Gino Maintenance

Setup

(1) VERY IMPORTANT. I've changed the way my perl code includes my perl modules.
They now find the PDB modules using an environment variable named '$PDB_HOME'. This has several side-effects:

(1.a) ANYONE wanting to call one of my perl scripts, including the very essential 'CreateRun.pl', must define this variable in their environment.

Eventually we can create a disk location in AFS for the production version of the code, and I'll make a group settings file that can be sourced that sets all this up.

Code Location

For the moment, I have the code checked out at '/u/gl/glast/software/DPF/PDB/'. I suggest users point to that location. Do a 'setenv PDB_HOME /u/gl/glast /software/DPF/PDB/' in your .cshrc files.

(1.b) Because of this change, taskProcessTemplate.pl, the wrapper script used to launch every pipeline taskprocess has changed and you should update your wrapper scripts accordingly. Feel free to just replace the line near the top which reads:
use lib '/u/gl/dflath/glast/software/DPF/PDB';
with:
use lib $ENV

Unknown macro: {'PDB_HOME'}

;

Scheduler Cron job

(2) Alex has put the scheduler in chron using the glast@slac account. I'll push it through some hoops tonight and watch it go. It will append it's output to a logfile. It now prints a tagline with the date before each execution.

(2.b) I've also asked him to run a watchdog process every 10 minutes. It will add one run to my demo pipeline, and email him if the previous run failed. Hopefully this will give us a heads up if things are going wrong.

Cautionary Tales

(3) I'm working on a stored procedure that will allow a run to be rolled back in the case of a failed task process. I'll wrap it with a perl script so it can be invoked from the command line. This script will remove the failed taskprocess instance and associated (write) dataset instance records from the DB and update the run table so that the failed process runs again on the next execution of the scheduler. It will take the taskname and runname as parameters (same as CreateRun.pl)

(3.b) Be careful with this utility, a mass rescheduling of many failed TP's may cause problems... See (4) below.

(3.c) This shouldn't need to be run often unless the pipeline is breaking down, too many runs are executed simultaneously, or there's an error in someone's pipeline configuration or script. If it's being used a lot, perhaps Navid can add the option of calling the stored proc from his front-end run view.

(4) Do not create very many runs simultaneously. There should never be more than 40 active runs scheduled for all tasks combined. The scheduler allocates up to 2 concurrent oracle connections to each run. We have a limited number of connections, and the DB will begin rejecting them if it's exceeded. This will cause your TP's to fail, and may require the run to be rolled back using the utility from (3), above.

Things that can go wrong:

Whenever something goes wrong, you'll have a run record with a runstatus_fk of 4 (failed). Check the ActiveTaskProcess record. It'll point you to the TP that was run last. Look at it's tpi record for that run. Also look at it's associated TPI_DSI records and make sure they all exist. Then check it's LSF logfile, which you can find in the directory specified in it's taskprocess record.

(5.1) Same Task Process always fails:

Error in the wrapper script (check for misspelled DS names, etc.) Error in wrapped script – this is up to the user to fix. They'll see the error in the LSF log file.

(5.2) Can't create a new run:

You should get an error from CreateRun.pl saying an integrity constraint was violated. You already have a run of this name.

(5.3) Sometimes Oracle gets slowed down by lots of activity. This will cause weird errors in the logs.

Try creating less runs at one time

(5.4) These are about all I've seen. Anything truly weird, call me! OR,

Check the logs (scheduler log, ask Alex where he's put it, and LSF log for the proc.) If a pipeline script or module broke, you'll see an error pointing back to the line.

Mods to Stored Procedures

(6) If Alex edits the stored procedures (pdbprocedures.sql) there's an additional thing to do. If the body of a function or proc changes, nothing needs to be done. If, however, a new proc is added, or a proc signature changes, the following commands must be executed from the DPF/PDB directory (you can get DPF/PDB and the code in your own home directory by running "cvs co DPF"):

First, open sqlplus and login to the pdb tablespace. (I do an "alias sqlplus "sqlplus glast_dp/BT33%Q9]MU@slacprod" in my .cshrc to make it autolog me in.)

From the SQLPLUS prompt do:

@pdbprocedures.sql;

To load the new procedures in the DB. If it says you have errors, do:

show err;

And fix them.

Then do:

exit;

And then, from unix, do:

./smartGen.pl > ORACLE_SP_DPF.pm

To recreate the stored proc wrapper script.

Then make sure both that and pdbprocedures.sql are checked back into cvs, and that the newly generated perl module is checked out into the glast users account at ~glast/software/DPF/PDB/

Cleaning Up

Cleaning out a task

This can be done from the pipeline configuration web

https://www.slac.stanford.edu/www-glast-dev/cgi/Pipeline

Cleaning up the last task process instance in a run

Run a command line script from linux:

patched version is in ~glast/software/DPF/PDB/

rollBackFailedRun.pl <taskname> <runname>

Kills the failed tpi and sets runstatus to waiting so scheduler gets it.

Helper Scripts

Dan has a demo pipeline (named demo) that can be used for playing around with. He has scripts that fill it with dummy data and empties it.

Two scripts in ~dflath/pdb/

Run

testFill.pl <startRunName> <endRunName>

To add data to demo pipe.

Run

demoTaskDeleteRuns.pl

To kill all runs.