PHENIX

1. dCache export settings: retry dcachesizeof command when it fails (this can be implemented very quickly)

This is handled in the new system—a resilient export method (daemon or not, it will handle errors gracefully)

2. Limit on number of simultaneous dCache exports (analogous to pftp limits)

All steps esp. job-state changes (i.e. submitting to condor or making IO Requests) will be throttled in a configurable manner.

3. Error code communication: when the CRS job terminates in an error, have the possibility to run our own script to record that error, and have the option to reset the job

Part of the resilient exporting will be error handling and post-run scripts, etc...

4. Pre-stating of inputs from HPSS

Done via the disk cache—two possibilities, the one that is best would be to have separate read/write caches for HPSS, thus CRS could control when a cache-wipe would happen (assuming light user-usage), but this may not be feasible (need input from HPSS group). The other way to do pre-staging in a structurally different manner is to move the files yourself into dCache and have the job read them from there.

5. Loading of jobs by group, with customizable weights for each group

CRS Job Queue can be implemented as a priority queue, or order the next jobs for input based on the files the request (a form of bundling—if multiple jobs request the same file, submit all of them at a time once that one file become available)

6. The option to automate simple commands, like the cleaning of DONE jobs, or the reset of ERROR jobs.

The whole interface/implementation will be different, jobs that are finished can be set to be automatically cleared.

7. Documentation of existing features, like the option to save output if export fails

The new CRS will have extensive documentation (easiest to do in parallel with writing it).

STAR

1. Your bullet (3) (prestaging)
   (a) Prestaging - this works on CRS only and not on CAS and we would need this to work on CAS. Has been a pending item for at least a year. We need that feature yesterday :-)

See notes for PHENIX above, we'll be abandoning the extra slot methodology—for you, right now, the best way to do this is probably to move files into xrootd yourselves and specify that as the input source.

(b) Also, we seem to have observed a low number of jobs in MAIN_SLEEP state perhaps indicating a problem even on CRS with the pre-staging. Could this be watched and the mechanism confirmed working?

Not applicable as the entire thing will be re-written.

(c) Further extension (post-CHEP, I can discuss this with you in more details) we can discuss of mechanism to interface the CRS with a better and more efficient way to restore files and change the pre-staging mechanism altogether, relying on ERADAT and/or a DataCarousel like approach for ultimate efficiency.

We can discuss other possibilities.

2. Your bullet (11) (better error codes/configurable scripts to run on failure)
   (a) We do not believe this mechanism to be in place

See notes for PHENX and design flowchart—proper error handling/post-run scripting will be implemented

3. Your bullet (10) (bundle staging of input?)
   (a) The requested feature is not well-described as it stands. What we would like to do is have the possibility of declaring multiple input files but having the executable and its executable-args cycle over each input (seeing only the one defined for its pass when it runs) one by one, possibly having the staging out done asynchronously. In other words, the job description may look like (may need a two dimensional array)

```
inputcycle=2

inputnumstreams[0]=1
outputnumstreams[0]=1

inputstreamtype[0][0]=HPSS
inputdir[0][0]=WhateverPathinHPSS
inputfile[0][0]=blabla1.daq

outputstreamtype[0][0]=HPSS
outputdir[0][0]=Wherever
outputfile[0][0]=blabla1.root


inputnumstreams[1]=1
outputnumstreams[1]=2

inputstreamtype[0][1]=HPSS
inputdir[0][1]=WhateverPathinHPSS
inputfile[0][1]=blabla2.daq
```

```
outputstreamtype[0][1]=HPSS
outputdir[0][1]=Wherever
outputfile[0][1]=blabla2.event.root

outputstreamtype[1][1]=NFS
outputdir[1][1]=Wherever
outputfile[1][1]=blabla2.MuDST.root
```

and our executable would run over sequence 0 and have as input blabla1.root and produce one output to stage out to HPSS, then run over sequence 1 and read blabla2.daq and produce 2 output to be staged out. Each cycle (pass 0 for example) would see only the environment variables ACTUAL_INPUT## etc ... associated with its pass (in this case, for pass 0 only the ones associated to `inputnumstreams[0]=1` and `outputnumstreams[0]=1`).

A possible additional target

`mergerexecutable=XX`

would be run at the end and see all environment variables associated to all input and all output. We believe this will allow for greater flexibility at the end.

This seems like it is feasible with CRS as laid out in the flowchart—just that each "pass" will be a separate job as far as CRS is concerned. I'm not sure I understand the benefits of iterating one executable over multiple files in one job above feeding multiple jobs the same executable? Discuss?

- Other items include
  - Bullet (5) (not deleting files on stage-out error): there is no documentation on this feature I know off and I strongly suggests updating the documentation as we go so both experiments could be aware of each other's requested feature (historically, we have come to the same conclusions as per needs, not necessarily at the same time as we view problems in different priority order).

This is taken care of in the new design, copying files to the local disk is easy and very cheap to do if all we do is create hardlinks to the files in the Condor execute directory in /home/starreco/<whatever> so that when condor cleans up those files they remain on filesystem (thanks to Chris for that idea). Documentation will be redone, freely available, and thorough.

  - Bullet (8) (configurable timeouts (in HPSS? maybe...) - pretty sure we cannot control this but if so, please comment

Anything we do job-side will have configurable timeouts—if we can redact jobs from the HPSS request queue that will be implemented too (Note: This is not possible now)

  - Bullet (9) (staging-out not holding slot) - understood from Chris this is not implemented although would be nice to have. Ideally, the slot could be released not based on occupancy but perhaps, raising some flags - one care would be to check that staging-out do not accumulate (we can discuss mechanisms further)

See notes for PHENIX, we may have an export daemon so condor doesn't block on export—this looks doubtful,

will reevaluate based on the observed timings of exporting in the new system.

- I noted we would soon discuss staging into Xrootd from production as possible additional method to export.

Yes,see the note about PHENIX prestaging and note for your #1 above.

- We had a discussion of the cleanup of DONE jobs - From STAR's stand point, only that state (and no errors) would be useful to cleanup. Possibility to do so should be optional.

See note about same for PHENIX.

- Creating class of jobs for a general priority scheme seems fine. Note that within the new longer term pre-staging mechanism suggested/hinted, re-shuffling of job order may happen (in other words, the interaction between both features will need to be thought with care before a design).

Yes, job priority/reshuffling based on common imports is possible in the new CRS. Have to balance between the efficiency you get from submitting many requests to Oak Ridge and letting it shuffle the order of reads versus the efficiency of implementing our own scheme.