The NCRC Grid Scheduling Environment

Frank Indiviglio, *Geophysical Fluid Dynamics Laboratory and* **Don Maxwell**, *Oak Ridge National Laboratory*

ABSTRACT: *In support of the NCRC, a joint computing center between NOAA and ORNL, a grid-based scheduling infrastructure was designed to allow geographically separate computing resources to be used as production resources in climate and weather research workflows. These workflows require job coordination between the two centers in order to provide a complete workflow of data staging, computation, post-analysis and archival. This paper details the design, implementation and initial production phase of the infrastructure and lessons learned from the process.*

KEYWORDS: XT, Moab, TORQUE

1. Introduction

The National Climate-Computing Research Center(NCRC) is located at Oak Ridge National Laboratory and is a joint computing center between NOAA and ORNL. The NCRC supports NOAA's Research and Development production modelling in climate and weather. The NCRC supports the NOAA's R&D community across the country and represents a significant paradigm shift in NOAA's computing strategy. The NCRC is the first remote production computing facility in NOAA's history and presented significant challenges in the preserving users' scientific workflows. With the NCRC, users' production model runs would run on Gaea, the NCRC's main compute resource, and supporting jobs such as post processing, archival, and automated analysis would have to run at the users' home centers. This workflow model required the design and implementation of a grid based scheduling environment that would allow for automated and seamless integration of the multiple sites and scientific workflows. In response to this, engineers from NOAA and ORNL designed and implemented a multi-site scheduling architecture to allow for the implementation of these workflows across the grid.

2. Design

The core design of the scheduling environment for NCRC needed to support three base requirements of the NOAA scientific community:

- The entire workflow would need to be automated. This would include archival and post-processing jobs that would run at the users local sites, separate from production runs on Gaea.
- The entire workflow should be able to be tied together using a scheduling platform. This would allow other sites or centers to be added into the environment with the same platform and little change to the underlying workflow.
- Provide enough abstraction to workflow developers to allow workflow components to be generic for all centers and sites.

These requirements formed the basis of the design for the scheduling environment that would tie the all of the user centers and sites into a common framework. These three requirements also allow current frameworks to be adapted to this new multi-site workflow structure in a concise and easy manner, by providing similar workflow architecture at each site independent of underlying hardware or software.

To better demonstrate how this scheduling architecture was to support this multi-site workflow, we will take a closer look at the hardware located at ORNL and at GFDL. Oak Ridge National Laboratory selected Cray to provide Gaea, a 30,912 core XT6 during the first phase of production. Along with the core XT6 infrastructure, Gaea also includes 4 schedulable eslogin nodes, 16 schedulable local data transfer nodes, and 8 schedulable remote data transfer nodes. This hardware forms the base of the GFDL's production compute as it moved from local HPC resources to Gaea. In addition to Gaea, GFDL has a post-processing and analysis cluster consisting of 60 post-processing nodes and 6 analysis nodes. Also located at GFDL is a 25PB DMF archive that supports and stores the entirety of GFDL experimental data. This archive serves as the data repository for input conditions at ORNL, and for all post-processing and analysis work locally run at GFDL.

Additionally GFDL's Modelling Services group provides the Flexible Modelling System (FMS). This software infrastructure is meant to abstract the computational hardware and intricacies of each platform a user may run on, allowing the user to concentrate on scientific experimentation and not the details of the system. The FMS infrastructure provides a runtime environment, which generates scripts for climate models, and an application infrastructure so that models that use FMS have common application characteristics.

In addition to supporting FMS, the scheduling environment had to be flexible enough to also allow support for other NOAA developed workflow managers. To allow for this, the scheduler environment would have to provide both basic and advanced features to support multiple workflow managers to allow for job control and workflow management.

In addition to these requirements and infrastructure available the scheduling environment had to provide a platform for centralized reporting, for accounting / allocation management. These features are essential in the shared environment, as the users from several centers must share these computing resources. This would be essential to allow allocation committees to make decisions based on monthly usage patterns of the aggregated user base.

To address these requirements, a grid based scheduling and allocation platform was conceived to allow for

workflow management, reporting and allocation across all of the potential user centers and computational sites. This platform would allow a centralized view of the HPC enterprise and provide both users and management with the features that would allow easy management of experiments and of the HPC enterprise. This design would consist of a centralized grid node that would provide users with a centralized submission and management point for all of their experiments.

In addition to grid job scheduling, grid authentication and data transfer facilities would need to be integrated into the workflow. This would facilitate the data migration to post-processing, and pre-staging the data from the computation site to the user's center. In previous workflows data movement depended highly on local tools and no authentication was required as the entire experiment happened inside a single HPC realm. In the new environment, jobs and user data cross in and out of different organizational security domains, so a common and reliable method for authentication was also needed to facilitate a truly capable multi-site workflow.

For this environment, MyProxy was selected as an authentication platform. This provided x.509 based security to the grid and provided a manageable way to trust user data flow in and out of centers and sites. This integration of a security method allows for the centralized management of user keys for data transfer and user authentication across the grid.

Key management is controlled via a MyProxy backend providing centralized authentication across user centers and computational sites. This MyProxy database is made highly available via MyProxy's replication services. These replication services allow for failover throughout the grid and multiple stores across the grid. Key control and assignment is currently controlled via the bastion host to Gaea, or other clusters. Users setup up a new master certificate once a year, and have their proxy certificate, which allows for remote authentication of data transfers, automatically renewed in the login process. The proxy certificate lasts 30 days, so a user may not login for this duration and his or her jobs will continue to run and automated data transfers will continue until the proxy certificate reaches its expiration date. The certificate is automatically propagated into the user's environment via provided scripts that are custom tailored for each center.

The estimated peak data transfer rates for the grid are about 85TB per day. This is 5TB of input transfer into Gaea, and 80TB of output generated per day. To allow this volume of data, NOAA in partnership with ORNL, Internet2, Indiana University, and others built the NWAVE network that provides direct, high-speed connections between NOAA user centers and computational sites.

To enable accounting and allocation across the grid, a tightly coupled distributed allocation management system was designed using Gold from Adaptive Computing. The allocation will allow centralized grid reporting from the grid-head or from the Users centers. This is achieved by deploying a local Gold instance to all resources, and at a scheduled interval synchronizing the central database at the grid-head. This synchronization can happen at an hourly or daily intervals as needed by the users or management. Initially the design called for a centralized allocation system with a highly available backup system, however it was later determined the distributed synchronization model would provide a better model for fault tolerance and general scheduling performance. Currently charges are incurred at job exit and pre-withdrawing time at execution start is not enabled. This was done to support the FMS model of requesting the max wall clock limit to allow for the most model interactions per run script.

Given these requirements and design decisions, it was realized early on in the implementation that a phased approach was needed to fully implement the design over time and feature maturity. The initial phase of production uses GFDL as a focal point of user interaction with Gaea and the grid scheduler.

3. Current Production

3.1 Workflow

The workflow needed to support NOAA users depends on tightly coupled events from initial submission to analysis. The current workflow, while allowing for submission at either site, begins on Gaea. During this initial phase of production users' workflows and the workflow itself is designed to begin on the login nodes on Gaea. This workflow itself is heavily automated by the Flexible Modelling System, which is developed and maintained at GFDL. The modelling system provides users with a common platform to run their experimentation over the local and remote sites.

An experiment can be treated as eight steps:
- *Experiment Creation/Submission*
- *Data Transfer*
- *Pre-process*
- *Model segment run*

• *Combination/Packaging*
• *Data transfer*
• *Archive*
• *Post-processing*

Each step is critical in the job run as it forms the basis for the next step to complete successfully. In order to better define the process, we will need to define each step and its effects on the following steps.

1. Experiment Creation/Submission-
The creation and submission of the experiment and provides the general structure of the job including: experiment duration and computational sites where the job will run, run time, data treatment, etc. The submission step provides structure for the rest of the job.

2. Data Transfer-
This data transfer step will move data from the local center to the remote site. The initial transfer of data is about 1/16th of the returning data stream but can vary based on experiment and model type to up to 1/5th of the returning data.

3. Pre-preprocess –
The preprocessing step prepares to transfer recently created input data for the next segment run. For some experiments this step may also require file checksumming of data that was recently moved, underwent format changes, data resizing, or other data transformations to prep for the model run.

4. Model segment run –
The experiment segment runs in this job stage. During the segment execution two types of data may be produced: history/forecast data and restart data. The history data will be further processed in the job chain, the restart data can be re-linked for use in the next segment run as input conditions. This re-linking would happen in the pre-process stage. In the event of a failure on the executable, the job segment should resubmit itself for re-run. Administration staff should be notified of the failure. A second failure should push the job stream into the hold queue for analysis.

5. Combination/Packaging –
This stage can perform two actions. The first action, if needed, is the combination of history data into a more concise dataset for post- processing. This step may not be needed for all job streams. The second function, needed for all job streams, is to copy the data created from the preceding run to the Long Term Scratch file system and to put the dataset into an archive format. Currently tar and cpio are the most widely used archive formats, the decision on file format is based on user and framework preference.

6. Data Transfer-
This version of the data transfer is the outbound data transfer to the local centers from the remote computing center. This type of transfer will account for a vast majority of data transfers. In addition to copying data from computational site to the users' center this job stage will also provide data verification services in the form of checksumming data at each end of the transfer. In the event that the generated checksums do not match the job will notify the administrators and attempt the transfer a second time. In the event of a second failure, the job should be moved into a hold queue for failure analysis.

7. Archive –
This step commits the newly arrived data from an intermediary storage medium (if needed), to the local center's archive. Currently each Center has its own archive and archive manager. In addition to transferring to the archive a secondary transfer into a secondary filesystem, which will be used as a starting point for post-processing or analysis. In GFDL's case this filesystem will most likely be named ptmp; the naming of this filesystem will vary by Center.

8.Post-Processing-The final step for each segment is having its data post-processed for analysis. This process retrieves data from the archive or a ptmp like filesystem and performs data operations on the segment, which may cause the data to grow by a factor of 4. The end result is then placed into the archive for use in analysis jobs.

Currently the workflow as implemented is more of a peer-scheduling environment. This allowed us to initially support two post-processing environments. This was needed to as the initial environment needed to utilize the existing legacy post-processing equipment at GFDL, prior to the installation of the new post-processing hardware in December of 2010. The peer-to-peer schema was selected to allow the production workflow to exist on the legacy equipment and development on both gaea and the new post-processing equipment.

3.2 Batch Implementation

The batch vendor selected to provide the primary workflow implementation was Adaptive Computing™ using their Moab Workload Manager®, Moab Grid Suite®, TORQUE Resource Manager and Gold Allocation Manager products. The current batch implementation has certainly evolved as different phases

have been added to the workflow and as scale has dictated. Many lessons were learned during the evolution that will be covered in a later section of the paper.

As described in the workflow discussion above, jobs must flow among several different compute resources even at times in different administrative domains for the entire process to complete. This complexity required the introduction of the Moab Grid Suite® to complement the Moab Workload Manager®. The grid software directs the job to the appropriate Moab instance responsible for scheduling the requested resources. Once there, the workload manager software provides the engine needed to schedule jobs on each compute resource. The scheduler will not select a job for execution unless an allocation can be obtained from the Gold Allocation Manager. Finally, the batch cycle completes with the TORQUE Resource Manager that actually launches jobs that are selected for execution. The process is illustrated in Figure 1.

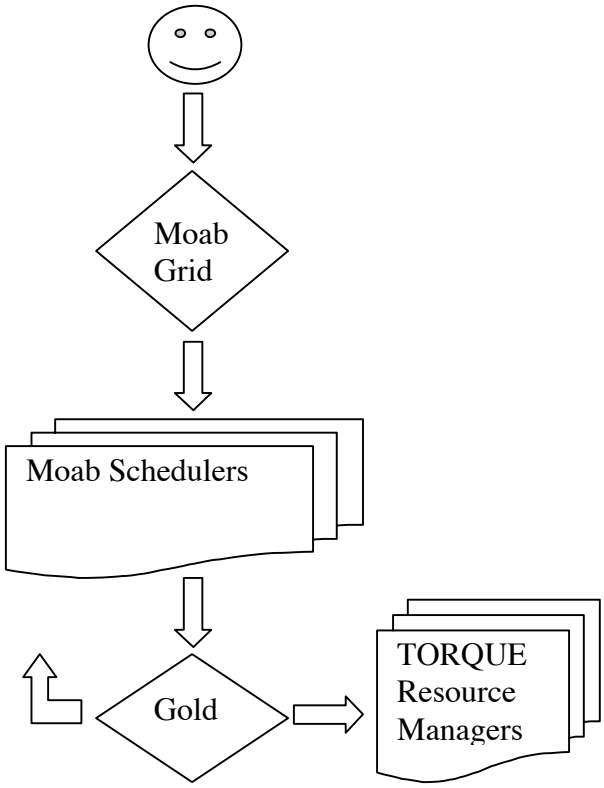

Figure 1. Moab Software Job Flow

As illustrated, the grid software provides the user with the flexibility to launch jobs from a single point to many compute resources that could exist in geographically dispersed locations. This was indeed a requirement for the GFDL workflow with compute resources located at ORNL and post-processing and data archival resources located at GFDL. With that in mind, a design was developed to not only meet that requirement but also to meet GFDL job scheduling priority requirements.

To briefly recap, the resources currently involved in scheduling are listed below in Table 1.

Compute Resource	Purpose
C1	Cray XT6 Compute Resource
T1	Cray XT6 Test Resource
esLogin	Login Nodes
LDTN	Local Data Transfer Nodes
RDTN	Remote Data Transfer Nodes
GFDL	Post Processing and Archival

Table 1.

All of these resources are located at ORNL except the resource labelled GFDL. Several issues led to the need to maintain multiple Moab servers to schedule all of these resources. First, GFDL being remotely located and in a different administrative domain dictated the need for a separate Moab instance to reside at that site. This provides the ability to more easily separate scheduling policies, user bases, administrative responsibilities and privileges, etc. So, at least two Moab instances were needed from the outset to provide grid communication between ORNL and GFDL. Next, due to the current requirement that a Moab instance reside inside a Cray X* system for scheduling that particular system, other Moab instances had to be established for each Cray resource. That brings the total Moab instances in the current grid configuration to four. Early on, there was a requirement that every Moab instance see every job from every other Moab instance. This turned out to not actually be a requirement and caused some issues that will be discussed later, but keeping that in mind, Figure 2 attempts to illustrate the connections that exist between the various Moab instances.

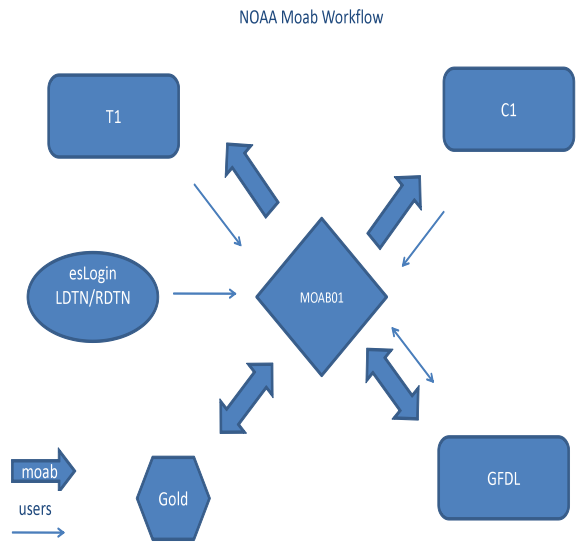

Figure 2.

The arrows in Figure 2 are used to indicate the communication patterns used by Moab and the users. For instance, users submit jobs from all of the compute resources to the Moab grid head (moab01) and cannot submit directly back to the compute resource on which the current job is running. The exception to this is GFDL where there is a double arrow for both users and Moab. All jobs can be seen on both moab01 and the GFDL Moab instance due to this communication and users can submit jobs in both directions. However, looking at the single arrows for C1 and T1, while moab01 migrates jobs destined for those resources to the appropriate Moab, C1 and T1 do not see any other jobs except those that are running in the respective Moab instance. Again, this became an important point that will be discussed later. As Cray X* systems require a Moab instance locally for scheduling, they also each have a TORQUE instance running locally. While this configuration is probably not entirely necessary given the architecture of TORQUE, it seems appropriate to map to each Moab instance. This configuration may be modified at a later date to centralize TORQUE services onto the grid head. There is currently a TORQUE server running on the grid head that serves as the resource manager for esLogin, LDTN and RDTN services. The final piece of the figure is the Gold Allocation Manager that resides at GFDL. Each job that is submitted to the grid head attempts to obtain an allocation from the Gold server. If an allocation is overdrawn for the account requested, the job is placed into an account with a very low priority allowing the job to run only when others that still have allocation are not running. This serves a couple of needs since it gives users who have not exceeded their allocation priority while also providing good machine utilization when those jobs are not running.

Finally, once a job arrives at the appropriate destination, it must be appropriately prioritized based on other jobs that are currently waiting. The GFDL job scheduling priority requirements were provided and the priority table below is a result of those requirements.

Factor	Unit of Weight	Actual Weight (Minutes)	Value
Class	# of days	1440	Urgent (10) Persistent (5) Debug (2) Batch (1) Windfall (-365)
Fairshare	# of minutes	1	(<>)5% user (+/-) 30 minutes (<>)5% class (+/-) 60 minutes
Queue Time	1 minute	1	Provided by Moab

Table 2.

While Table 2 does not cover the scope of every requirement, it is a good summary of most of the requirements. Generally, there are five queues or classes that have a given priority with an accompanying project allocation in Gold. Fairshare targets are used to attempt to provide approximately 50% of the compute resources to the urgent and persistent classes. A fairshare target in Moab provides the capability of giving objects such as users, groups, accounts, classes, etc. a targeted percentage of the system, and this target is adjusted based on parameters that are configurable. In this case, any time a class configured with a fairshare target is above or below five percent of its target it is adjusted in the appropriate direction by 60 minutes. It should be noted that all priorities have been normalized to minutes since queue time is provided by Moab in minutes. Having a single unit of priority tends to make priority discussions less complicated.

A couple of other noteworthy requirements not covered above that are provided by Moab are a standing debug reservation of ten percent of the machine which also services interactive jobs, and a novel queue which gathers jobs greater than twenty-five percent of the machine for later execution. The novel queue generally

remains on hold until after a maintenance period. GFDL does not typically run capability jobs in their standard workflow and chose not to generally allow the machine to drain in order to run a large job outside of the workflow.

4. Future

4.1 Workflow

As the workflow becomes more mature, improvements in job management and overall grid management will certainly be needed as computational resources become larger and the workflow as a whole becomes more complex. Improvements in job tracking and job "trees" are desired to allow users to trace job lineage and hierarchy. This will also allow users to better track job and experiment flow through the grid. Initially the idea of a control job was developed for management of an entire experiment. This job would be created during phase one of the workflow and serve as an information store on the grid for the experiment over its segment runs. These changes and are planned for the workflow from the scheduler perspective to provide better information to the users and workflow developers.

This type of enhancement will need to include advanced features of the scheduler such as triggers and job templating. This work is being planned in the next phase of development to provide a richer feature set to the user. Further work in automated job stream checking, job order checking and the ability to show better statistics and job relationships across the grid is also underway to provide the user base and workflow developers a base to enhance the scientific platform across the R&D enterprise. Feature enhancements are already being worked on to provide job dependencies on job names, the ability to group jobs onto similar nodes to facilitate data sharing between runs, and the implementation of the grid-head which will provide a centralize view to the workflow instead of across peers which may limit job tracking and relationship.

4.2 Batch Implementation

While scheduling has reached somewhat of a steady state, there is certainly room for improvement with the current system. A few of the deficiencies are multiple job numbers for the same job when using a grid configuration, potential job starvation at the grid head when a migration policy of JUSTINTIME is enabled, and the inability to easily define and maintain policies and priorities for multiple Moab instances. A few steps are being taken to address these issues as outlined below.

A common complaint from users in the Moab grid environment is the fact that multiple job numbers must be tracked for the same job. When a job is submitted to the grid, a job number is generated that reflects the Moab instance the client is communicating with along with the next numeric job number (i.e., gaea.774862). Once that job is migrated to the destination Moab instance and is visible to the TORQUE Resource Manager serving those particular compute resources, it gets a TORQUE job number (i.e., 426102). The TORQUE job number is what is commonly used to identify output and error files, but the user generally only sees the initial grid job number upon submission, a situation that leads to confusion. While there are tools to correlate these job numbers (showq –v), it can be a source of confusion for both the user and administrators attempting to troubleshoot an issue. This problem has been resolved in new versions of both Moab and TORQUE whereby Moab will provide the job number to TORQUE, and it will be accepted by TORQUE.

Job dependencies necessary for chaining together the workflow also proved to be somewhat of a challenge in the grid environment. In order to accommodate these dependencies, a job migration policy of JUSTINTIME had to be configured on the grid head. This policy will only migrate the job to the final Moab instance once it is ready to run. In other words, all job dependencies must be met and there must be available compute resources to run the job. This latter attribute of JUSTINTIME means that larger jobs could potentially be starved if smaller jobs continue to be migrated as slots become available. If a small job completes making room for another small job, a new small job will be migrated creating a cycle that could starve a larger job.

Multiple Moab instances generally require maintaining policies and priorities on each instance that can prove to be challenging. In a grid environment, centralizing these policies and priorities on the grid head at a single control point should solve the problem. However, certain parameters are not yet instance or RM aware making it more difficult to provide different policies for each instance from the grid head.

These latter two issues and others can be addressed by first moving to an existing grid configuration of master/slave and eventually to a new development effort underway to remove the Moab instance from the Cray X* systems completely. Some development effort to make more parameters instance or RM aware will be required for each scenario. The requirement for a Moab instance to run locally on each Cray X* is based purely on the inability of the Cray job launch software dubbed ALPS (Application Level Placement Scheduler) to communicate outside of the Cray. In a master/slave configuration, all scheduling decisions are moved to the grid head

simplifying the need to maintain policies and priorities in multiple places and removing the need for JUSTINTIME and the problems associated with it. However, a Moab instance must still run locally on the Cray due to the ALPS constraint.

Ultimately, removing Moab from running locally on the Cray solves several problems. Centralization is a key reason as already mentioned, but it also provides the flexibility of provisioning resources as needed to meet the needs of running a complex Moab configuration. The Cray service nodes have traditionally lagged in technology refreshes and are often lacking in the resources needed to efficiently run Moab. With no requirement for Moab to run internally, customers can acquire the appropriate resources to run Moab more efficiently. The current development effort underway to accomplish this task involves the use of a new Adaptive Computing product called Moab Service Manager (MSM). This product will run locally on the Cray and basically provide a socket connection to the Moab engine in order to provide the required connection to ALPS. This should provide a much more efficient grid configuration by eliminating the need to run multiple Moab instances at a given site.

5. Lessons Learned

5.1 Workflow

Early on the workflow development team planned to use a large number of specific scheduler options such as triggers and advanced API features. In the end they switched to a more basic approach in an effort to deliver a workflow that would be flexible and general enough for several platforms. These decisions allowed an accelerated on boarding of the enhanced workflow. In addition it let the development and system team to debug grid-issues and more rapidly repair shortcomings since the workflow compartmentalizes most of the steps of the workflow into single jobs.

Most of the lessons learned were the based on adaption to the new platform using its expanded feature set. FMS packages run information (such as diagnostic selection) in the run script. These run scripts were found to be larger than the expected limit of moab, they also stress the overall system as the transportation of these scripts becomes taxing as job volume increases. The ability to submit from each scheduler also added complexity as the workflow developers needed to pay attention to the amount of submissions and number of hops from the destination. These issues were able to be handling and address via configuration changes. Other lessons learned with respect to submission were that the developers had to take into account the networking aspects of grid scheduling. Large volumes of simultaneous submittals caused timeouts and delays in interactive response for the workflow and interactive users. These issues are being addressed both on the scheduling side through software enhancement and through the workflow by inserting delays and checking for timeouts at submission time.

5.2 Batch Implementation

Early on, with Moab instances running on the grid head and locally on each Cray X*, it seemed appropriate to just use the standard configuration whereby all service node Moab clients inside a particular Cray pointed to the Moab instance running on that respective Cray. The primary login nodes (esLogins) utilized point to the grid head Moab instance, but not the Cray clients. This configuration meant that each Cray needed the ability to see every other Moab instance in order to provide users with the capability of submitting a job to any Moab instance from another job. This configuration proved problematic with timeouts, hop counts being exceeded, job migration confusion, etc. Finally, after taking a closer look due to all of the issues, it seemed a better outcome could be achieved by pointing all clients at the grid head making it the focal point for all job submissions and removing the spaghetti configurations that pointed all Moab instances to all Moab instances. The grid head could see all instances and with this configuration, there was no loss of functionality. This configuration has proven to be much more stable and has solved the problems mentioned.

Conclusion

The initial implementation of the NCRC grid scheduling environment has evolved into tool that can support the production system spanning NOAA and ORNL. This system provides flexibility by providing key features that allow multiple workflow managers to function across computational centers and user sites. As the commercial and provided software mature they will offer more grid capable features. This maturation will provide users with even more capable tools with which to manage their job flows. The resulting increase of capability will allow for further growth of existing resources and further expansion of the R&D environment at both NOAA and ORNL.

About the Authors

Frank Indiviglio is the Production Lead for the Geophysical Fluid Dynamics Laboratory/NOAA in Princeton, NJ. He can be reached at frank.indiviglio@noaa.gov

Don Maxwell is a Senior System Administrator at Oak Ridge National Laboratory primarily focused on the Cray XT series. He has been a key member of past teams in bringing up new supercomputers for the NCCS. He can be reached at maxwellde@ornl.gov.