Art and SAM integration

Adam Lyon on behalf of SCD/REX/DH and Jim Kowalkowski on behalf of SCD/CET
March 6, 2012

Introduction

Art is an analysis and reconstruction framework system used by many IF experiments. Currently, Art can read input files from a file list. For running on the Grid, Art needs to interact with SAM for file deliveries. This document describes how Art and SAM will be integrated to add this functionality and describes the tasks for each group.

Art will communicate with SAM via the SAMWeb http interface. Going this route means that SAM will not have to deploy a client. Instead, Art will make http POST calls to SAMWeb with a standard http communication library. The SAMWeb http protocol is described in detail at https://cdcvs.fnal.gov/redmine/projects/sam-web/wiki/Interface_definitions .

Interaction Timeline

A meeting was held on March 6, 2012 to work out the interaction between Art and SAM for file transfers. Those present were Kowalkowski, Paterno and Greene representing CET; Lyon, Illingworth, Mengel, and Norman representing REX/DH; and the following representing experiments: Norman (NOvA), Lyon (g-2), and Greenlee (MicroBoone).

Pre-submission

Before a user submits a job, he or she must create a SAM dataset definition [see glossary] describing a meta-data query that will resolve to the desired files. REX will provide a mechanism for creating the SAM dataset definition.

Job Submission

The user submits his or her job with the standard "jobsub" tool developed by REX. The user will specify the desired SAM dataset definition name in the jobsub call.

The jobsub script will submit a DAG (direct acyclic graph) set of jobs to condor. The first job that is run will create the SAM project. The name of the SAM project is deterministic (based on the user name, time of submission, etc) and can be constructed by other jobs without having to communicate with SAM.

Once the first job creates the SAM project and completes, the analysis jobs commence. Each job runs a job wrapper script developed by REX.

Analysis jobs

The job wrapper script constructs the SAM project name and makes an http POST call to SAMWEB's "establishProcess" (the URL of the http POST call is deterministic and based on the experiment) in order to start the SAM Process. The SAM project name is passed in as an argument in the POST call. The call returns the SAM Process ID and a "base URL". The base URL will be used for all subsequent communication to SAMWeb about this particular process (this URL is not deterministic because for scalability the SAMWeb server may delegate responsibility to other servers). With the Base URL and the SAM Process ID, Art (or another application) can construct a URL for SAMWeb calls

Note that there are some situations where the job may start more than one SAM process, where each process corresponds to a different SAM project (an example use case is pulling different types of event overlay files). This use case may not be implemented at first, but should not be difficult to add in the future. The pair of base URL and SAM Process ID defines an input file "stream".

The job wrapper script invokes Art by passing in the SAM Process ID and the base URL. An example call could be,

```
gm2 --sam-web=http://samweb-mirror12.fnal.gov:8080/project/myProject
    --sam-process=12233
```

Both of these arguments must be given to Art upon invocation.

Unless specified, modifications to Art will be performed by the CET group.

File requests within Art

At some point Art will demand a file to read. The input file loop is as follows:

From the base URL and the process ID, Art constructs the URL corresponding to SAMWeb's "getNextFile". Art performs the http POST request.

The getNextFile POST request may produce one of several results from SAMWeb:

- SAMWeb returns error code 400-499. These codes represent unrecoverable errors (e.g. the SAM Project does not exist). Art will follow its standard fatal error procedure.

- SAMWeb returns error code 500-599. The codes represent potentially recoverable errors (e.g. The DB server could not be contacted). Art should retry the "getNextFile" POST Request a maximum number of times. Once

that limit is exceeded, the error is deemed unrecoverable and Art will follow its standard fatal error procedure.

- SAMWeb returns code 202 (try again later) along with a suggested wait time. SAMWeb returns this code if it knows a file is not immediately available. Art should wait the specified amount of time and repeat the POST call.

- SAMWeb returns code 204 (no more files). This code is returned when the file pool is exhausted. Art should treat this condition as reaching the end of its file list and perform the appropriate actions.

- SAMWeb returns a URI corresponding to the file location.

The last case, returning a URI, means that the file has been delivered and Art can retrieve it. The file URI can be of types file://, gridftp://, srm://, etc. Only file:// will be used initially. Unless the file is actually delivered to the worker node (unlikely as we do not expect to have local SAM cache on each worker node), it will have to be copied from its delivery point to the local node (many experiments have tried streaming the file directly to the application, but that has never been more efficient than simply copying the file to a local location). The copy procedure may be complex and depends on the storage type (e.g. using "cpn", "gridftp", or "srm" tools).

To encapsulate the mechanisms for moving the file to the local node, REX/DH will supply an Art Service providing a method that, when given the file URL, will move the file to the local node and return a local file path (or error code) that Art can use to open the file. Art should block while waiting for this method to return.

If the move failed, Art (not the transfer service) will make an http POST call to SAMWeb's "updateFileStatus" with the argument "skipped", indicating that the file transfer failed. Art should then follow its file load failure procedure.

If the move was successful, Art (not the transfer service) will make an http POST call to SAMWeb's "updateFileStatus" with the argument "transferred", indicating that the file was transferred successfully (and allowing SAM to unlock the source location of the file). Art then processes the file normally.

Art closes the input file when it has exhausted its data, some successful condition is reached (e.g. maximum number of desired events), or there was a processing failure. Art then makes an http POST request to SAMWeb's "updateFileStatus" with an argument of "consumed" if the processing was successful or "skipped" if the processing failed.

Art now calls SAMWeb's "getNextFile" as described above, and the loop continues.

Processing completion

At some point, Art will complete its processing and exit with success or a failure code.

Returning results to the user

Art may produce output files as a result of processing. When an output file is closed, Art will call a Service written by REX/DH with a method for notifying it that an output file is available for transfer and its path. Art will block while waiting for the service method to return. The service method should in most instances return almost immediately, deferring transfer to later or delegating transfer to another process. Art will never delete output files it has created.

Art will also call the service when it opens and closes an input file. The idea here is that the service can then keep track of which input files correspond to each output file (while it is agreed that this mechanism is less error-prone, REX/DH would rather have Art itself keep track of the input file to output file correspondence and fill the meta-data accordingly).

REX/DH needs to determine how this Service will work. Options include notifying another process on the machine and transferring the output file in parallel with the Art processing, or appending the output path to a list and then moving the files after Art processing completes.

Job completion

When the output files are transferred to their destination, the job wrapper script makes an http POST call to SAMWeb's "setStatus" with an argument of "completed" if the job was successful or "bad" if this job would need recovery.

The analysis job ends. When all of the analysis jobs end, a final job is started to end the SAM Project with a call to SAMWeb's "endProject".

Notes:

The SAM Process ID would go into the output file meta-data (not the project name)

Output file Meta-data

We need another meeting on this topic.

Glossary

DAG: DAG stands for Direct Acyclic Graph and is a Condor feature that allows for dependencies in job scheduling. For example, with a DAG one can ensure that job A runs to completion before jobs B, C, and D start and then job E starts only when the previous jobs all complete.

SAM Dataset Definition: The SAM dataset definition is a meta-data query that resolves to a desired set of input files for processing. Dataset definitions may be created by SAM client tools on the command line or with experiment specific web based tools. The actual list of files that satisfy the meta-data query is called the SAM snapshot.

SAM Process: The SAM process represents a job that demands files from SAM. The job starts the SAM process by specifying the parent SAM Project name. Using this mechanism, a collection of jobs can work in parallel and pull files from the Project (SAM ensures that a particular file goes to one and only one process). It is possible for a job to start more than one SAM process under the same Project, but this has little utility as SAM already has file prefetching capabilities and forcing such behavior with multiple processes in the same job in general does not improve efficiency. A job may, however, need to pull files from more than one pool (e.g. different types of overlay events). In this case, it is reasonable for a job to start multiple processes under different projects. Projects, in this example, would represent pools of different types of overlay events.

SAM Project: The SAM project represents the pool of files available for delivery (e.g. the SAM Snapshot) and the collection of SAM Processes pulling files from that pool. The Project is typically started by specifying the SAM dataset definition name, and SAM snapshot is created at that time. The name of the SAM Project is also specified. Internal to SAM, a Project is actually a unix process called “pmaster” running on the SAM station node. The pmaster responds to requests from SAM processes and coordinates the delivery of files particular to the Project.

SAM Snapshot: The SAM snapshot is a list of files that satisfy a SAM dataset definition at a particular point time.

SAM Station: The SAM station is an application that coordinates all communication between SAM projects, processes and the SAM database as well as all file delivery activities.