PROCEEDINGS OF SCIENCE

SFrame - A high-performance ROOT-based framework for HEP data analysis

D. Berge
European Laboratory for Particle Physics (CERN), Switzerland
E-mail: David.Berge@cern.ch

J. Haller
Universität Hamburg, Germany
E-mail: Johannes.Haller@physik.uni-hamburg.de

A. Krasznahorkay[*†]
New York University, USA; on leave from the Institute of Nuclear Research of the Hungarian Academy of Sciences (ATOMKI)
E-mail: Attila.Krasznahorkay@cern.ch

In a typical data analysis in high-energy-physics a large number of collision events are studied. For each event the reconstruction software of the experiments stores a large number of measured event properties in sometimes complex data objects and formats. Usually this huge amount of initial data is reduced in several analysis steps, selecting a subset of interesting events and observables. In addition, the same selection is applied to simulated Monte-Carlo events and the final results are compared to the data. A fast processing of the events is mandatory for an efficient analysis.

In this paper we introduce the SFrame package, a ROOT-based analysis framework, that is widely used in the context of ATLAS data analyses. It features (*i*) consecutive data reduction in multiple user-defined analysis cycles performing a selection of interesting events and observables, making it easy to calculate and store new derived event variables; (*ii*) a user-friendly combination of data and MC events using weighting techniques; and in particular (*iii*) a high-speed processing of the events. We study the timing performance of SFrame and find a highly superior performance compared to other analysis frameworks.

13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research
February 22-27, 2010
Jaipur, India

[*]Speaker.
[†]We thank S. Ask, N. Berger, T. Eifert, and A. Höcker, for their contributions to the SFrame development.

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike Licence.

PoS(ACAT2010)048

1. Introduction

The Large Hadron Collider (LHC) has started operation this year. The experiments are already recording data close to their maximum bandwidth, which is around 300 MB/s in the case of the ATLAS experiment. The data is reconstructed at the Tier-0 site at CERN, and then distributed worldwide. The computing model of the experiments foresees that the reconstructed data files should be analyzed using various GRID technologies.

While the grid job submission framework in ATLAS provides easy access to the computing resources worldwide, the high latency (order of 1 day) of such an analysis makes it very inconvenient when developing or tuning an analysis. This is generally solved by creating smaller datasets using plain ROOT [1] TTree-s to store the event information in GRID jobs, downloading the datasets, and analyzing them locally. In general even this slimmed version of the datasets can be too heavy for running/developing the final analysis code. The SFrame model is to reduce the dataset size by keeping only interesting events and variables in multiple steps, each time creating a smaller dataset. This model scheme is sketched in Figure 1.

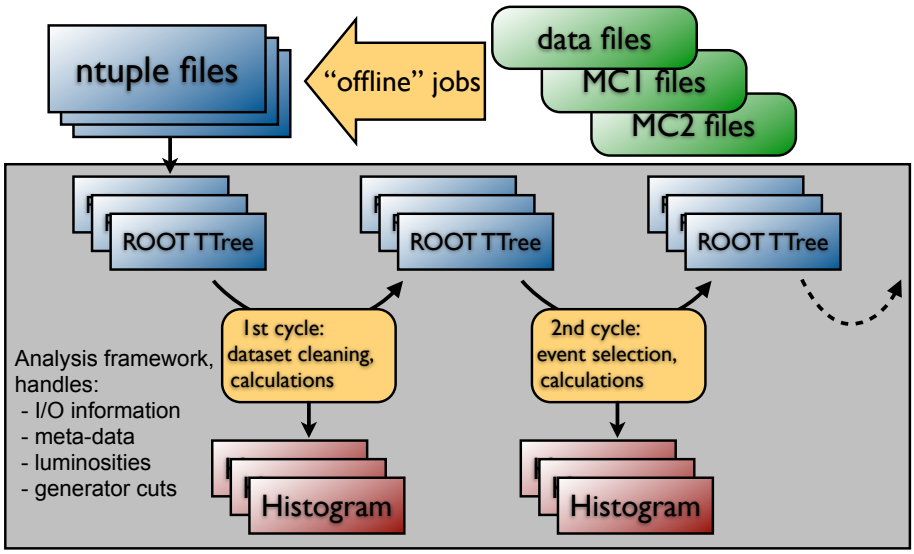

Figure 1: Analysis scheme for processing large sets of input data locally.

The SFrame package was designed based on a few general requirements. It had to be based on ROOT, as the event data storage is relying on ROOT. Reading TTree-based ROOT files had to be efficienct and fast. The code should provide simple interfaces for storing all kinds of TObject based and even custom objects in the output ROOT files.

The final requirement was that it should be easily configurable. It has to be simple to run the same analysis code on different datasets or configure the analysis differently under certain conditions. We chose to base the job configurations on XML, an easily extendable language to describe complex configurations, for which ROOT has built-in support.

2. Using SFrame / Quick Start

The SFrame code [2] is hosted on SourceForge [3], as it is not strictly tied to other parts of the

PoS(ACAT2010)048

ATLAS code. Most of the code's documentation can be found on the accompanying Wiki pages [4].

To check out and compile the code, one has to do the following as an example:

```
# cd analysis_directory/
# svn co https://sframe.svn.sourceforge.net/svnroot/sframe/SFrame/trunk \
    SFrame
# cd SFrame/
# source setup.[c]sh
# make
```

This sets up the environment for running SFrame jobs, compiles the needed libraries, the main executable, and an example library. To run the first example cycle on a machine that has access to CERN's AFS disks, one can run:

```
# cd analysis_directory/SFrame/user/config/
# sframe_main FirstCycle_config.xml 2>&1 | tee FirstCycle.log
```

The package also provides a number of scripts that make it easy to quickly create a new analysis package, and add analysis cycle skeletons to it:

```
# cd analysis_directory/
# sframe_new_package.sh MyAnalysis
# cd MyAnalysis/
# sframe_create_cycle.py --name My::AnalysisCycle \
    --linkdef include/MyAnalysis_LinkDef.h
```

This creates a directory alongside the SFrame directory, with the nominal SFrame package structure, and places one skeleton cycle – that has some comments in it –, with the file names `AnalysisCycle.h` and `AnalysisCycle.cxx` in the `include/` and `src/` directories. After editing these files, the package can be compiled with a simple "make" command in the package's top directory.

3. Code structure

The SFrame code is divided into 3 parts, each one compiling a separate shared library.

- **SFrame/core:** This directory holds the source files for the shared library which has to be used by all SFrame jobs, `libSFrameCore.so`.
- **SFrame/plug-ins:** This directory holds some helper classes that can be used in an analysis. These classes are not used by SFrame itself, but they can provide convenient functionalities in the users' code. The library created is called `libSFramePlugIns.so`.
- **SFrame/user:** This directory holds some user code examples. The examples try to demonstrate most functionalities of SFrame, including the the reading and writing of ROOT TTrees, creating histograms, configuring user properties from the configuration XML file, etc. The

PoS(ACAT2010)048

directory also serves as a template for users creating their own SFrame analysis package. The library created from the code is called `libSFrameUser.so`.

3.1 The SCycleBase class

The most important class of the core SFrame library is the one called `SCycleBase`. This is the base class that all analysis cycles should inherit from. It provides convenient functionalities for developing analysis code quickly. Since the base class provides a lot of different functionality, it has been broken up into many components. The inheritance structure of the class can be seen in Figure 2.

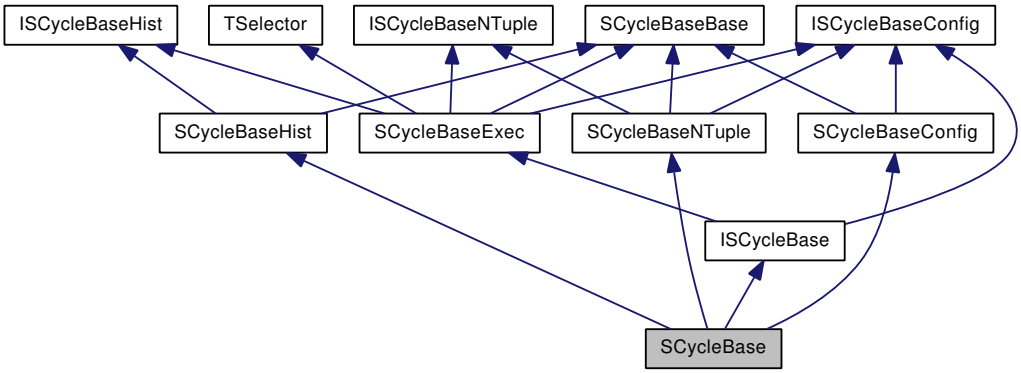

Figure 2: Inheritance structure of the SCycleBase class. The classes that have names beginning with "I" are "interface classes".

The detailed description of all these classes can be found under [5]. With this modular design it is quite easy to extend/change the functionality provided by the package. The core SFrame code only assumes that all cycle classes implement the `ISCycleBase` interface. So it is possible to create cycle classes that don't inherit from `SCycleBase`, but some other base class that also implements this `ISCycleBase` interface. A good example for this functionality is explained in Section 4.

3.2 The configuration file structure

The user analysis jobs are configured using XML files in SFrame. The user example package of the code holds two example configurations for running the two cycles implemented in the package. The XML files themselves hold a large amount of documentation on the structure of the files, and the online documentation also has a long section on the format of the XML files [4]. We only list the most important aspects of the configuration files here.

- `<Library .../>` and `<Package .../>` definitions: The user has to declare all the shared libraries and all the PAR packages [6] that are needed to run the analysis cycle(s).
- The `<Cycle ...>` definition: The user can declare a cycle by adding a piece of code like this:

```
<Cycle Name="My::AnalysisCycle" OutputDirectory="./results/"
       PostFix="_test" TargetLumi="123.4" RunMode="PROOF"
```


PoS(ACAT2010)048

```
        ProofServer="lite" ProofWorkDir="" ProofNodes="4" >
    ...
</Cycle>
```

This instructs the code to create a cycle object from the class called "My::AnalysisCycle", and execute it in 4 parallel processes on the local machine using PROOF-Lite.

- The `<InputData ...>` definition: SFrame uses the concept of InputData blocks to describe the input files of the analysis cycles. Each InputData block collects the files describing a homogeneous set of events. For example one would put a given type of Monte Carlo files, or data files recorded with the same conditions, in one InputData block. SFrame produces one output file per InputData, which collects the statistics of all the events belonging to that InputData.

4. SFrameARA

One commonly used extension of SFrame is called SFrameARA. ARA stands for Athena [7] ROOT Access, and refers to a set of ATLAS offline software libraries, which make it possible to read regular ATLAS reconstructed events in ROOT.

As discussed before, the `SCycleBase` class was created in a modular way to make it possible to easily add to or modify the functionality of the base class. The SFrameARA code takes advantage of this functionality. It modifies the behavior of some of the functions of the constituent classes of `SCycleBase`, to make it possible to read in so called ATLAS POOL files within a regular SFrame job.

This modular design allowed other kind of extensions in the past as well, for instance one group could extend the code to make it possible to read HepMC event files as inputs to SFrame jobs.

5. Performance

One of the main reasons for using SFrame is its performance. It can provide processing speeds very close to purpose-written code, while simplifying many aspects of writing the code. In the following we present a simple comparison between a number of different architectures in a computing-heavy job.

For the exercise we generated a set of 10 million events using the ATLAS offline software. Each event held a variable number of particles, with random kinematic distributions following some simple requirements. The results were saved into a set of ATLAS POOL files. Then, still using the ATLAS offline software, we translated these events into simple ROOT files, which didn't need any external ROOT dictionaries to read. We used the POOL and "flat" ntuple files as inputs to the performance comparison jobs.

Then we implemented the same, simple analysis code in all the architectures detailed below: we create all the possible 2, 3, 4 and 5 particle combinations from the generated particles, calculate a set of variables from the combinations, and fill about 20 histograms with the results.

PoS(ACAT2010)048

The measurements were run on 2 PCs running Scientific Linux 5 64-bit, each having a quad-core AMD Phenom^{TM} 9600B CPU, and 4 and 8 GBs of RAM, respectively. The generated input files were served from a 4 TB LaCie 4big Quadra^{TM} disk connected to one of the PCs. The tests used ROOT v5.22/00d, Python 2.5, and ATLAS offline software 15.6.4. Each measurement was run 5 times, to try to lessen fluctuations in the results. The "Athena" and "SFrameARA" jobs were using the generated POOL files as inputs, while all the other jobs used the ntuples.

Architecture		Input location	
		Local	XRootD
Athena		1.77 ± 0.02 kHz	N/A
ACLiC		3.85 ± 0.03 kHz	3.77 ± 0.04 kHz
CINT		259.0 ± 2.2 Hz	N/A
PyROOT		127.2 ± 2.2 Hz	123.1 ± 1.6 Hz
SFrame	LOCAL	4.04 ± 0.02 kHz	4.02 ± 0.03 kHz
	PROOF-Lite	15.92 ± 0.15 kHz	15.81 ± 0.13 kHz
	PROOF	N/A	29.53 ± 0.17 kHz
SFrameARA	LOCAL	3.26 ± 0.02 kHz	3.26 ± 0.02 kHz
	PROOF-Lite	13.03 ± 0.16 kHz	12.88 ± 0.18 kHz

Table 1: Performances of different software architectures, running the same analysis job. The meaning of the architectures is as follows; Athena: AthAlgorithm running in a small ATLAS offline job; ACLiC: The analysis code put into a class created by TTree::MakeClass(...), compiled and run from ROOT; CINT: A macro executed in ROOT in interactive mode; PyROOT: A Python analysis script using the ROOT bindings; LOCAL: SFrame running on 1 processor core; PROOF-Lite: SFrame running on 4 processor cores; PROOF: SFrame running on all 8 processor cores.

The results of this simple performance comparison can be seen in Table 1. A few comments have to be added to the results. The ATLAS offline software (Athena) was performing very nicely on the small generated events. Realistic analyses can usually run with about 100 Hz on reconstructed events. ROOT's compiled mode (ACLiC) is only slightly slower than the SFrame code running on one processor core (SFrame LOCAL), the only difference is that SFrame uses slightly quicker histograms than ROOT's own classes. As I/O was not a limiting factor in the tests, both the SFrame and SFrameARA jobs scaled very well with the number of processor cores. Finally, it seems very obvious that interpreted code (CINT, PyROOT) can not be used efficiently to perform high-performance analyses on LHC data.

6. Summary

The SFrame code provides a large set of convenience features for developing high performance HEP analysis code very quickly. Thanks to its flexible design, it can also be easily extended using additional libraries to support the needs of some special groups.

The code – despite the convenience features – performs almost as well as analysis code can perform, written from scratch. The development of the code is ongoing, we expect to add more features/optimizations as we develop more and more complex analyses using SFrame on LHC data.

PoS(ACAT2010)048

References

[1] R. Brun and F. Rademakers, "ROOT: An object oriented data analysis framework", Nucl. Instrum. Meth. A **389** (1997) 81.

[2] Links: http://sframe.sourceforge.net/, http://sourceforge.net/projects/sframe/

[3] Link: http://sourceforge.net/

[4] Link: http://sourceforge.net/apps/mediawiki/sframe/

[5] Link: http://sframe.sourceforge.net/Doxygen/

[6] Link: http://root.cern.ch/drupal/content/working-packages-par-files

[7] ATLAS Collaboration, "ATLAS Computing: Technical Design Report", ATLAS-TDR-017, CERN-LHCC-2005-022

PoS(ACAT2010)048