTIATOR_CONSIDER_PREEMPTION = False
    CLAIM_WORKLIFE = 100000

Further Ideas for short HTCondor jobs

- BPS can clusters science work to be run by single job.
  - User defines how to cluster using pipeline terminology.

- Clustering side-effects:
  - HTCondor no longer has information about what science is currently being done (black box)
  - A failure inside a cluster causes ‘entire job’ to fail, which could block more downstream jobs than needed if clustering wasn’t used.
  - Potential for reducing number of input file reads by keeping data in memory, but the middleware currently always reads.

- HTCondor team to consider internal clustering strategies

Challenges in Reporting Status of large DAGs

- For Workflows of 100k - 1 millions jobs, reading NODE_STATUS doesn't scale
  - "bps report" currently parses <run_id>.node.status file
  - condor_dagman DAG_* classads are for the whole dag
    - DAG_JobsCompleted , DAG_JobsHeld , DAG_JobsIdle
    - DAG_JobsRunning , DAG_JobsSubmitted , DAG_NodesDone, DAG_NodesFailed

- Targets for HTCondor development : enhanced Categories for the DAG
  - Node.status file, DAG_* ads could be more fine grained using a classad
  - An example: bps_job_label is a classad assigned to all jobs by BPS
  - Users interested in status report divided by the bps_job_label classad

Extending DAGMan: Special Job Ordering

- One particular pipeline needs certain jobs to be run in a particular order but not a rigid dependency (shouldn't block/prune the other jobs on failure.)
  - Currently POST scripts are used to fake successes, and replacing actual dependencies with special jobs to handle when blocking is actually required.
  - Works if special jobs are first in workflow, but failures before special jobs will block more than we want.
- HTCondor investigating adding *weak dependencies* to indicate order without success/failure dependencies.

DRP Preparation at the USDF - SLAC Summary

- BPS + the htcondor plugin
- Allocating user glideins (--auto) on our lsstsvc1 service account
- Processing all internal to SLAC site/network
- Steady use of ~ 10,000 cores
  - 30-core or 120-core glideins
  - Processing over 40-50 days
- HTCondor job classad info is loaded into OpenSearch for campaign study
- Data Preview 2 campaign has made substantial progress, nearing completion

Rubin workflows for FRDF UKDF with HTCondor

- Base scenario has BPS workflows initiated at USDF SLAC

- HTCondor Collector, Schedd are run in Kubernetes at SLAC
  - IDTOKENS security
- AllocateNodes for different platforms: frdf, lancs, ral
  - Remote sites run ARC CE service to interface to local Slurm
  - AllocateNodes prepares grid universe arc jobs for glideins
  - Glideins running at remote sites join Kubernetes Collector
  - Nodeset attribute labels site
  - User at USDF/SLAC submits BPS run for the remote Nodeset

Rubin workflows for FRDF UKDF with HTCondor

- Test runs have been done using FRDF, LANCS, RAL
  - FRDF access from USDF has enabled broader Rubin middleware testing
  - Investigating resource access (dedicated and opportunistic nodes)
  - Users at USDF/SLAC now see a small ‘global pool’
- Looking into BPS submission to k8s service from other locations
  - FRDF, UKDF
  - Cloud based