Larbatch Tools Update

Larsoft Coordination Meeting
Apr. 25, 2017

H. Greenlee

Outline

- Larbatch overview.
- Recent (and not-so-recent) improvements.
  - Posix-like interface layer & error-handling.
  - Validate-on-worker.
  - Streaming.
  - Separate bookkeeping and validation from data.
  - Offsite/OSG.
  - Add support for non-artroot data.

Larbatch Overview

- A batch job submission and validation framework, based on jobsub_client.
- Scripts.
  - project.py – Submit node script for submitting and validating batch jobs.
    - Controlled by xml configuration file + command line options.
  - projectgui.py – GUI interface for project.py.
  - condor_lar.sh – Batch worker script for running art (or compatible) executable.
  - lar.py – Batch worker script for analyzing ntuples w/art-compatible interface.
- For additional information, refer to larbatch wiki.

Posix and dCache

- Originally, project.py used exclusively posix access for i/o (python open function, python module os, python module shutil).
- Submit node posix i/o can have issues accessing files in dCache (mainly hang risk), but remains more performant on average than access using grid tools (gridftp or xrootd).
- Problems associated with dCache i/o are handled inside a posix-like interface layer, which all file i/o now goes through.

Posix Interface Layer

- Posix-like interfaces are supplied internally using python module larbatch_posix.
  - Interfaces simular to python open function, python module os, python module shutil.
  - Internally, actual i/o may be implemented using posix, ifdh, gridftp, or xrootd.
    - Posix i/o is normally preferred, if available.
    - Grid-accessible paths (e.g. /pnfs/...) will automatically revert to access using grid tools if not nfs-mounted.
      - You can run project.py on nodes that don't have dCache nfs-mounted, including your laptop.
    - A higher level of error-handling than using normal posix interfaces.
      - Including timeouts and ability to detach unkillable hung file accessing processes.

Validate On Worker

- Condor_lar.sh now has the ability to do many validation checks on the batch worker, as well as declaring files to sam and copying to FTS, on the batch worker, reducing load on submitting node.
  - Project.py will preferentially use on-worker validation results, if available, as opposed to doing its own validation checks.
  - However, main advantage of on-worker validation is declaring validated files to sam.
- To turn on validate-on-worker.
  - Add <check>1</check> to project configuration to enable validate-on-worker and sam declaration.
  - Add <copy>1</copy> to project configuration to enable copy to FTS dropbox from batch worker.
- Contributed by Joel Mousseau.

Streaming

- Starting and stopping sam consumer processes is handled by batch worker script, condor_lar.sh (using ifdh establishProcess).
- Recently added feature allows schema to be specified (xrootd vs. default gridftp) when starting sam process.
  - Use schema “root” to request xrootd.
- Xrootd streaming works with any executable that uses root i/o (including art RootInput).

```
<stage name="ana">
  <inputdef>prod_muminus_0-2.0GeV_isotropic_uboone_mcc8_ana</inputdef>
  <fcl>/uboone/app/users/greenlee/testrel/ana.fcl</fcl>
  <outdir>/pnfs/uboone/scratch/users/greenlee/ana/&name;</outdir>
  <logdir>/pnfs/uboone/scratch/users/greenlee/ana/&name;</logdir>
  <workdir>/pnfs/uboone/scratch/users/greenlee/work/&name;</workdir>
  <bookdir>/uboone/data/users/greenlee/ana/&name;</bookdir>
  <numjobs>30</numjobs>
  <schema>root</schema>
</stage>
```

Bookkeeping Directory

- Add a bookkeeping directory (<bookdir>) in your xml file to drastically reduce the number of file accesses in dCache.
  - Small (non-root) files from batch job will be rolled into one tarball on batch worker and untarred in the bookkeeping directory by project.py when you invoke the "--check" option..
  - All files created on the submit node by project.py will be made in bookkeeping directory.
- Best practice: specify bookkeeping directory on local disk (no need to be grid-accessible), or bluearc.

```
<stage name="reco1">
  <fcl>reco_uboone_mcc8_driver_stage1.fcl</fcl>
  <outdir>/pnfs/uboone/scratch/users/greenlee/reco1/&name;</outdir>
  <logdir>/pnfs/uboone/scratch/users/greenlee/reco1/&name;</logdir>
  <workdir>/pnfs/uboone/scratch/users/greenlee/work/reco1/&name;</workdir>
  <bookdir>/uboone/data/users/greenlee/book/reco1/&name;</bookdir>
  <numjobs>100</numjobs>
</stage>
```

Runing Offsite

- Latest larbatch condor_lar.sh includes an internal definition of environment variable UPS_OVERRIDE to allow run time environment setup to succeed on nodes running linux 3.x kernels.
  - export UPS_OVERRIDE="-H Linux64bit+2.6-2.12"
  - Contributed by Justin Hugon.

Non-Artroot Data

- By default, batch script condor_lar.sh invokes the larsoft art executable "lar" (identical to default art executable "art").
- Now added ability to specify an alternative executable using xml element <exe> in stage configuraiton.
  - Executable must understand the same options as lar/art.

```
<stage name="ana">
  <exe>lar.py</exe>
  <inputdef>greenlee_test_cint</inputdef>
  <fcl>/uboone/app/users/greenlee/testrel/cint.fcl</fcl>
  <outdir>/pnfs/uboone/scratch/users/greenlee/ana/&name;</outdir>
  <logdir>/pnfs/uboone/scratch/users/greenlee/ana/&name;</logdir>
  <workdir>/pnfs/uboone/scratch/users/greenlee/work/ana/&name;</workdir>
  <bookdir>/uboone/data/users/greenlee/book/ana/&name;</bookdir>
  <numjobs>50</numjobs>
  <maxfilesperjob>60</maxfilesperjob>
  <jobsub>--expected-lifetime=short -f /uboone/app/users/greenlee/testrel/SelectionI.C</jobsub>
  <jobsub_start>--expected-lifetime=short</jobsub_start>
  <schema>root</schema>
</stage>
```

Reading Ntuples and Trees in Batch

- Requirements.
  - Submit parallel batch jobs for reading ntuples and trees.
    - Generate histogram or filtered ntuples that are able to be combined using "hadd."
  - Able to read ntuples from sam dataset or file list.
  - Able to run pyroot or cint macros.
  - Generate sam metadata for output histograms or ntuples.

Reading Ntuples and Trees in Batch – Strategy

- Use same project.py and condor_lar.sh for reading ntuples and trees, as artroot files.
- For this to be possible, we need a drop-in replacement for standard artroot exectuable lar. Main requirement is that lar-replacement should understand the exact same command line options and arguments as lar.
- Drop-in replacement is called lar.py (a python/pyroot script).

lar.py – Command Line Options

- Lar.py supports a subset of the command line options of lar (specifically those used by condor_lar.sh):

```
 $ lar.py -h
lar.py [options]

Options:

-h|--help                  - Print help message.
-c|--config <fcl>          - Configuration.
-s|--source <input>        - Input (single file).
-S|--source-list <list> - Input (file list).
--sam-web-uri <uri>        - SAM project uri (input project).
--sam-process-id <pid>  - SAM process ID (input project).
-o|--output <output>       - Specify output.
-T|--TFileName <output> - Specify output (synonymous with --output).
-n|--nevts <nev>           - Number of events to process.
--nskip                    - Number of events to skip.
--rethrow-default          - Ignored (for compatibility).
```

lar.py Features

- lar.py:
  - Understands fcl files.
    - Fcl files are the only way using project.py and condor_lar.sh to feed configuration parameters (other than command line options) into lar.py.
    - lar.py has its own fcl configuration language, just like (but different than) lar.
  - Extracts one single TTree from each input file that it opens, which TTree is subjected to further processing.
    - lar.py doesn't know or care what data is in the TTree.
    - Works with, but not limited to, analysis trees.
  - Interacts with sam using ifdh python interface.
  - Has its own internal file and (optional) event/entry loop.
  - Can import arbitrary python modules containing pyroot macros and call them.
  - Can load cint macros and call them.
  - Opens and owns one output root file.

Pyroot Analysis Modules

- lar.py can import external (system-supplied or user-written) python pyroot modules, which handle most of the actual TTree processing.
  - Lar.py has no built-in knowledge of any TTree content.
- Each analysis module should supply one public factory function called “make” to make an instance of a python analysis class.
  - Analysis class should inherit from base class RootAnalyze.

```
def make(config):
    obj = MyAnalyzer(config)
    return obj

class MyAnalyzer(RootAnalyze):
   def __init__(self, config):
      ...
```

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info   : {}
            sam_metadata : {}
            cint_analyze : { CintMacro   : "SelectionI.C+"
                             AnalyzeTree : "SelectionI"
                             HistDir     : "selectI" } }
```

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info    : {}
            sam_metadata : {}
            cint_analyze : { CintMacro    : "SelectionI.C+"
                             AnalyzeTree : "SelectionI"
                             HistDir     : "selectI" } }
```

Name of input TTree

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info    : {}
            sam_metadata  : {}
            cint_analyze  : { CintMacro    : "SelectionI.C+"
                              AnalyzeTree : "SelectionI"
                              HistDir     : "selectI" } }
```

Python modules to import

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info   : {}
            sam_metadata : {}
            cint_analyze : { CintMacro   : "SelectionI.C+"
                             AnalyzeTree : "SelectionI"
                             HistDir     : "selectI" } }
```

Configuration Parameters that are passed to imported modules.

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info    : {}
            sam_metadata : {}
            cint_analyze : { CintMacro    : "SelectionI.C+"
                             AnalyzeTree : "SelectionI"
                             HistDir      : "selectI" } }
```

Tell lar.py to form Ttrees from multiple input files into a single TChain. Do this if imported modules should only be called once.

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info   : {}
            sam_metadata : {}
            cint_analyze : { CintMacro   : "SelectionI.C+"
                             AnalyzeTree : "SelectionI"
                             HistDir     : "selectI" } }
```

Tell lar.py to skip its internal loop over TTree entries, because in this case the imported cint macro has its own loop over entries.

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info   : {}
            sam_metadata : {}
            cint_analyze : { CintMacro    : "SelectionI.C+"
                             AnalyzeTree : "SelectionI"
                             HistDir     : "selectI" } }
```

Steal sam metadata from MicroBooNE artroot services configuration (artroot services are not actually loaded).

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info  : {}
            sam_metadata : {}
            cint_analyze : { CintMacro   : "SelectionI.C+"
                             AnalyzeTree : "SelectionI"
                             HistDir     : "selectI" } }
```

Event_info is a system-supplied utility module for extracting run, subrun, and event information from AnalysisTree trees.

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info    : {}
            sam_metadata  : {}
            cint_analyze  : { CintMacro    : "SelectionI.C+"
                              AnalyzeTree : "SelectionI"
                              HistDir     : "selectI" } }
```

Sam_metadata is a system-supplied module for generating sam metadata (json file).

An Example lar.py FCL file

```
#include "sam_microboone.fcl"

process_name : roottest
services : { FileCatalogMetadata: @local::art_file_catalog_mc }
input_tree : "analysistree/anatree"
chain : True
loop_over_entries : False
modules : { event_info   : {}
            sam_metadata : {}
            cint_analyze : { CintMacro   : "SelectionI.C+"
                             AnalyzeTree : "SelectionI"
                             HistDir     : "selectI" } }
```

Cint_analyze is a system-supplied utility module that supplies pyroot-to-cint interface.

Pyroot Analysis Classes

- Analysis Classes should derive from RootAnalyze.
- RootAnalyze provide default implementations (mostly empty), which analyzer classes can override, for all functions called by lar.py. Here is the complete list.

```
class RootAnalyze:
    def branches(self):
    def open_output(self, tfile):
    def event_info(self, tree):
    def analyze_tree(self, tree):
    def analyze_entry(self, tree):
    def begin_job(self):
    def end_job(self):
    def begin_run(self, run):
    def end_run(self, run):
    def begin_subrun(self, run, subrun):
    def end_subrun(self, run, subrun):
```

Cint Analysis Class

- System-supplied utility class CintAnalyze inherits from RootAnalyze and overrides the following functions.

```
class CintAnalyze(RootAnalyze):
    def __init__(self, pset):
    def branches(self):
    def open_output(self, tfile):
    def analyze_tree(self, tree):
    def analyze_entry(self, tree):
```

- CintAnalyze loads a cint macro (compiled or interpreted).
- In general, CintAnalyze expects the loaded cint macro to supply one or more top level functions.
  - Analyze_tree and analyze_entry functions take one TTree* argument.
  - Open_output function takes one TDirectory* argument.
  - Names of cint macro functions are fcl parameters.

Using lar.py with project.py (Example)

```
<stage name="ana">
  <exe>lar.py</exe>
  <inputdef>greenlee_test_cint</inputdef>
  <fcl>/uboone/app/users/greenlee/testrel/cint.fcl</fcl>
  <outdir>/pnfs/uboone/scratch/users/greenlee/ana/&name;</outdir>
  <logdir>/pnfs/uboone/scratch/users/greenlee/ana/&name;</logdir>
  <workdir>/pnfs/uboone/scratch/users/greenlee/work/ana/&name;</workdir>
  <bookdir>/uboone/data/users/greenlee/book/ana/&name;</bookdir>
  <numjobs>50</numjobs>
  <maxfilesperjob>60</maxfilesperjob>
  <jobsub>--expected-lifetime=short -f /uboone/app/users/greenlee/testrel/SelectionI.C</jobsub>
  <jobsub_start>--expected-lifetime=short</jobsub_start>
  <schema>root</schema>
</stage>
```

- Submitting and checking batch jobs.

```
$ project.py --xml cint.xml --submit
$ project.py --xml cint.xml --checkana
$ project.py --xml cint.xml --merge
```

(New) Best Practices for Using project.py

- Specify <outdir> and <logdir> to be the same and locate in /pnfs.
- Specify <workdir> to be different than <outdir> and locate in /pnfs.
- Specify <bookdir> to be different that <outdir> and <logdir>, and locate in /uboone/data or a local disk.
- When reading from sam, always include:
  - <jobsub_start>--expected-lifetime=short</jobsub_start>
- Consider allowing your job to run on offsite OSG.
  - <resource>DEDICATED,OPPORTUNISTIC,OFFSITE</resource>

Status of Tools

- Required updates to project.py and condor_lar.sh are available in mrb package larbatch.
- Lar.py and utility modules are available in mrb package ubutil.
  - ubutil/root_batch – MicroBooNE- and analysis-tree-independent files.
    - lar.py.
    - cint_anlayze.py.
    - root_analyze.py.
  - Ubutil/root_analyze – MicroBooNE or analysis-tree-specific files.
    - event_info.py.
    - sam_metadata.py.
    - hitana.py – native pyroot example.
- Since files have been in development recently, it may be necessary to check out larbatch and/or ubutil.

Migrating lar.py to larbatch

- Currently lar.py is not available in larbatch.
  - Only missing piece is python fcl module (also included in ubutil).
- Python fcl module has been made available to art team.
  - Art team will need to tweak fhiclcpp ParameterSet interface to make one method public ("walk").
  - At that time, new version of fhiclcpp can also include python fcl module.

Sam4Users

- Project.py has long had the ability to interact with sam, both reading files from sam and storing output files in sam.
  - Ability to rename generator files using uuid's to ensure uniqueness.
  - Ability to truncate file names to fewer than 200 characters, while ensuring uniqueness.
- Up to now, only support for classic sam.
- We think it would be useful to add support for sam4users.
  - I.e. invoke sam4users-style sam interactions via project.py command line options.
  - This will probably be the next new feature.