DØ Grid Production Architecture

Adam Lyon
May 2007

1 Introduction

This document describes the architecture of the DØ Grid Production system, mainly comprising of `D0ReproTools`, `SAMGrid`, and `D0RunJob`. The architecture of the currently running system is described and deficiencies are identified. Subsequent sections detail possible improvements to the architecture with my commentary on the positives and negatives of each change.

The main goal of this system is to provide a mechanism for reprocessing and Monte Carlo production on offsite computing clusters. In the beginning, the system used native SAMGrid sites, forming a DØ grid. Recently, the system has been adapted to treat the OSG and LCG grids as additional clusters. We are migrating to expand the use of OSG and LCG and limit the instances of native SAMGrid clusters.

The DØ Grid Production has been very successful for DØ. The p17 and p20 reprocessing tasks were performed as well as a refixing task (about 3.5 billion events processed total). The system has been in use for MC production for several years and produces millions of events per week.

DØ has proposed expanding the capabilities of the system to encompass more applications and additional production tasks that are currently performed by hand on the analysis farm. The current architecture of the system makes adding additional functionality difficult.

2 Current Architecture

The DØ Grid Production system may be divided into three main categories: Request tracking and submission, Grid and Fabric interfaces, and Workflow. A picture of the architecture is shown in Fig. 1.

2.1 Request Tracking and Submission

A "request" means that a specific type of processing is meant to be run (*e.g.* reprocessing, Monte Carlo Generation) with a corresponding set of parameters or metadata describing how processing should proceed. The metadata may include items such as the application(s) to run, a SAM dataset to run over, parameters for the Monte Carlo generator, *etc*. A request may also be in certain states. A new request may require an expert review to ensure it is correct. An approved request may be waiting in a queue

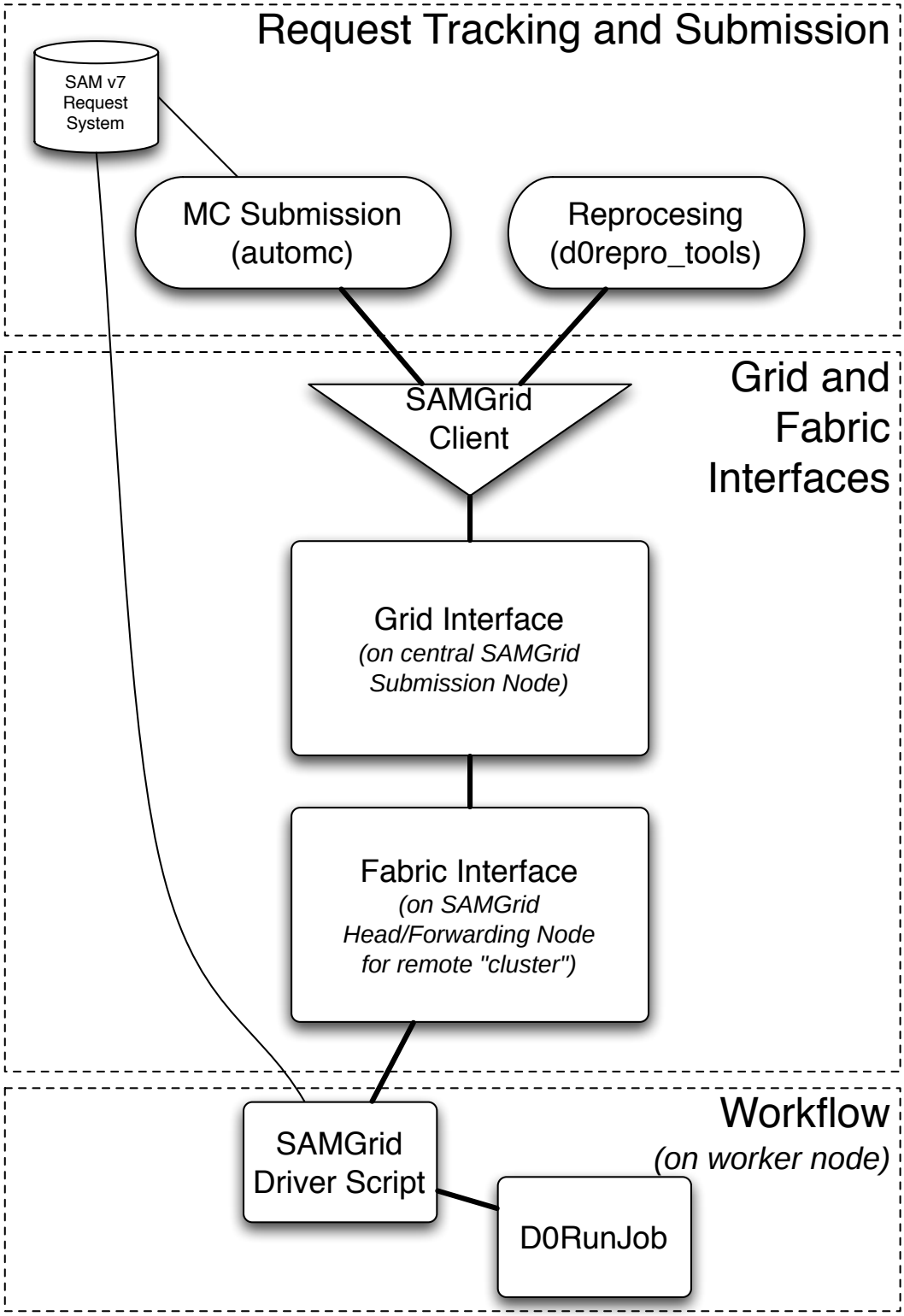

Figure 1: The current DØ Grid Production Architecture

to be run. A held request may have some problem with the metadata or may require resources that are not currently available. A running request may eventually complete or fail. Requests are transformed into SAMGrid jobs to be run by the Grid system. Jobs that suffer failures may need to be recovered. The logic of all of the above is handled by the Request Tracking and Submission Systems. Right now there are two such system. The Reprocessing effort uses `d0reprotools` while MC production uses `automc` in conjunction with the SAM v7 Request System Database.

2.1.1 SAM v7 Request System

The SAM v7 Request System handles associating metadata with a request, holding the state of requests, and associating requests with the files they process and produce. The close association of requests and files makes the SAM database a natural place to store such request information. In SAM v5 there were two request systems in the DB, one for primary reco production (whose author was long gone) and another for MC requests. These systems were merged into one and additional capabilities were added. The v7 Request System assigns an ID number to each request and metadata for the request are stored in the regular SAM keyword – value pair parameters database tables. There are command line and Python interfaces for creating requests, setting and viewing the metadata, and changing the request state. The Workflow part of the D0 Grid Production System uses the SAM Request System to get parameters for its applications. Currently, only MC Production uses this system.

Using SAM commands, users can search for requests meeting their criteria (*e.g.* requests to generate Monte Carlo with a specific set of $WW\gamma$ anomalous coupling values). Users can also easily create datasets of files that were produced by particular requests.

The SAM v7 Request System is maintained by the SAM Team (currently, Steve Sherwood is the active developer, though aside from minor feature requests, this system is completed and is in production).

2.1.2 Automc

`automc` is used by the MC Production team to submit SAMGrid jobs based on approved and ready requests in the SAM v7 Request System. `automc` polls the Request System to find new requests ready to run, creates the job description language (JDL) files for those requests (translating a request into a SAMGrid job), submits the SAMGrid job(s) changing the state in the Request System to “running”, and waits for the jobs to complete, setting the “Complete” or “Failed” state in the Request System accordingly.

`automc` is maintained by the MC Production Team. Joel Snow is the active developer.

2.1.3 DØ Repro Tools

`d0reprotools` is used for Reprocessing. It does not (yet) use any request system. Since Reprocessing jobs have little metadata (mostly just the dataset to run over), there is not

much need for a repository of parameters. But the Reprocessing request state (which to run next, is it finished) must be tallied by hand.

The main purpose of `d0reprotools` is to provide an easy mechanism to submit and track the success (or failure) of Reprocessing jobs. It creates the JDL file from the job parameters (*e.g.* dataset to process) and submits the jobs to SAMGrid. It also checks the status of the jobs and looks for failures (by watching what SAMGrid returns and looking at the produced files [or lack thereof if problems]). `d0reprotools` can also create a recovery job for failures being careful not to duplicate successfully processed events.

`d0reprotools` is maintained and developed by Daniel Wicke.

2.2 Grid and Fabric Interfaces (SAMGrid)

The Grid and Fabric Interfaces involve the handling and processing of jobs to be run on some cluster by interfacing with standard grid tools as well as, in some cases, interfacing with tools on the cluster itself. This part of the system is more commonly known as SAMGrid or JIM (for Job Information and Management).

There are three parts of the Grid and Fabric Interfaces: client software allowing users to interact with the system (*e.g.* submit SAMGrid jobs), a Grid interface to submit the SAMGrid job to the particular cluster or grid, and a Fabric interface to interact with with resource management systems and tools (*e.g.* batch system, scratch storage) on the cluster itself.

SAMGrid has application specific code in it and only works for certain applications.

All software mentioned here are maintained by the SAMGrid team with Gabriele Garzoglio, Parag Mhashilkar, and Andrew Baranovski as the main developers.

2.2.1 SAMGrid Client

The SAMGrid client (`samg`) is command line tools for the human (or machine) interface into the system. Users can submit SAMGrid jobs by specifying the JDL and perhaps an input sandbox. The client software may in principal be installed anywhere so long as it can communicate with a SAMGrid submission node that handles the Grid interface.

In practice, it is rare for an actual person to interact directly with the SAMGrid client. Usually, the interaction is handled by the Request Submission and Tracking software discussed above (`d0reprotools` or `automc`).

2.2.2 Grid Interface (Submission Node)

The Grid Interface is software running on a “SAMGrid Submission Node”. The purpose of this software is to submit the job with its relevant metadata to a “cluster”. Cluster is used loosely here. In the days before mature grid tools, DØ grouped clusters together to form its own DØ Grid. Each remote cluster had a head node with SAMGrid software for interfacing to that cluster’s specific tools (the Fabric – see next section). Such clusters

are now called "SAMGrid Native". There are still some SAMGrid Native remote sites with their head nodes, but now the emphasis is on using the Open Science Grid (OSG) and LHC Computing Grid (LCG). As an extension to the remote cluster idea, we treat the OSG and LCG themselves as clusters, each with its own forwarding node (same as a head node) that knows how to communicate with that grid's protocols and tools (*e.g.* that grid's fabric). Thus, the purpose of the Grid interface and the submission node is to transport the job to the appropriate head or forwarding node so that it can be sent to that cluster's fabric.

The Grid Interface interacts with a SAMGrid Broker to determine the destination cluster for an incoming SAMGrid job. In many instances, the destination must be manually specified in the job's JDL. The Grid Interface software then forwards the job to the destination cluster's head or forwarding node. The Grid Interface software periodically tracks the progress of the job and monitors completion. Log files and such produced by the job are returned to the submission node and kept for a short time for the user.

2.2.3 Fabric Interface (head or forwarding node)

As mentioned above, the purpose of the Fabric Interface is to communicate with the tools and protocols of a particular cluster or grid to run a SAMGrid job. The Fabric Interface software running on the head or forwarding node consists of the Globus Gatekeeper, the SAMGrid Job Manager, and SAMGrid Batch Adapters.

The Globus Gatekeeper receives and authenticates the SAMGrid job from a SAMGrid Submission Node (Grid Interface) using standard grid protocols.

The Job manager then translates the SAMGrid job into local jobs suitable for the cluster (a local job is the execution unit that runs in the cluster's batch system). This process is currently very application specific. Depending on the job type specified in the JDL, the SAMGrid job may be translated into one or more local jobs. For example, a SAMGrid job for a Monte Carlo Generation request translates to a local job per 250 events requested (*e.g.* a 1000 event request is turned into four local jobs). A Reprocessing SAMGrid job is translated into one local job per file to be processed. The job manager also adds additional information to the JDL to specify the data queue to use for jobs processing files out of SAM. Multiple data queues are attached to SAM server node to increase the efficiency of data transfers. The selection of a data queue depends on the application type and in some cases the cluster location itself. For example, clusters with known poor networking connectivity will use certain data queues to avoid slowing down jobs running at faster sites. Different applications have different data access requirements, and so organizing data access in different queues by grouping applications with similar data access patterns minimizes the time spent waiting for data staging.

If required, the Fabric Interface software will also start a SAM Data Handling project with the appropriate SAM station. The Project Master process resides on that head or forwarding node and handles all requests from the running application instances for more files. Each project master instance requires machine resources and thus is one of the factors that limits the number of running jobs a head or forwarding node can simultaneously service.

Once the number of jobs has been worked out and the data queue selection has been added to the JDL, if necessary, the jobs are then submitted by the job manager to the cluster's batch system with the SAMGrid Batch Adapters. The batch adapters abstract the interaction with different batch systems (condor, LSF, PBS) and different grid submission systems in the case of OSG and LCG (a triumph of SAMGrid is the idea of treating the OSG and LCG grids as batch systems). The job manager then monitors the progress of the batch jobs and returns output and exit information back to the submission node.

2.3 Workflow

The Workflow is the execution of code running in the job on a cluster's worker node. The workflow consists of a SAMGrid driver script and the `d0runjob` software. `d0runjob` is an organized collection of scripts that know how to run specific DØ applications and perform data handling tasks such as storing new files back into SAM.

The SAMGrid driver script is executed by the batch system when the job starts. This script is maintained by the SAMGrid team. It is responsible for obtaining the executable tar-ball from some common location, setting up the environment for the applications, and then calling the appropriate `d0runjob` scripts to actually run the applications. Setting up the environment means moving needed auxiliary files to locations expected by the applications and extracting parameters from the Request System database to pass to `d0runjob` scripts. The SAMGrid driver script then calls the appropriate `d0runjob` scripts. All of these steps depend on the job type passed through the JDL, and thus is inherently application specific.

The `d0runjob` scripts execute the desired DØ programs to process data. Metadata are created for the files produced by the processing and those files are stored back into SAM either to tape or to durable locations for intermediate files.

`d0runjob` is maintained and developed by Peter Love and Michel Jaffre.

2.4 Deficiencies in the Current Architecture

There are several deficiencies in the current architecture that make it difficult to add new features. Furthermore, the fact that `d0reprotools` does not use the SAM v7 request system means that non-MC processing requests must be tracked by hand.

There are many instances mentioned above where the Fabric interface needs to know application specific information, made possible by including the job type in the JDL. Specifically, the job type is used to determine,

- the number of local jobs to run for a SAMGrid job,
- whether or not a SAM project is necessary for the jobs,
- which data queue to use.

Furthermore, the SAMGrid driver script also uses the job type to perform tasks necessary to setup the environment for `d0runjob`.

Having application specific code in SAMGrid means running new applications leads to new job types being defined in many places of the code managed by different groups. In an ideal world, SAMGrid should know nothing at all about the application to run and any decisions that must be made in the Grid or Fabric interfaces should depend on parameters in the JDL. Furthermore, the SAMGrid driver script should be very thin and application generic with the thick DØ specific code consolidated in a stronger `d0runjob`. While these changes are desirable, they add to the maintenance burden of the components above and below SAMGrid. While this situation should make the overall system easier to maintain, it is not immediately clear if it is possible to realistically achieve that outcome.

3 Possible Improvements to the Architecture

In this section I list possible improvements to the architecture. The goal is to more sharply define the borders of the software components to leave the Grid and Fabric Interfaces as application generic as possible.

Note that I do not discuss changes beyond making components work better together. Certainly there is much to do make each component itself work better. For example, there are tasks to improve the automation of job submission and tracking, making SAMGrid components more reliable, and making `d0runjob` handle more applications and MC generators.

3.1 `d0reprotools` Uses the SAM v7 Request System

Right now, the SAM v7 Request System is not used for non-MC jobs, so the work of tracking requests must be done by hand. In order to utilize the Request System, `d0reprotools` must be significantly improved. The advantages of using the Request System are,

- Request status information are held in a DB and can be easily queried,
- Application parameters are also stored in the DB for easy query and checking,
- It is easy to determine the files produced by a particular processing run. This capability is important for certification, accounting, and debugging,
- We will have one Request System in common for all production tasks.

The disadvantages are significant changes to `d0runjob` as well as modifications may be needed for the Request System itself, since it was written with input mostly from the MC group.

I believe that this improvement is important for future production and should improve the process of Reprocessing and primary-processing on the Grid.

As far as I know, while this improvement is known to the d0reprotools developer, no progress has been made.

3.2 D0runjob Sets Up Its Own Environment

As currently implemented d0runjob is really a collection of scripts. The SAMGrid driver script executed by the batch system does all of the work of setting up the application environment (this task includes moving auxiliary files to the right directories and extracting application parameters from the Request System) as well as executing the correct d0runjob scripts. This situation makes the SAMGrid driver script a thick system with hard coded application specific decisions. Adding a new application means changes to d0runjob as well as the corresponding changes to this driver script. In a more ideal world, the SAMGrid driver script should be very thin with all application specific code embodied in d0runjob.

Making this improvement is a significant change the d0runjob software component. Ideally, there would be one entry script called by the SAMGrid driver script. The latter would pass in the parameters specified in the JDL. One of these parameters would have to be some specification of the application type or category so the d0runjob entry script would know what to do, but this parameter would not be used by the SAMGrid driver script. The d0runjob entry script would then extract necessary information from the Request System DB, further specifying particulars of how the applications are to be run. d0runjob uses that information to setup the environment and run the applications. This improvement leaves a very thin SAMGrid driver script and a very thick workflow system, which an appropriate architecture.

The main advantage of this improvement is that all of the application specific code necessary to run the application rests in the same code base and is managed by the same group. There would need to be communication between the d0runjob group and the Request tracking and submission groups (automc and d0reprotools) to ensure that the JDL parameters and the Request System information are sufficient for setting up and running the applications. In principle, this communication must exist already, with the difference here being that close communication with the SAMGrid group would no longer be necessary when adding a new application. The SAMGrid driver script no longer needs to change at all when adding or changing an application.

The disadvantage of this improvement is the added burden on the d0runjob team. Currently, the SAMGrid team acts as the integrator of the three components and does final integration testing. In this improved scenario, communication with the SAMGrid team would not be necessary as often, and so the d0runjob team may be responsible for integration testing. In fact I view this change as good, as application specific development, testing, and debugging would rest mainly with the same group.

There are changes necessary to the SAMGrid driver script in order to remove the application specific code and to call the d0runjob entry script.

3.3 The Fabric Interface Does Not Depend on Job Type

Currently, the Fabric Interface software makes many decisions based on the job type JDL parameter, for example the number of local jobs for the SAMGrid job as well as selection of the data queue. In principle, these decisions may be made at the request creation level and passed as JDL parameters. For example, the number of local jobs and the data queue name may be passed explicitly in the JDL. The Fabric Interface software would simply act according to those parameters without making any decisions.

The advantage of this improvement is that the Grid and Fabric Interfaces are truly application generic. A new application can be added with no changes to the SAMGrid code running on the head or forwarding node.

The disadvantage is that Grid policies and configurations must be enforced by the Request System, which is perhaps not the appropriate place. For example, the choice of data queues is purely an optimization for running different applications on the Grid, and perhaps the best place for that decision is in the Grid and Fabric interfaces where policy is set by the SAMGrid team. Moving those decisions outside of SAMGrid would require extra communication between the SAMGrid team and the production team. Indeed it may be impossible to move all of these decisions outside of the head/forwarding node, because some choices depend on the computing element location. For example, one choice for data queues depends on the network connectivity of the computing site, and that information is not known until that site is chosen by the Grid interface. Therefore, if this improvement is in place, there would be a need for some override mechanism so that Grid policies that can only be enforced when the job is already in the system are enforced there instead of before the job is submitted. This could be a complicated system.

Currently, the choice of the number of local jobs for an MC SAMGrid job is set by MC Group policy. This parameter could be sent in the JDL instead of set at the Fabric Interface layer. If, however, we decide to make this parameter dependent on the computing site (maybe more reliable sites can produce more MC events in one shot), then this scheme will not work.

My feeling is that while this improvement is ultimately desirable, it must be thought through very carefully. This problem can be attacked piece-meal with JDL parameters added one at a time. Ignoring the caveat in the previous paragraph, one could start with the local jobs per SAMGrid job for MC production change. I feel that the data queue selection is too complicated to do outside of the Fabric Interface software. So long as the job types are broad enough, it should be possible to add new applications with only minimal changes to the Fabric Interface software.

3.4 All Merge Code in `d0runjob`

Currently, the code for merging is almost entirely performed by the SAMGrid driver script with `d0runjob` used only to run the merge application. The SAMGrid driver script is filled with logic determine the metadata for the new merged files. In principle, this code should reside in `d0runjob`, which already determines metadata for files stored by

other applications. In an ideal system, the SAMGrid driver script would simply call the `d0runjob` entry script (see section 3.2) and `d0runjob` would perform the merge task.

I would imagine that putting the merging code into `d0runjob` would be fairly straightforward. This change would obviously move the responsibility for merging from SAMGrid to the `d0runjob` developers, increasing their support burden.

4 Conclusions and Recommendations

An improved DØ Grid Production System would have nearly the same architecture as the current system, but with each component performing better defined tasks. Most notably, the SAMGrid Grid and Fabric Interfaces and the SAMGrid driver script should be as application neutral as possible. Application specific code currently in the SAMGrid Fabric Interface could be moved up to the Request tracking and submission systems. Code in the SAMGrid driver script could be moved down into `d0runjob`. While making such changes increases the support and development burden on the other components, having an application neutral SAMGrid would make adding new capabilities much easier than it is now.

My recommendations are:

- Change `d0reprotools` to use the SAM v7 request system (see section 3.1). Using the already existing SAM v7 request system will improve the organization of tracking reprocessing requests as well as primary-processing. This change will also put `d0reprotools` on par with the MC `automc` tool so that all aspects of production will use the same Request System.
- Change `d0runjob` to set up its own environment (see section 3.2). I believe that this change alone will significantly improve the time needed to add new applications to the system, so long as the `d0runjob` team can support this extra burden of development and integration testing. Indeed, much work has already been done on this step.
- Push job type dependence out of the Fabric Interface (see section 3.3). This improvement is probably the most difficult to implement, but can be done piecemeal. Removing application dependence in the Fabric Interface means that new applications may be easily added, but we risk adding complexity to the Request Submission systems. Such changes spreads where Grid policies are implemented to places where perhaps they do not belong and could make site dependent decisions difficult. If any action is to be taken on this improvement, I recommend moving the number of local jobs per SAMGrid job up into the JDL (set by the Request Submission system) as a good first (and perhaps only) step.
- All merging code in `d0runjob` (see section 3.4). This change puts the merging code in its appropriate place, but adds a new burden to the `d0runjob` developers. I think this improvement is important, but perhaps not at the highest priority.

Such improvements should simplify the work needed for the individual component improvements requested by DØ and for increased stability.