SAM at the Intensity Frontier

New interfaces for SAM to dramatically improve ease of use and integration.

Robert Illingworth, Adam Lyon[1], Marc Mengel, Andrew Norman, Rick Snider

May 2011

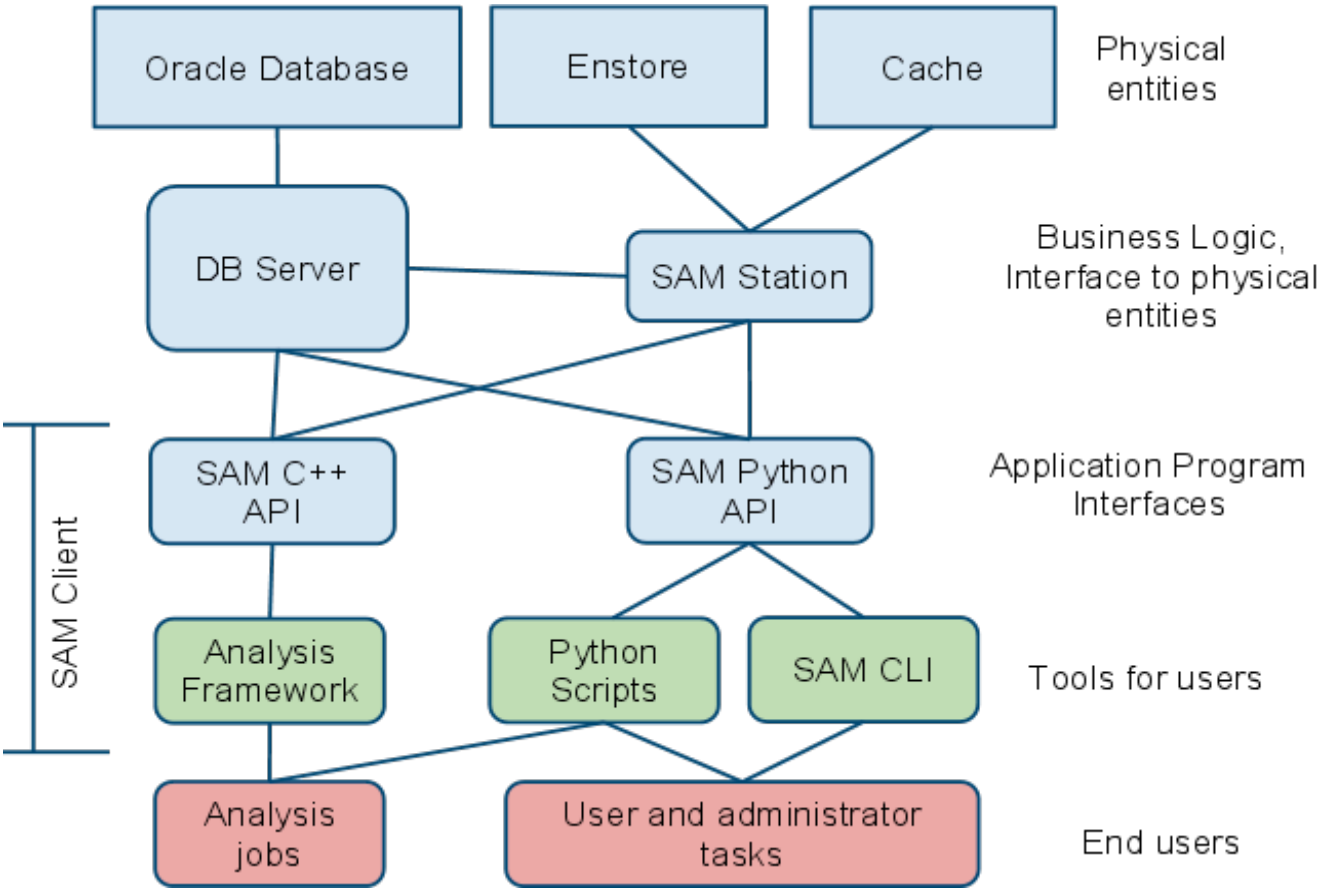
Introduction

SAM is a full featured data handing system used with great success at the Run II experiments. We believe that the features of SAM will be greatly beneficial, if not eventually crucial, for the IF experiments as well. We are now facing two difficulties with SAM: 1) Integrating SAM into an experiment's framework is not trivial and 2) the SAM interfaces presented to the users demands a learning curve and differs greatly from the typical experience with files.

For the reasons above, adoption of SAM by the IF experiments has been very slow. We wish to make changes to SAM to make integration and adoption easier. For a moderate amount of development work, we can make a large impact on SAM and its acceptance by the IF experiments.

The present layered interface

[1]Contact Adam Lyon lyon@fnal.gov for questions about this document.



The figure above displays the present layers in SAM. At the top are the physical entities: the Oracle database, the Enstore tape system, and the Cache disk servers (or a cache system like dCache). The next layer contains the low level application code that directly controls the physical entities. The DB server and the SAM station are services that contain the business logic of the system and reside on server nodes. The next layer begins the client software. We have C++ and Python APIs that provide access to the system. Under the C++ API is the analysis framework code that deals with data handling. For CDF, DØ, and MINOS, we have had significant development efforts to integrate the C++ API into the experiment's analysis framework code. Hanging off of the Python API are various python scripts (mostly used by SAMGrid) and the SAM command line interface (CLI). Finally, the end user layer includes the analysis jobs making use of SAM as well as the users and administrators doing their data handling tasks (e.g. creating datasets, checking locations of files, etc). The Python API and CLI are complicated and have a steep learning curve. Another drawback of this system is that the SAM Client code (the APIs and the code necessary for the CLI) must be deployed on any system where SAM will be used. If we make a change to the client, we must update code at all sites.

A change of philosophy

The SAM commands and interfaces are very all-encompassing, allowing a very rich set of tasks that can be performed. For example, querying the database involves a “dimensions” language so that users can input generic queries in a language easier than SQL. But in our experience at CDF, DØ, and MINOS, regular users utilize only a few functions:

- Users typically do not create their own datasets. Rather, experts in the physics groups set up datasets for common samples and advertise them on experiment web pages.
- Users very rarely make use of much of the metadata stored for a file. Experts as well use a very small subset of the available metadata to create data sets.
- The most common use of SAM is to look up a desired dataset on some experiment web page and supply that to a job submission script.

In practice, the cost of this interface is the difficulty of use, steep learning curve, and the complexity of the code behind it. Furthermore, integrating SAM with the experiment’s C++ analysis framework requires significant work. The C++ and Python APIs mean that both need to be changed when a part of the DB server or SAM station changes.

The good news is that for the most part, development on the DB server and SAM station are complete. We do not plan additional development except for bug fixes and minor improvements. This situation implies that the APIs and the CLI will remain largely unchanged for the future. But the difficulty in integration and use remain.

We propose a change in philosophy. Instead of presenting an all-encompassing interface that is difficult, present a simple interface for a limited set of tasks. That is favor ease of use at the expense of generality. The old CLI will remain so that complicated tasks can still be performed, but those tasks should be relegated to experts instead of regular users.

We also wish to make SAM more portable by reducing the need to deploy the SAM client code everywhere.

Three fronts

There are three fronts we wish to address.

1. Defining metadata queries
2. Retrieval of information for interactive use
3. Retrieval of information for batch/grid use

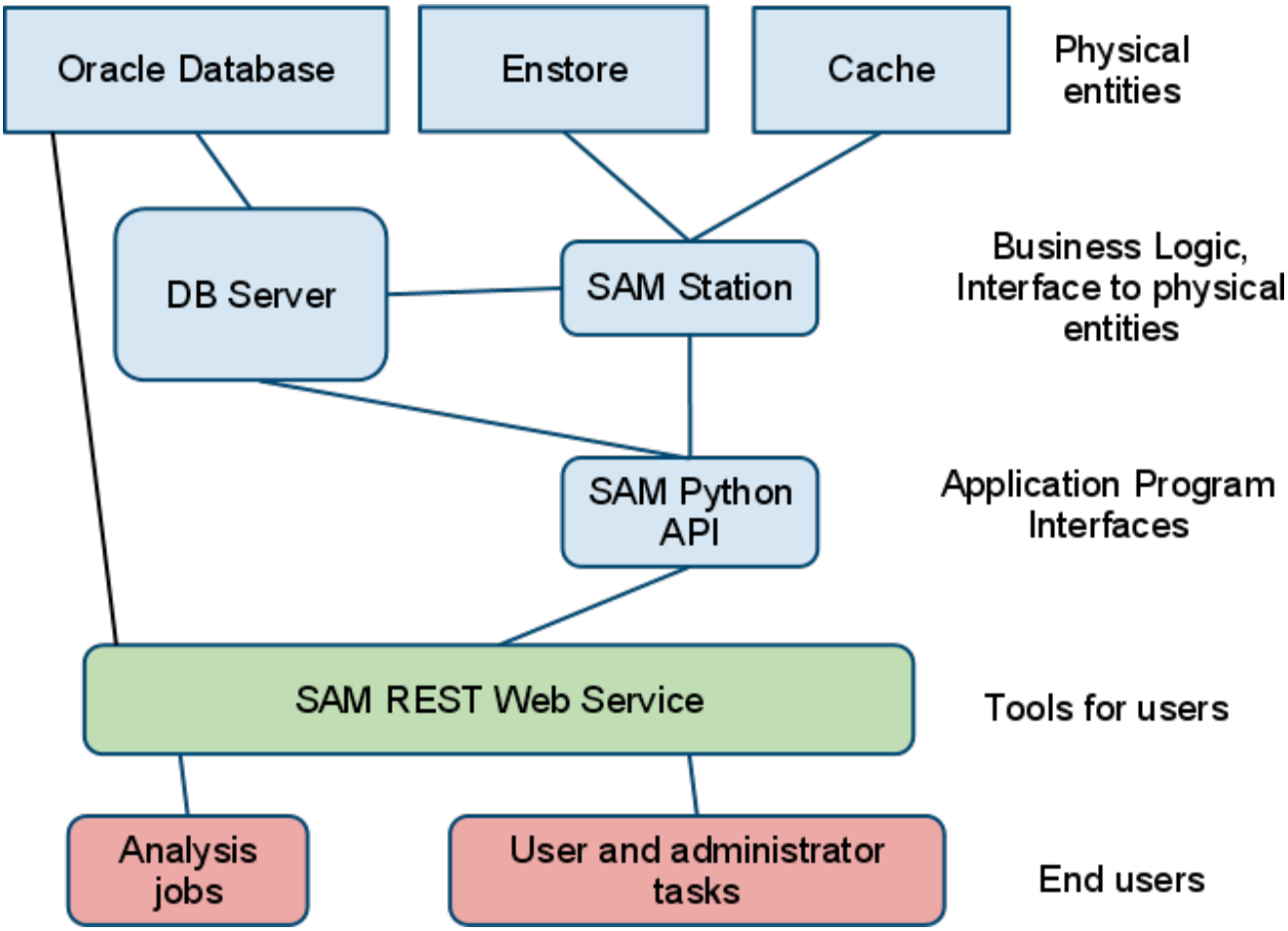
Currently, metadata queries are defined with a dimensions language. The dimensions language consists of conditional query statements (e.g. “application_name reco application_version p20.01.02 data_tier root_tree stream bla”) with many shortcuts to make it simpler than SQL. The command to do a metadata query is “sam translate

constraints". Neither the command nor the dimensions language are intuitive. Furthermore, the dimensions code is extremely old and fragile. Changing this code to add new features now is nearly impossible. The advantage of the dimensions language is that it allows complex queries involving every piece of metadata. But as mentioned, this generality is rarely needed, and the complexity it generates hinders use.

Interactive users are faced with learning a new system of commands to access the system. Granted, these commands are not particularly hard to use, but the options can be complex and the learning curve is not easy. Since SAM deals with files, it would be easier to use an interface that users already know.

As mentioned, SAM access from production and analysis applications is allowed by a specially written layer that interfaces the experiment's framework to the SAM API. Since typically experiments have different frameworks, we have had to write different interface layers -- a time consuming task. We wish to streamline to one interface layer suitable to all experiments.

The solutions



SamWeb

For majority of users, we envision replacing the CLI and analysis framework integration with a "SAM REST Web Service" layer (henceforth shortened to SamWeb). The idea here is to have a centrally located web server (and proxies if necessary) that will respond to a specific and limited set of requests via http GET, POST, and PUT calls. The responses can be text, XML or JSON (the format may be settable in the http request). Some examples of functionality of this interface are,

- Return a list of files given a pre-canned metadata query (e.g. run, subrun numbers and stream ID, MC sample ID, dataset definition name…)
- Create a SAM dataset defintion using a pre-canned metadata query
- Start a SAM project given a definition name
- Start a SAM process
- Get the next file for a process
- Release a file for the process
- Store a file (upload the file)

Given that SAM already has a Python API and there exist several long-standing well developed scalable web servers for Python, we believe that writing SamWeb is not a long-term project. Rather, it is an exercise in integrating these pieces.

The interface layer between SAM and the experiments would be simpler. The experiment code would need to make HTTP requests to the SamWeb service. There are several C++ libraries that can handle such requests (e.g. libcurl) or some frameworks may have this capability built in.

A huge advantage of the SamWeb interface layer is that the SAM client software would only reside on the SamWeb servers, not on every node that needs to talk to SAM. This change makes deploying SAM **much** simpler. If we alter the API, we only need to update the SamWeb server, not every client.

Examples of calls could be…

Return the list of files matching criteria
http://samweb.fnal.gov/listMatchingFiles?run=12334&subrun=45&stream=nu

Start a SAM project
http://samweb.fnal.gov/createProject?group=minerva&defintion=nu_123

Get next file
http://samweb.fnal.gov/getNextFile?process=1234554

As mentioned above, results could be returned in the most appropriate format.

We believe that SamWeb addresses the first and third fronts (defining queries and retrieval of information for batch jobs).

Note in the diagram that the SamWeb interface can have a direct connection to the Oracle DB. We envision new metadata queries that the current dimensions language cannot handle. As mentioned, development of the software behind the dimensions language is simply no longer feasable. As an alternative, such queries may be performed by SamWeb as direct SQL on the Oracle Database. The fact that such queries would occur on the server and not in the client is another advantage of SamWeb.

samfs

When users want to deal with files, they use the file system commands. For example, cd, ls, cp, and cat. It would make SAM much more intuitive if it too could be accessed like the

regular Linux file system.

dCache recognizes this philosophy as well. To browse the dCache catalog and identify files to access, one utlilizes pnfs, which is a remote file system mounted on the node. The server, however, is really a database and each ls triggers a query. pnfs, however, is quite limited – there is only one "path" of metadata (the destination path in pnfs - each directory piece is metadata, and there can be only one path to the file). You cannot cp or cat the files in pnfs. To access the file, you need to issue a dccp or dcap dCache access command, using the path as a file identifier.

We envision a system much more flexible than pnfs for SAM that would involve FUSE (this system is hence called samfs). FUSE implements a fully functional filesystem interface to a user program. That is the program reacts to filesystem calls, like "read directory", "get file attributes", "read file", "follow symlink". FUSE is well established code used by many applications (including cernvm-fs and sshfs). FUSE has a python binding (several actually) and so can easily be tied to SAM. This technology allows intuitive queries such as:

List files given run, subrun, data type
```
ls /samfs/run/268211/123/nubar
```

List subruns for a run
```
ls /samfs/run/268211
```

List files in a dataset
```
ls /samfs/dataset/common_nuappear_winter2011
```

Operations on files, including retrieval and open, could also be performed. Say a user wanted a specific file to test a program. That person could do,

Download the particular file (note the use of "get" in the path) to your machine
```
ls /samfs/get/run/233331/12/nubar/nubar_233331_12_processed.root
```

Open it in root (the actual location of the file is in some local cache that the user need not know)
```
root
TFile* myFile = new TFile("/samfs/run/233331/12/nubar/
nubar_233331_12_processed.root")
```

The download may be fairly fast or quite slow if the file is coming from tape. The user uses the same intuitive system.

We also imagine storing files with this system.

One of the authors of this document (Adam) put together a very crude proof of principle

FUSE sysem that does actual SAM queries (but only simulates data movement). The fact that such a system could be written very quickly and easily was impressive.

It is possible to hook samfs up to SamWeb instead of directly to the python API.

The development effort

We have yet to perform a detailed study of the software effort needed to bring these ideas to production. As a precursor, however, we have discussed within REX whether to stay on the SAM route for data handling or consider alternatives. The concerns raised were,

> There seems to be no other data handling system that provides the features and success of SAM that can also accommodate multiple experiments. Though one could imagine adopting a data handling system from the LHC. The problem here is that each LHC experiment has written their own system tightly integrated with their infrastructure. Almost none of the IF experiments plan to adopt an LHC experiment infrastructure. The exception is Minerva, which uses an old version of LHCb's Gaudi. Minerva has in fact adopted SAM for their data handling. Furthermore, CMS and ATLAS started with a different data handling philosophy than SAM - "move the jobs to the data" instead of SAM's "move the data to the jobs". "Move the jobs to the data" has turned out to have significant performance problems and management headaches, and both CMS and ATLAS have started making their DH systems look much more SAM-like.
>
> A disadvantage of SAM is that it is very monolithic, but we believe that breaking it up into pieces would require a development project that would be too expensive for the value added.
>
> SAM's metdata infrastructure may not be able to accommodate new metadata elements desired by IF experiments. In this case, we can add parallel database tables with this information and provide special queries to access (CDF does this now). The proposed SamWeb interface would make integration of these queries easy.

The conclusion was to stay with SAM and essentially freeze the core software. Some small number of additional features may need to be added, but that should be done outside of the main system, if possible.

The SamWeb idea here in particular makes integration of those new features easy. Furthermore, of we do decide to move to another data handling system, having communication occur via the web interface insulates the experiments from the DH system choice.

If the ideas in this document are accepted, the next steps are to gather requirements and write specifications. As mentioned, we believe that the development work would not be significant and the value added important. We already have the SAM python API, and we have extensive experience with python directly connecting to Oracle if we need that functionality. Also as mentioned, there exist several python web services (e.g. Twisted python) that we could use for SamWeb. FUSE already has several python bindings. So the development work is more of integrating these pieces instead of writing application and business logic.

Conclusions

We believe we can make significant improvements to the usability and deployment of SAM without mounting a lengthy software project. The use of python in SamWeb and samfs allows for easy access to the SAM client software that could sit on some centralized service node. The ability to have one experiment interface layer to SAM and providing ease of use for several of the most popular SAM functions would be a big step forward. Finally, if the full API were needed for some purpose, it could still be deployed or used at a master site.