Mu2e Production Techniques

Ray Culbertson

CSAID workshop

Jan 17, 2024

Doc–47453-v5

Mu2e time frame

- Mu2e detector components in final assembly
- Subdetector Vertical Slice Tests in progress or starting
- We do some VST production now
- Global Runs (horizontal slice tests) Mar 1, quarterly
  - recently: this might request processing
- Calorimeter moves to hall in a few months
- Cosmic rays, with production and calibration in ~July
- Tracker moves to hall in fall
- More cosmic rays, production, calibration
- KPP (detector sub-project end) in ~ one year or so

Topic 1 - Job control and recovery

how to set up production tools to detect errors and run recovery with high reliability, minimal resource usage, and minimal operator effort

Three Mu2e production styles

- Permissive
  - POMS resubmits based on job's status
  - no checks on output, goal is 97% complete
  - used for current simulation
- Rigid
  - no POMS
  - jobs write to temp locations
  - post-processing careful checks on output
  - used in older production simulation, and some today
- Proposed
  - POMS resubmits based on ddisp status
  - recovery jobs can fix output, no post-processing

Permissive production style

- Characteristics
  - All POMS, fife_launch, and fife_wrap based
  - effectively no job script
  - jobsub completion percentage requested 95-100%
  - no recoveries, ignore held jobs
  - write by add_to_dataset, add_metadata, job_output.dest
- Results
  - little setup, maintenance, effectively no experiment code
  - usually, mostly works OK
  - not uncommon to have incomplete datasets or file records
  - not uncommon to have different counts for different datasets output from the same job
  - no CRC check for incorrect writes

Rigid production style

- Characteristics
  - fcl generated by operator before each stage runs
  - No POMS, only custom experiment scripts
  - one universal job script on the worker node
  - current system, in total, few 1000's lines of perl code
  - only used for simulation

- Results
  - always 100% correct, CRC-checked end-to-end
  - always complete output
  - often requires several operator steps to repair, complete

Rigid production style writes (1)

- One stage starts with one non-DAG submission
  - fixed, ordered list of pre-generated fcl files
  - jobs tracked by input fcl file

- Job script
  - run one art exe
  - mkdir a unique (~hashed) dCache output dir
  - write output files to this dir
  - write manifest of files and CRC's, including manifest self-CRC
  - after write, mv hashed dir to JID.PROCESS
    - claims the job slot against any rogue job restarts
    - signals write is done

Rigid production style writes (2)

As jobs finish

- Operator runs a check script
  - checks output file CRC's
  - make an entry in SAM - a virtual file based on fcl file
    - checks for duplicates (output from same fcl file)
    - defines what jobs are done
  - mv output dir from temp dir to "good" dir
- Operator, when submission is complete
  - generate recovery from missing virtual entries in SAM
  - repeated until no more missing jobs
- Operator, when recoveries are done
  - copy files to permanent location
    - recoverable, with CRC check
  - declare output files to SAM

Rigid production style results

- Pros
  - CRC ensured at every step
  - no duplicates
  - 100% recoveries
  - only accept all or no output files from a particular job
- Cons
  - several operator steps
    - automation could be improved
  - two extra writes to dCache
  - extra virtual SAM entry
  - no POMS

Proposed production style (1)

Main concept

- a grid job will only exit with success if the job can prove all output is in the final location with correct CRC
- if a job fails for any reason, resubmit it
- the recovery job will repair/replace output files
- can be repeated as needed

Main points

- no post-POMS processing of files
- no intermediate dCache read/writes
- ~nothing outside of POMS campaign
  - no feedback from post-processing to POMS resubmission
- ~100% automated recovery

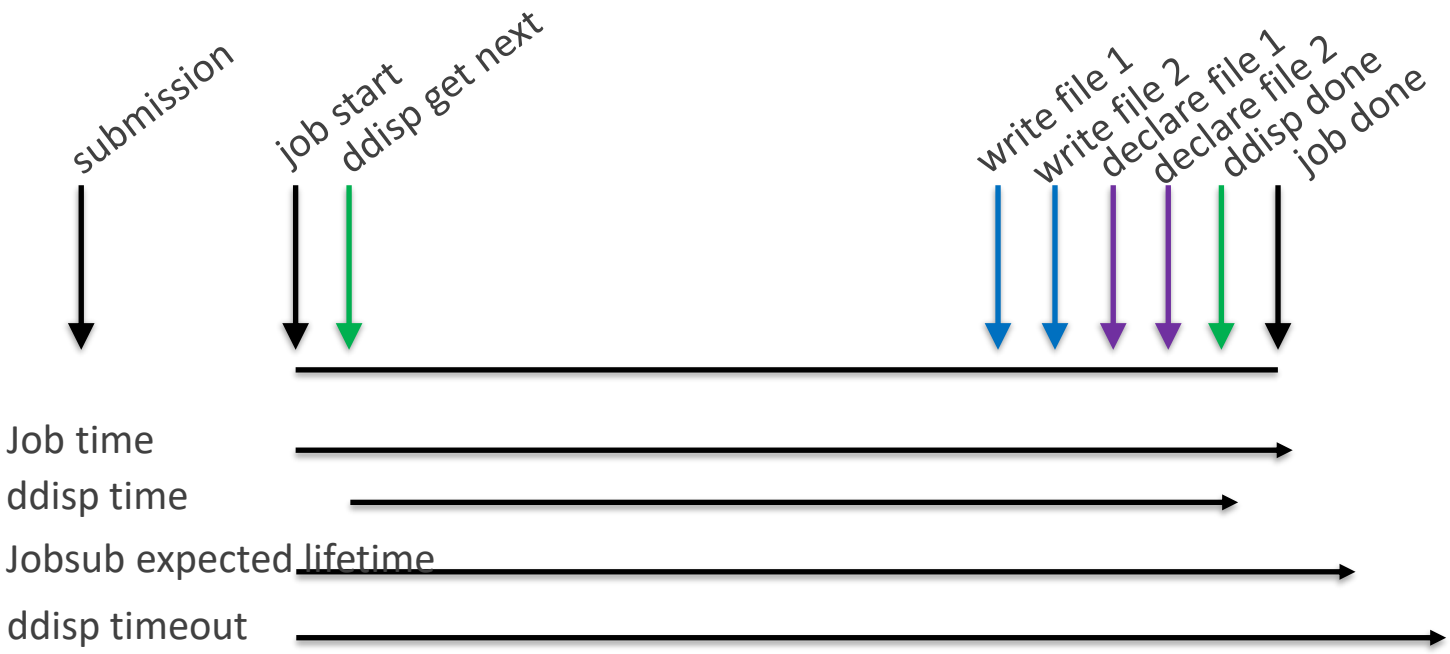
Proposed production style (2)

- POMS and metacat based
- stages driven off ddisp input dataset projects

Job script procedure

- write output files to final dCache location
- check file CRC via dCache database REST API
  - exit on problems, or, optionally, rewrite
- declare files to metacat
- if output files or metacat records exist, overwrite them
- on any error, exit with ddisp(no retry)/job failure
- if all output steps are successful, declare ddisp/job success
- POMS recoveries based on ddisp consumer status

A job timeline



- job might be interrupted anytime - with partial output
- set ddisp timeout longer than jobsub expected-lifetime
- job has "write permit" during ddisp active time

Error categories

- Daily, must be anticipated, handled automatically
  - dCache read/write failures, overloads
  - exe, conditions database, or script problems
  - timeouts, going to hold
  - container kills job
- Rare, but hopefully accounted for
  - tools (condor/metacat/ddisp) disconnect
  - tools show known rare behavior
- Arcane, would be a problem for any system
  - tools report success on failure
  - databases, files corrupt in place
  - random interfering activity

Error response (1)

- Mostly, jobs write, check OK, end with success
- Daily errors
  - script ends by reporting failure to ddisp
  - job dies, no ddisp report, ddisp times out
  - both these failures might leave partial output
- Daily errors handled
  - POMS sees failed/timed out ddisp file, submit recovery for this file
  - new job overwrites any existing output, returns success
- Data is released to the next stage or to users when all projects are complete and stopped

Error response (2)

- Rare errors
  - jobs are running w/o full connection to condor
  - can't write job records to elasticsearch
  - jobs can't contact metacat or ddisp
  - ddisp reboots?
- only a problem if correct, reported output is overwritten by rogue job bad output
  - more than one process has write permit at the same time
  - a job with write permit occurs after another reports success
- both would indicate a fundamental failure of POMS+ddisp, and the failure modes should be eliminated

Strong rules

Required so that it always works

1. ddisp timeout must be longer than job timeout so no output write ever occurs outside the ddisp expected write window
   - assume job timeouts work - if not, add a timeout in script
   - can timeouts fail? period check? job can't be killed? D state?
2. the same input file is never active with write permit for more than one ddisp consumer
   - recovery project must not include files that might be or become active in previous projects - all previous consumers resolved by report, timeout, or not started and canceled
3. a set of job output files is not used until the file is part of a successful consumer

POMS behavior

Required so that it always works

When considering a recovery job

1. wait for no active ddisp consumers
2. stop project
3. select all files not ending in successful consumer

This assumes that POMS makes a new ddisp project when submitting recoveries, but that's not really necessary

POMS behavior model OneP

Assume there is only one ddisp project per compaign/stage or slice of input files.

When considering a recovery job

1. wait for some large fraction of condor jobs to end
2. submit some number of recovery jobs, pointed to the existing, running ddisp project
3. repeat until the project is complete or retries exhausted

see next for details

POMS behavior model OneP - details

Using only one ddisp project for initial submit and recoveries

- Submit 100 jobs
- find 60 success, 10 fail, 10 timeout, 10 running, 10 unrun
- estimate the number of recoveries needed, with extras
  - in this case, maybe 20 or 30 recoveries
- submit 30 recoveries
- let ddisp sort it out
  - no jobs can clash (from strong rules)
  - extra jobs are rejected by "ddisp next", and just exit
- One could submit 105% of the project size every time
- recovery could still work, even weeks later

What will it take to get OneP done

- From Mu2e
  - code mostly exists, need tool features to finish testing
  - Mu2e DH code is ~2k lines of python using tool API's
  - mostly Mu2e conventions, conveniences
- Tools:
  - POMS must implement OneP recovery pattern
  - ddisp needs the virtual project (~done)
  - ddisp must sort "timeout" to "retry"
- Ideally
  - metacat allows "--force" to unretire update in one step
  - Rucio allows file updates (CRC, size)
  - metacat does not require a dataset when declaring a file

Topic 2 - Rucio in production

if production output is written directly to the file final location, and might be replaced during recoveries, then Rucio is a not a good tool for storing locations at this stage

General approaches (1)

1) Rigid

- files are written, and recovered, to temp locations
- when recoveries are done, copy files to permanent location and then write permanent Rucio record
- forces delay of following stage until Rucio record is available, or provide a secondary mechanism to find files

General approaches (2)

2) Proposed style

write files to final location, recover in final location

2a) with Rucio

- write Rucio record, overwrite this during a recovery job
- ddisp for following stages works OK
- problem: Rucio "can't" update CRC for the fixed file name

2b) Proposed style w/o Rucio

- do not write Rucio records during grid jobs
- provide a secondary mechanism for locations for following stages

Mu2e locations

- fixed file names (no hashes)

  tier.owner.description.configuration.sequencer.format
- we have three locations (RSE's in Rucio language)
  - dCache : tape, disk (persistent) and scratch
  - more can be added

- in all cases:

  file_dcache_path = simple_function(location, file_name)
- files with metacat, Rucio entries are in one of RSE locations

- unregistered files can go unstructured directories

Mu2e Proposal

- follow "Proposed w/o Rucio" general style
  - during production, do not write Rucio records
  - when production tranche is complete, declare Rucio records
  - Rucio records must be maintained for future DH flexibility
  - records are verified coherent automatically by a separate validation process (Mu2e does this already - very useful)
- problem: how does ddisp work w/o Rucio records?
  - use new "virtual dataset" ddisp project which skips Rucio
  - POMS stage must know I/O RSE by a separate mechanism
  - for production, almost always a constant
  - in stage config?
  - might include adding an RSE flag in metacat dataset record

Topic 3, some notes

- ddisp virtual project is ~done
- metacat and Rucio should not require a file is associated with a dataset - why this constraint?
- might be helpful if "ddisp create project -c" can provide content extracted from a dataset metadata as well as a file
- packaging declad is a high priority request for us

Summary

- Mu2e proposes a production style
  - write output from grid to final location
  - all recoveries directly within POMS
  - must allow file replacement and metacat record update
  - requires solid control of write permits via ddisp
  - hopefully we can implement OneP
- Mu2e proposes writing Rucio records after production is done
- We have a few months to complete this development and commission production before cosmic data starts
- the same collaborators are working on the same time frame to complete declad (FTS), spack, AL9, and other transitions