SFrame plugin for Ganga

Marcello Barisonzi

Overview

- Why do I need the GRID to run my analysis?
- What is Ganga?
- Why a dedicated SFrame plugin?
- Mini-tutorial:
    - Requirements
    - Installation
    - Running SFrame on Ganga
- Performance results

Why do I need the GRID?

- GRID analyses are the officially sanctioned way of analyzing data
- During ATLAS runs, large amount of data generated: unrealistic to download data on local PCs
- Even now, access to MC samples implies accessing the GRID (CASTOR becoming obsolete)
- It´s fast! (more on that later)

- However:
  - GRID commands difficult to use, easy to make mistakes, no tracking (what was my job number again?)
  - Several tools available to make GRID more user-friendly

What is Ganga?

- Ganga (Gaudi / Athena and Grid Alliance) is a Python interface to the GRID that is being developed jointly by ATLAS and LHCb.
- Makes task of using the GRID easier (not foolproof, though)
- Features:
  - Automated job tracking (job id, location, status etc.)
  - Job splitting and merging
  - Sandbox creation and retrieval
  - DQ2-based dataset searches
- Ganga homepage and tutorials available at: http://ganga.web.cern.ch/ganga/

Why a dedicated SFrame plugin?

- Ganga defines several types of applications, among which the generic **`Executable`** and the Atlas-specific **`Athena`**
- The **`Athena`** class has facilities to access data via DQ2, however it has the overhead of an Athena job: cmt configuration, large tarballs of code, etc. -> it´s an overkill for SFrame
- The **`Executable`** class, on the other hand, is too simple: it does not support DQ2, nor job splitting & merging
- Ideally, the SFrame plugin should offer full facilities combined with ease of use.

Mini-tutorial: Requirements

- The SFrame plugin for Ganga needs the latest version of Sframe -> modified directory structure, easier to maintain
- Ganga needs Python 2.4 or above to run
- The SFrame libraries (and your analysis code) must be compiled with exactly the same ROOT version installed on GRID machines: my advice is to use ROOT straight from an ATLAS SW release; for example, use this to set ROOT 5.14:

```
export ROOTSYS=/afs/cern.ch/sw/lcg/external/root/5.14.00e/slc3_ia32_gcc323/root
export PATH=${ROOTSYS}/bin:${PATH}
export LD_LIBRARY_PATH=${ROOTSYS}/lib:${LD_LIBRARY_PATH}
```

(you can also use an ATLAS release on afs/desy.de or ifh.de)

Mini-tutorial: Installation

- Retrieve the SFrameApp package from the WebCVS:
  http://isscvs.cern.ch/cgi-bin/cvsweb.cgi/?cvsroot=atdesyz
  or via the shell:
  ```
  export CVSROOT=:ext:isscvs.cern.ch:/local/reps/atdesyz
  export CVS_RSH=ssh
  cvs co TopPhysics/SFrameApp
  ```
- Once Ganga is set properly, go to the `TopPhysics/SFrameApp/Lib` directory and source the `setup.sh` script (this copies some utility files from the Ganga distribution, you have to do it only once)
- If you don´t have a `~/.gangarc` config file, create one typing `ganga -g` then edit the config file:
  - 1. In line 34, add `RUNTIME_PATH = GangaAtlas:$HOME/TopPhysics/SFrameApp` (assuming `$HOME/TopPhysics/SFrameApp` is where you checked out the CVS)
  - 2. In line 240, uncomment `VirtualOrganisation = atlas`

Mini-tutorial: Installation

- If you set up the environment correctly, by starting Ganga you should get a message like this:

```
ATLAS User Support is provided by the Hypernews Forum Ganga User and
Developers
You find the forum at
https://hypernews.cern.ch/HyperNews/Atlas/get/GANGAUserDeveloper.html

or you can send an email to hn-atlas-GANGAUserDeveloper@cern.ch

The SFrame Plugin is experimental: use it at your own risk
In any case, never write for help to marcello.barisonzi@desy.de
```

- Now you can test the SFrame plugin!

Running SFrame on Ganga

- Ganga is based on the Python shell, so you can enter commands interactively or write a script file.
- Create a new job object:
  ```
  j = Job()
  ```
- Assign SFrame as the application used by the job:
  ```
  j.application = SFrameApp()
  ```
- Set the directory where the SFrame directories (bin & lib & dev) are located:
  ```
  j.application.sframe_dir = '/path/to/SFrame'
  ```
- **Which SFrame configuration file do you want to use?**
  ```
  j.application.xml_options = '/path/to/my_example_top.xml'
  ```
- (This is tricky) Which ATLAS SW release contains the ROOT version you want?
  ```
  j.application.atlas_release = '13.0.10'
  ```
- Now prepare the tarball with the SFrame libraries:
  ```
  j.application.prepare()
  ```

```
SFrameApp                              : INFO     Creating SFrame archive:
/afs/ifh.de/user/m/mbarison/gangadir/workspace/Local/sframe-00005.tar.gz ...
SFrameApp                              : INFO     From /afs/ifh.de/user/m/mbarison/SFrame
```

Using remote datasets

- Choose a DQ2 dataset from the ATLAS Wiki and assign it to the job (let´s take ttbar production for example):
```
j.inputdata=DQ2Dataset()

j.inputdata.dataset = \
'user.top.TopView121303_MuidTauRec.trig1_misal1_mc12.005200.T1_McAtNlo_Jimmyv12000601.001'
```
- These options are a temporary workaround until OSG and LCG will talk to each other:
```
j.inputdata.match_ce_all = False
j.inputdata.type = 'DQ2_DOWNLOAD'
```
- How many files are there in the dataset?
```
j.inputdata.list_contents()
[large list of files]... In total: 316 files
```
- Let´s split the dataset in 16 jobs and merge it afterwards
```
j.splitter = SFrameAppSplitterJob()
j.splitter.numsubjobs = 16
j.merger = SFrameOutputMerger()
```

CE Selection

- Local job running is disabled by the SFrame plugin, so we have to choose the GRID backend and the CE:

```
j.backend = LCG()
j.backend.CE = "your_favourite_CE"
```

- Some standard CEs you might want to use:

  Karlsruhe : "ce-fzk.gridka.de:2119/jobmanager-pbspro-atlasS"

  Zeuthen   : "lcg-ce0.ifh.de:2119/jobmanager-lcgpbs-atlas"

  Hamburg0 : "grid-ce0.desy.de:2119/jobmanager-lcgpbs-atlas"

  Hamburg2 : "grid-ce2.desy.de:2119/jobmanager-lcgpbs-atlas"

  Hamburg3 (SLC4): "grid-ce3.desy.de:2119/jobmanager-lcgpbs-atlas"

- You can get more CEs by using the command `lcg-infosites` (from the command line)

Submitting the job, retrieving the results

- Now that the job is defined, give it a name and let it run:

```
j.name = 'My Test'
j.submit()
```

```
Ganga.GPIDev.Adapters        : INFO     submitting job 170.0 to LCG backend
Ganga.GPIDev.Lib.Job         : INFO     job 170.0 status changed to "submitted"
(etcetera)
```

- You can check the status of the job with `j.status` or `jobs`, and when the status is `completed`, you can run the merger:

```
j.merge()
```

- You will find a ROOT file with all your output data in:

```
~/gangadir/workspace/Local/<job_id>/output
```

Behind the curtains

- The SFrame plugin looks in the directory pointed at by `j.application.sframe_dir` and makes a tarball containing:
  - The SFrame executable contained in bin/
  - The core and user libraries contained in lib/
  - The DTD file contained in dev/
- Next, the XML option files is parsed, and the expected output file is added to the output sandbox. The XML options file is added to the tarball too.
- The DQ2 dataset is scanned, and logfiles are purged from the input list
- The dataset is split over several jobs, and the tarball is sent over the GRID

On the GRID

- The ATLAS SW environment setup script is sourced (so we get valid ROOT and Python versions)
- The tarball is unpacked and the SFrame libraries are added to the standard path
- Data is retrieved by DQ2 and a PoolFileCatalog is generated
- The XML option file is updated: the list of input files for SFrame is overwritten with the local filenames of the DQ2 data, luminosities are retained
- SFrame runs with the updated XML config files
- The output data is retrieved by Ganga and it is merged locally by the merger application (hadd or addAANT)

Performance

- Benchmark: analysis of the CSC 5200 sample with a dummy analysis
- Sample size: ~350k events, 158 TopView files
- Locally:
  DQ2 download time: 2h31m
  SFrame running time: 1m44s per file -> 4h33m
- Running on the GRID, split on 16 jobs (~10 files per job):
  On the DESY-ZN CE:
  Average time from submission to retrieval: 44m±25m
  Average job running time on GRID: 24m±2m
- DESY-HH times similar, FZK 100% faster
- LUDICROUS SPEED!

Conclusions

- The SFrame plugin for Ganga works!
- Need for beta-testers outside Zeuthen
- Interest from the SFrame development team (possible exchange of ideas with Attila K.)
- Wanted list:
  - Smarter setup scripts
  - Remote compilation
  - Bookkeeping of luminosities
  - Multiple datasets in one job
- To be announced to the Ganga users community soon, possibly co-opted in Ganga release 4.4 (better engineering needed)