The SAM-Grid Fabric Services

G. Garzoglio[a], I. Terekhov[a], A. Baranovski[a], S. Veseli[a], L. Lueking[a], P. Mhashilkar[a] and V. Murthi[a]

[a]Fermilab, Batavia, IL

To enable globally distributed computing for a large HEP experiment, a collection of computing and data storage facilities, together called the Grid fabric, must be linked together in a coherent way. The standard Grid software, including most notably the Globus Gatekeeper and Meta Directory Service, provides core tools to insert a site into the Grid and for its low-level monitoring. In practice, large experiments have data and job handling infrastructures that are not governed by the core tools. For example, local job submission is seldom done directly to the batch system, but rather, through an interface that allows for pre-submission steps (such as the decomposition of a job into smaller chunks) or is tightly integrated with a data handling system such as SAM. Likewise, monitoring is seldom done in terms of individual processors or individual jobs, but rather, via cluster-wide aggregated characteristics. In this paper, we present some of the work we have done to abstract the management of the fabric facilities of the FNAL Run II experiments, in order to enable globally distributed computing.

1. INTRODUCTION

The goal of the SAM-Grid project [1–5] is to provide fully distributed computing for DZero and CDF, the two experiments at Fermilab currently collecting large datasets during the Run II of the Tevatron Collider. For about the first 18 months of the project, we have concentrated on the integration of standard grid tools, such as the Globus Toolkit [6] and Condor-G [7], with SAM [8–12], the data handling system of the experiments. The subsequent phase of deployment to the collaborating institutions has proven to be unexpectedly difficult[13], mainly for the integration of the Grid infrastructure with the local resources: the Fabric. We believe that the principal cause of the problems was the lack of "standard" services (see also [14]), such as local storage management, sandbox management or batch system adaptation mechanisms. The SAM-Grid has overcome these problems by implementing the needed local services. In this paper we describe these services.

2. LOCAL BATCH SYSTEM ADAPTATION

After our first experience deploying the SAM-Grid, we concluded that the "standard" batch system interfaces implemented in the Globus Toolkit are not flexible enough to include most of the resources of DZero and CDF. Even at sites running the same batch system, we observed that different administrators frequently configure the batch system differently because of local constraints, thus requiring the local users to submit jobs using slightly different commands. In some other cases, the terms of the agreement to use the resources could be respected only by adding special attributes to the job submission request. For example, when DZero submits monte carlo production jobs at the condor cluster of the University of Wisonsin at Madison, the job description file must include special attributes to prevent job eviction. Another example is running monte carlo jobs at the IN2P3 computing facility in Lyon, France: the BQS batch system is locally configured to let the job overcome the downtime of their local mass storage system, HPSS, only in case the job is submitted with a special option. These attribtues and submission options are all non-standard and site specific.

In addition to the site-specific peculiarities of the batch system, for the typical high energy physics application, job submission is seldom done directly to the batch system, but rather, through experiment-specific local interfaces that take care of the job preparation. Such preparation steps include the triggering of data handling systems such as SAM, the use of local job sandboxing mechanisms, or the decomposition of the job into smaller tasks, generally executed as parallel instances.

In order to address this concern, the SAM-Grid team has developed a series of job management scripts that use the experiment-specific interfaces. These scripts are invoked via standard grid mechanisms, such as the Globus Gatekeeper. From within these scripts, the invocation of the local batch system commands is done via an intermediate layer that abstracts the basic interactions with the batch system. This layer is configured locally with the specific commands used for job submission, look up and cancellation. The batch system adapter is also configured to interpret the output of the batch system commands, to enable the extraction of relevant information, such as the local job id after submission, the status of the job after lookup or the error messages after any command invocation.

This additional level of indirection in the configuration of the local job management has been proved of fundamental importance during the phase of deployment. Despite its usefulness, though, because of its flexibility, the configuration of the batch adapter is still considered by our collaborators one of the most challenging parts of the configuration procedure and it is, in practice, left to the SAM-Grid system experts.

3. DYNAMIC PRODUCT RETRIEVAL

The portability of the code is the key element for the execution of jobs on the grid. At DZero, tools have been developed that address the problem of recreating the Run Time Environment (RTE)[15] of the typical applications. These infrastructures give the users the ability of packaging their programs with all the software dependencies, thus enabling the execution in rather "hostile" computing environments. There are certain classes of jobs, though, that use standard applications driven by user specified configuration parameters, which require little or no user-provided code. These applications can be quite large (on the order of a few Gigabytes, even when compressed) and packaging them for every job would be quite costly. These costs add up, considering that every such archive needs to be transported and temporarily stored possibly in more than one place throughout the lifetime of the job.

On the other hand, experiments also maintain dedicated clusters, configured to provide access from any worker node to any standard experiment products. When running on these clusters, users just need to provide their custom version of the code or the configuration files to run the standard applications. Administrators are responsible for installing and maintaining these products, trying to compromise between the available disk resources and the users' requests to run on a large span of product versions. This model is quite expensive for the administrator and does not let the users take advantage of non-dedicated clusters that may be available to the community using RTE mechanisms.

Within the SAM-Grid, we use a hybrid model that promotes the advantages of the two approaches. Many commonly used applications, such as the Monte Carlo production products, which are made portable via RTE techniques as in the first model, are stored into the data handling system of the experiments. The clusters that are part of the grid are configured to provide a local data cache, managed by a SAM data handling service. Users who run jobs that use these standard applications can specify them in the Job Description File as dependencies, as in the second model. After the job has entered a cluster, the middleware is responsible for the delivery of the dependencies to the worker nodes, before passing control to the user's application. This approach has many advantages. First, products are no longer installed and maintained by the system administrator, but rather, brought into the data handling cache and installed upon request. Therefore, the size of the software provided directly by the user can be rather small. Second, since the cache is automatically managed, there is no longer a maintenance concern for the availability of older software versions at a site. Third, applications and their usage can be thoroughly catalogued using the metadata mechanisms of the SAM data handling system. Fourth, this mechanism can be used in conjunction with user-provided RTE executables, providing a high degree of flexibility in running the jobs.

4. LOCAL SANDBOX MANAGEMENT

When submitting a job to the grid, a user is required to supply a description of the characteristics of the job, such as application name, product dependencies, optional input dataset, etc., and/or an archive containing the software and configuration files needed to run the application. This archive is sometimes referred to as the input sandbox. The output of the job, such as the error and output streams, relevant log files and output files is sometimes called the output sandbox. It should be noted the SAM-Grid handles the large input and output data files and potentially the product releases via the SAM data handling service, hence dramatically reducing the size of the sandboxes, since these files do not need to be part of them.

The standard grid tools implement a protocol, GRAM[6], that allows the transport of the sandboxes and the submission and monitoring of the job. Nevertheless, there are no standard tools to manage the sandboxes at the local cluster. Ideally, the sandbox management should be able to rely on a local storage service, with a well-defined interface to, at a minimum, store, retrieve and remove input and output files from anywhere within the cluster. Even if implementations of local storage services are available [16,17], they are not very widely deployed and generally considered non-standard. Instead, what is generally provided at the cluster is disk space accessible from the gateway node, and some mechanism of intra-cluster communication. This is achieved at every site using a wide variety of different strategies. Typically, either nodes use a common network file system, or the batch system is configured with some form of input file stage-in/output file stage-out, or the nodes have access to an open network. Each of these strategies is not a general solution and each has weaknesses. For example, the typical network file systems used, NFS, has scalability problems, especially during writes; stage-in/stage-out often can only be triggered from special places within the cluster and at certain times only, such as e.g. from the head node at the time of submission; open networks are becoming less and less popular, considering the proliferation of site firewalls.

Considering that no standard local storage service exists today, the SAM-Grid sandbox management infrastructure, instead of trying to adapt to all possible different cluster configurations, starts up dynamically a gridftp server, hence guaranteeing a uniform intra-cluster transport mechanism. Input sandboxes coming from the grid are kept compressed at the gateway node in a disk area unique to the job. For every job, the infrastructure creates a self-extracting archive, containing the user's delegated credentials, the gridftp client and the directives necessary for the delivery of the input sandbox and the dependent products. It then submits an appropriate number of parallel instances of the job, using the batch system adaptation mechanism described above, relying only for this first executable on the native intra-cluster transport mechanism. At the worker node, the environment of the job is recreated and the control passed to the user's application. After the execution has terminated, before cleaning up the scratch space, the custom output is packaged and transferred to the gateway node, in order for it to be bundled together with the output of all the other job instances, and sent back to grid. The SAM-Grid makes this output bundle available to the user for download from the web.

5. JOB COMPLEX-STATUS LOGGING

As discussed in the paragraphs above, a job submitted to the SAM-Grid is decomposed in an appropriate number of parallel instances at the remote execution cluster. This hierarchical relationship is in principle even deeper, considering that by design a single grid job can be partitioned to run concurrently at multiple remote sites. Being able to monitor and log the complex status of the job, i.e. at the granularity of the single job instance running at a certain node of a cluster, is of crucial importance. To present a uniform view of the job status, we need a representation independent from the status given by the particular type of batch system. The schema representing the job status should also be flexible enough, to allow the addition of extra information, as our understanding of the associated relevant metrics evolves with time. Moreover, in the spirit of the grid, we promote a distributed logging architecture, where every local monitoring service is remotely accessible.

To conform to the considerations above, noting that the hierarchical structure of the jobs is naturally represented in XML format, our job status logging service is based on XML databases deployed at every execution site. Each database is accessible from anywhere outside and inside the cluster. This makes possible the central gathering and summary of the status of the jobs in the whole grid [18]. In addition, applications can use this service to publish internal details of the jobs. For example, the job flow manager of the experiments [19], which is typically used for the production of simulated (Monte Carlo) events, uses this mechanism to make available the current stage of the production chain of every job instance. This is possible because of the schema flexibility of the XML database, and the ability of pushing information into the system.

6. CONCLUSIONS

The SAM-Grid project provides Grid and Fabric level services for job, data and information management. We believe that the general understanding of the services necessary at the Fabric in order to seamlessly interface to the Grid is lagging behind with respect to other topics in Grid computing. We had a first hands-on experience with these limitations and we have overcome them implementing the Fabric-level services described in this paper.

REFERENCES

1. SAM-Grid project: http://www-d0.fnal.gov/computing/grid
2. "Grid Job and Information Management for the FNAL Run II Experiments"; proceedings of CHEP 03; I. Terekhov et al.
3. "SAM-GRID: A System Utilizing Grid Middleware and SAM to Enable Full Function Grid Computing"; proceedings of Beauty 02; R. Walker et al.
4. "The SAM-GRID project: architecture and plan."; proceedings of ACAT 02; G. Garzoglio et al.; NIMA14225, vol. 502/2-3 pp 423 - 425
5. "Meta-Computing at D0"; proceedings of ACAT 02; I. Terekhov et al.; NIMA14225, vol. 502/2-3 pp 402 - 406
6. Globus Toolkit: http://www.globus.org
7. Condor: http://www.cs.wisc.edu/condor
8. SAM project: http://d0db.fnal.gov/sam
9. "D0 Data Handling"; proceedings of CHEP 01; V. White et al.
10. "SAM Overview and Operational Experience at the D0 experiment"; proceedings of CHEP 01; L. Carpenter et al.
11. "Resource Management in SAM and the D0 Particle Physics Data Grid" "; proceedings of CHEP 01; L. Lueking et al.
12. "The Data Access Layer for D0 Run II"; proceedings of CHEP 00; L. Lueking et al.
13. "Experience using grid tools for CDF Physics"; proceedings of ACAT 03; M. Burgon-Lyon et al.
14. "GRID2003 Lessons Learned", PPDG Document 37, http://www.ppdg.net
15. RTE project: http://www-d0.fnal.gov/~ritchie/CPBdemo.html
16. Disk Farm project: http://www-isd.fnal.gov/dfarm
17. "Flexibility, Manageability, and Performance in a Grid Storage Appliance", by J. Bent et al., proceedings of HPDC XI
18. Database monitoring: http://dbsmon.fnal.gov
19. RunJob project: http://www-clued0.fnal.gov/mc_runjob/mainframe.html