Air Quality Forecasting on Campus Grid Environment

Yonghong Yan, Barbara M. Chapman, and Babu Sundaram
{yanyh, chapman, babu}@cs.uh.edu

Department of Computer Science
University of Houston

Abstract

Air Quality Forecasting (AQF) is a new discipline that attempts to reliably predict atmospheric pollution. The application has complex workflow and in order to produce timely and reliable forecast results daily, each execution requires multiple computational and storage resources to be simultaneously and collaboratively available. Deploying AQF on grid is one option to satisfy such needs, but requires related grid middleware to support automated application-specific scheduling and execution on grid resources. This paper presents our initial experience of deploying AQF on a campus grid environment and our current efforts of developing a solution of grid-enabling AQF-like applications in Gracce project. Gracce has the goal to provide domain users a grid platform supporting from the management of an application and its dataset, to the automatic execution and viewing of results. In Gracce, application workflow is described using GAMDL, a powerful data-flow language for domain users in describing application logics. The Gracce metascheduler architecture, which includes a workflow-orchestrated metascheduler, an event-driven workflow engine, and an execution runtime system provides the required functionalities of scheduling application workflow in global level and coordinating workflow executions.

1 Introduction

Air Quality Forecasting (AQF) [22] is a new discipline that attempts to reliably predict atmospheric pollution, especially high levels of ozone. The application incorporates multiple dependent computational modules that make intensive use of numerical tools, requires high compute power for the simulation of meteorological and chemical processes, and entails the transfer, storage and analysis of a huge amount of observational and simulation data [6]. We participate in an effort to build such a service, with the goal of providing timely, reliable forecasts of air quality for the Houston-Galveston region and for several other regions in the South Central USA that have encountered problems with air quality in the recent past [3, 4]. On-going work at the University of Houston (UH) aims to create, test and deploy an AQF application as well as to establish a suitable development and deployment environment.

Grid technology [15], and middlewares to enable the creation of such grids, provide a potential strategy for meeting the computational and storage needs of AQF executions. Users with large-scale problems, such as AQF applications, may exploit multiple distributed high performance computing resources in a grid environment to produce high quality results that cannot be achieved from single-domain resources. As the grid technology becomes mature and standardized, deploying application on grid to efficiently use grid powers is becoming more important than technology and standardization themselves. Additional efforts are required to fill the gaps between grid visions and domain expectations. Yet such efforts are still in the stage of trial and related experiences are very application-specific and technology oriented.

Including ours [2, 3], most of current approaches of grid application deployment starts with the packaging or wrapping of legacy application codes with grid services and utilities of grid remote execution and automatic file transfer, and presents them in a grid portal to domain users. Efforts in supporting the automated application-specific scheduling and execution on grid resources, and thus providing users an end-to-end grid environment (not grid technology) are very few. This paper presents our experience of deploying AQF application on a campus grid environment and our current efforts of developing a solution of grid-enabling AQF-like applications in Gracce project [28]. The initial efforts provided a working, but

not feature-complete solution to support AQF run on the resources across our campus grid. It is the basis for the next stage development of a general-purpose solution in Gracce.

Gracce has the goal to provide domain users an application grid platform supporting from the management of the application and its dataset, to the automatic execution and viewing of results. In Gracce, application coordination and collaboration, a typical example of which is workflow, is described using GAMDL, a powerful data-flow language for domain users in describing application logics. Gracce metascheduler architecture is designed as a software above the available grid infrastructural middlewares to provide functionalities of grid resource allocation, workflow coordination and runtime control. The architecture includes a workflow-orchestrated metascheduler with planning and reservation features, an event-driven workflow engine able to coordinate the scheduling process and job execution, and a runtime system to control workflow executions.

The organization of this paper is as follows. Section 2 introduces AQF application, our initial efforts in deploying AQF on UH Campus grid [2], and the requirements to support automatic AQF run on grid in production quality. At the end of this section, software and projects related to these requirements are surveyed. Section 3 presents the current two major efforts in Gracce project, GAMDL and Gracce metascheduling architecture. Section 4 summarizes our work and its strengths.

2 Experience of AQF on Campus Grid and New Requirements

Our initial efforts in deploying AQF on grid utilized the basic functionalities provided by Globus toolkits 2.x [12] and provided a working solution to support AQF run on the resources across our campus grid [3]. But it is not feature complete to build an application grid environment, which is our ultimate goal of application deployment on grid. In this section, we introduce AQF application and our current deploying approach, and analyze issues in the approach. Also based on our current experience, three additional features that are required for grid middlewares to fulfill our goal are identified in current stage of the project. Middlewares and efforts related to these features are studied at the end of this section.

2.1 AQF Introduction

AQF is an integrated computational model that is composed of three subsystems: the PSU/NCAR MM5 mesoscale weather forecast model [9], the Sparse Matrix Operator Kernel Emission System code (SMOKE) [24], and EPA's CMAQ chemical transport model [7]. AQF execution is a computational sequence of the three subsystems on heterogeneous resources with increasing resolution and decreasing geographical boundaries. Figure 1 illustrates the workflow of a nested 2-day forecasting operation over a single region of interest by a three-domain computation. The 36km domain computation provides coarse forecast data over continental USA, the 12km provides data across the south central USA, and the 4km forecasts air quality across a smaller geographic region. Each rectangle represents a computational module and each arrow indicates the flow of data between modules. An AQF daily run starts with the download of the data of ETA weather forecast analysis on 15:30PM, and should produce results before 6:00AM the next day to researches and state and local officials [22, 34]. In our experience, a sequential run on a 256-CPU Linux cluster can only finish AQF forecast timely for 12km domain, with about 30G data generated daily. Substantial computational and storage resources are required in order to provide high-quality forecasting in an urban area based upon 4km and 1km domains. Enabling AQF on UH campus grid and utilizing the parallelism of module executions in AQF workflow are two approaches we explored for the timely and accurate forecasting in finer domain regions [2].

2.2 Initial Experience of AQF Deployment on Campus Grid

The UH campus grid currently consists of a heterogeneous cluster of Sun SMPs, a Beowulf cluster and an SGI visualization system, with 9 TB storage, at UH High Performance Computing Center (HPCC) [31], and clusters of Sun SMPs and Beowulf and several Sun workstations in different departments. AQF modules are installed and configured in these resources, and disk and tape spaces are allocated for its daily execution. Sun Grid Engine (SGE) [35] and Platform LSF [32] have been installed to manage resources

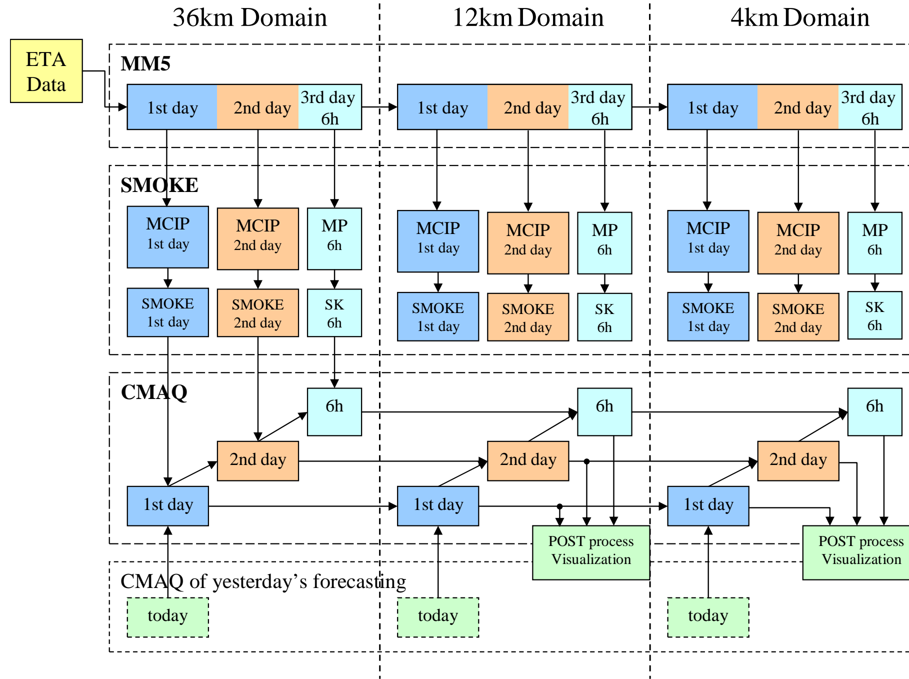

Figure 1: AQF Application Workflow

within the individual administrative domains. Globus toolkits [12] are installed on these resources for grid job execution and file transfer. UH HPCC serves as the CA [14] in our campus grid and is responsible for granting grid accounts. Individual departmental resources are configured to accept only the certificates from this CA. To make it as easy as possible for users to interact with the services provided through the campus grid, EZ-Grid [3], a light-weight web-based portal, have been developed. It uses the Java CoG Kit [11] to provide a convenient interface to all Globus functions, including grid authentication using X.509 certificates and management of GSI proxies [14], job specification, submission and management, file transfers, and grid resource information and load status.

In current campus grid setup, AQF workflow structure is described using an XML file, and a Perl script controls AQF workflow execution and interacts with Globus in EZ-Grid portal. A module in the workflow is described as a task in the XML file that will become a grid computational job (Module, task and job refer to the same entity in different context, the term "module" is from application people, task is a workflow concept and job is often used in grid context). Dependencies between modules are specified as the parent-child relationships of tasks in which parent tasks produce the data that are consumed by child tasks. For each task, details about the executable and resources where it is going to be launched are hard-coded in the task RSL file [36]. The Perl script reads the XML file and controls the overall execution of AQF tasks, including submitting jobs to grid resources, initiating file transfer when the data are available, and resolving task dependencies. So basically, the AQF execution scenario, which is about where, when and how each workflow task is going to be launched and a dependency is handled, are predefined in the description XML file and the control Perl script.

There are several issues in our current solution. Firstly, computational resources are pre-allocated for AQF tasks and are assumed to be available during task execution periods. The allocated resources specified in task RSL file are defined by system administrators, who also reserves the resources in the local scheduler to ensure their availabilities. Obviously, this type of human-scheduling policy is not suitable for the changing grid environments and resource allocation should be automated to provide best decisions according to the resource load status. Secondly, failures in a grid resource will cause the failure of the whole AQF run if without users' intervention. There is no scheduler to allocated resources for a task whose dedicated resource fails. Specifying a secondary resource in the RSL is one solution, yet normally the secondary resource is rather busy such that the task would have to wait for long period in the local queue. Thirdly, the non-standard XML and script approach for workflow description and execution control is error-prone and introduces lots of burden for domain users and system administrators. Domain

users are required to be a programmer for XML, Perl and RSL for their applications, which is a daunting task in the process of AQF deployment. Instead of digging down all details of grid setup and resource scheduling in AQF run, users expect a complete application execution environment, from graphical user interface to final view of execution results.

2.3 New Requirements and Studies of Related Grid Middlewares

Understanding the above issues and users' expectations, and also based on our experience in campus grid setup and AQF deployment, we have identified three features that are required from grid middleware to support the automatic AQF execution on grid:

- **Grid Application Modeling and Description** This is to provide domain users a modeling language to describe an application so that users are relieved from the tedious details of workflow, execution control and grid activities. Such language should target for application people, be powerful but easy to use and require only introductory or even no knowledge in grid technology when describing a complex application structure, and should be easily integrated with other grid middlewares, such as workflow and scheduling systems.
- **Grid Metascheduling** AQF daily run requires grid middleware to provide functionalities of automatic resource allocation for AQF workflow tasks across grid. Grid metascheduler operating on the global level is the middleware that may possibly satisfy such needs. But an AQF job typically consists of several dependent tasks and placing these tasks on the appropriate resources across a grid for efficient execution is a much more complex problem than scheduling a single-executable job. Moreover, the scheduling decision must ensure application Quality-of-Service, in our case, which means forecasting results must be generated timely.
- **Workflow Orchestration in Resource Allocations** In workflow execution, the approach of scheduling tasks on resources right after the task dependencies are resolved can not guarantee resources can be discovered and allocated. Resource co-allocation normally requires planning of workflow execution and advanced reservation of resources. In making these decisions, metascheduler should consider the task execution scenario based on AQF workflow to make sure the resource co-allocation are properly coordinated with the application workflow.

There have been lots of efforts addressing issues of scheduling in grid computing area. Globus GRAM [19] and RSL [36] are the early, de-facto standard in providing solutions for secure job execution in metacomputing environments. However, GRAM and Globus itself do not have grid scheduling and brokering functionalities. DUROC [17] is an early effort in Globus 2.x to address the issues of resource co-allocations in RSL-specified multi-request for resources. Globus GARA [13], Maui Silver [33] and architecture defined in [10] introduced advanced reservation [23] into GRAM co-allocations architecture. SNAP [18], which extends Globus GRAM and GARA, proposes a service negotiation protocol into grid scheduling process. Pegasus [8] addresses workflow job scheduling issues as AI planning in constructing and execution workflow from application logic workflow.

Although issues related to grid scheduling have been researched in different projects, efforts to develop a fully functional grid metascheduler are very few, and to our best knowledge, none of them addresses workflow orchestration issue in resource allocations. Community Scheduler Framework (CSF) [26] implements a number of low-level services as a development basis for implementing a fully functional metascheduler. Maui silver [33] scheduler jobs across Maui-managed resources and is not standard based. GRASP [29] aims to provide an OGSI-compliant resource allocation and reservation services following the requirement of Grid Scheduling Architecture proposed by Grid Scheduling Architecture Research Group of GGF [30]. Nimrod/G [20] is an resource management and scheduling system with focus on computational economy in scheduling tasks based on their deadlines and budgets. MARS Metascheduler [1] is an on-demand scheduler which discovers and schedules the required resources for a critical-priority task to start immediately. We also studied efforts in addressing various issues in grid scheduling.

Efforts that address scheduling in workflow community are also very limited. Triana [5] workflow engine schedule tasks across multiple resources either in parallel or in a pipeline. GridFlow [16] executes a

grid workflow according to a simulated schedule. Pegasus [8] separates abstract workflow with the concrete workflow and relies on Condor DAGMan [27] to schedule the workflow jobs. For workflow descriptions that directly interest us, several languages are studied based on our AQF needs. Business Process Execution Language (BPEL) [25] is an XML-based workflow definition language that allows businesses to describe enterprise business processes. BPEL is in low-level web service level, which additional extension and wrapping development needed to make it easy of use by grid application owners. XScufl [39] is a specific workflow definition language for Taverna project, but XScufl is too fine grained for describing scientific applications. Abstract Grid Workflow Language (AGWL) [21] "programs" application control flow using constructs of imperative programming style. For data-flow applications, users have to translate the dataflow into control-flow to use AGWL.

3 Gracce: Building An Application Grid Environment

Driven by AQF application, Gracce (Grid Application Coordination, Collaboration and Execution) project [28] was proposed to develop a set of grid middlewares for grid application deployment. The vision of Gracce is to provide domain scientists an application-specific grid environment, supporting from the management of an application and its dataset, to the automatic execution and viewing of results. In Gracce solutions, domain users are only required to provide application descriptions and major resource requirements, and Gracce is responsible for allocating grid resources for tasks, placing tasks on resources for execution and monitoring them, and returning the results back to users. The three required features for automatic AQF execution on grid are addressed by two efforts in Gracce: the Gracce Application Modeling and Description Language, and Gracce metascheduler architecture.

3.1 Gracce Application Modeling and Description Language (GAMDL)

GAMDL is a high level abstraction language for domain scientists to describe their applications in grid environment. GAMDL is a data-flow modeling language, compared with other solutions that describe application control-flow structures. GAMDL models an application in its domain logics and users do not need to extract application control structures to construct a workflow. By using conditioned properties and conditioned pipes, GAMDL allows control-flow to be defined within dataflow. GAMDL also introduces the concept of multiple-value property (mvproperty) for easy description of similar application entities, such as files, modules, and executables. In a GAMDL document, a universal ID (uid) is used to identify and reference an application entity, which may not be defined in the same document. This is very helpful in programming and mapping application entities with persistence services, such as RDBMS, XML and Java Object.

GAMDL is specified using XML-Schema and a grid application is represented as a `gridApp` XML document with four major child elements: `appExecutables, appDataFiles, appModules and appMdDeps`, which specify the required executables, files, modules, and module dependencies respectively in an application. A task in application workflow is modeled as a "module", a term domain users are familiar with. A module, which consists of executables, input/output file set, and its grid job specification, is normally a computation unit that will become a single-executable grid job. Dependency relationships between modules can be specified in either parent-children pattern indexed by parent tasks or child-parents pattern indexed by child tasks. In each relationship, dependencies are specified by pipes, whose pipeIn specifies the piped output of parent tasks, and pipeOut specifies the piped input of child tasks. Each pipe is conditioned by a boolean string that will be evaluated runtimely to decide whether the piped dependency should be handled or not.

The GAMDL description for AQF is attached in appendix. In AQF gridApp XML document, mvproperties are defined by either including from files (`uhaqf.mvproperties` in this example) or defining them directly(`mdName`). The file `uhaqf.mvproperties` defines three mvproperties: `md={mm5,smoke,cmaq}`, `dmsz={36K,12K,4K}`, and `day={d1,d2}`. The `mdName($md,$dmsz,$day)`, defined as `uhaqf-$md-$dmsz-$day` ($ operator on a mvproperty refers to its values), is extended into 18 (which is #`md`*#`dmsz`*#`day`, # operator on a mvproperty returns the number of its values) values. So this one sentence is enough to reference all the AQF computational modules. AppExecutables, appDataFiles and appModules are all

defined here as uid reference and they should be specified in other documents. AppMdDeps are specified using child-parent relationships, which is easily understood from the GAMDL itself. For detail about GAMDL, we refer readers to [28].

3.2 Gracce Metascheduling Architecture

We define metascheduler as "a grid middleware that discovers, evaluates and co-allocates resources for grid jobs, and coordinates activities between multiple heterogeneous schedulers that operate at the local or cluster level". There are two aspects covered in this definition, the "scheduling" aspect which addresses resource co-allocation issues for applications requiring resources simultaneously at multiple sites, and the "meta" aspect that concerns about the brokering from global grid requests onto resource local schedulers. To address these two aspects, our metascheduler design separates job execution from the metascheduler, making scheduling process independent from underlying grid middleware for job execution. This allows metascheduler to work with various remote execution utilities. The separation is achieved by the concept of Execution Plan (EP) for a workflow job. The job EP contains scheduling decisions for each task and mechanisms for dependency handling, and a separate runtime system translates the EP into execution-specific scripts for job execution and controls. The defined architecture has three components, Metascheduler, EPExec runtime system, and GridDAG workflow engine. To deploy this architecture in a grid environments, we assumes the installation of Grid Information Services. The complete setup is shown in Figure 2.

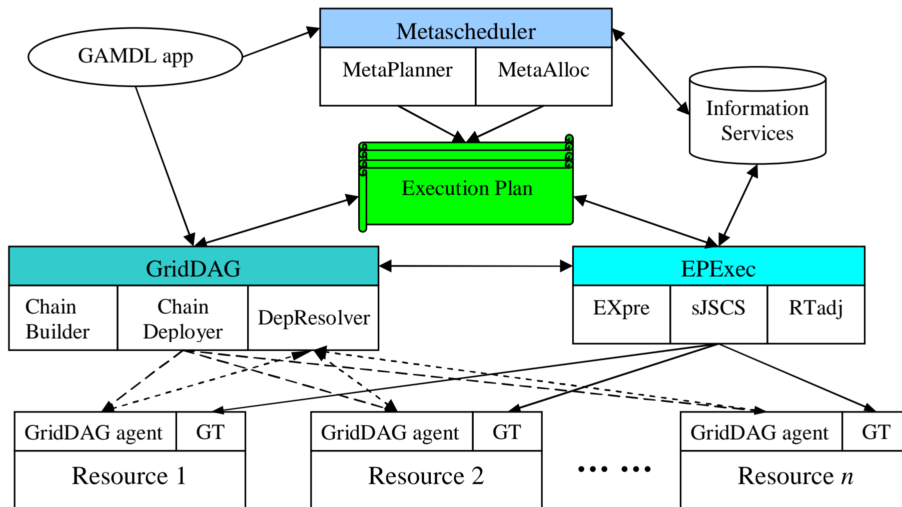

Figure 2: Gracce Metascheduling Architecture

GridDAG is an event-driven workflow system to coordinate the execution of tasks with dependencies. GridDAG events, such as completion of tasks or dependent file availabilities are WS-Notification NotificationMessages produced and consumed by corresponding entities in the architecture. The sequence of these events are referred as event chains and two GridDAG modules, chain builder and deployer setup and install these chains. GridDAG DepResolver keeps track of the events along job execution and invokes certain handlers upon receiving events. GridDAG agents, installed optionally on grid resources generate the outgoing events and consume incoming events locally.

Metascheduler plans job execution and co-allocates resources for workflow tasks. Two modules are designed for these two functionalities, MetaPlanner and MetaAlloc. MetaPlanner predicts the execution scenario for each task; and MetaAlloc searches for suitable resources, negotiates the resource provision and makes reservation with resource providers. The whole metascheduling process is orchestrated by job workflow and the output of this process is a job EP which includes resource allocation decisions and mechanisms for task dependency handling.

EPExec submits task jobs following the EPs, and monitors and manages the execution of these tasks. EPExec also works with GridDAG for handling task dependencies. During execution, EPExec may adjust EP according to the real execution scenario. Being independent from metascheduler, EPExec

can be developed to support different methods of job submission and remote execution utilities. EPExec has three components, EXEpre (Execution Preparation), sJSCS (simple Job Submission and Control Service) and RTadj (RunTime Adjuster).

The life-cycle of a grid workflow job in this architecture is described briefly below:

1. Grid users submit to the Metascheduler a workflow job (possibly with preferred deadline) specified using the application GAMDL.

2. Metascheduler plans the execution of the tasks and the dependency handling mechanisms, and allocates resources for tasks executions. As a result, the job's EP that includes the decision details is outputted.

3. The job's EP is forwarded to EPExec for job execution, which submits tasks to their allocated resources and monitors their executions.

4. During execution, GridDAG agents and DepResolver handle task dependencies and decide whether task dependencies are resolved. EPExec also handles failures and makes required adjustments when the executions are not following the plan.

3.2.1 Gracce metascheduler

Gracce metascheduler plans job execution and co-allocates resources for workflow tasks by the two modules, MetaPlanner and MetaAlloc. MetaPlanner predicts and identifies the execution windows for each task and MetaAlloc searches a list of candidate resources, negotiates and makes the agreement with resource owners of resource provision. A task's execution window (EW), represented by a <EWstarttime, EWlength> pair, is a time period in which task execution shall be. EWstarttime is the window start-time and EWlength denotes the length of this window. The metascheduling process is orchestrated by job workflow so that execution orders of dependent tasks are kept and parallelism of execution of tasks without dependencies are employed.

Given a workflow job, the scheduling process starts with the allocation of resources for the first task in the workflow by the MetaAlloc. When resources are allocated, MetaAlloc also identifies the task's EW. Then, metascheduler processes the child tasks of the first task. First, MetaAlloc discovers a list of candidate resources for each child task and calculate the costs of file transfer between the resource for first task and the candidate resources for child tasks. Secondly, MetaPlanner predicts the task EWs on each candidate resource. EWstarttime is calculated by adding three time value together, the EWstarttime and the EWlength of the first task, and the cost for dependency handling; and EWlength is equal to task's wall-clock execution time. Thirdly, the predicted task EWs associated with each candidate resources are processed by MetaAlloc again, which will choose the best resource and finally reserve the resource for each task. Metascheduler then moves on to process other tasks until the last one. For tasks with more than one parent tasks, MetaPlanner considers the one with latest EW in prediction. Brother tasks will normally have overlapped EW, which means that they may be in execution at the same time. In calculating any time value and task EW, certain grace periods or buffer time are applied.

MetaAlloc allocates grid resources for jobs, mainly computational resources for tasks in a sequence of resource discovery, negotiation, and reservation. In resource discovery, MetaAlloc looks up in Grid Information Services the resources that satisfy task resource requirements and are also available during its EW. The process of filtering resource is split into two stages to ultimately identify a list of candidate resources for the given task's specification. In first stage, resources are selected by a simple match-making of each attribute of task's specification with static resource information. The resources on which the task is able to run are picked to be further evaluated according their runtime information. So in second stage, selected resources are checked for their availabilities during task EW and MetaAlloc finally identifies a list of candidate resources. For each of these candidates, MetaAlloc reservation negotiates with the resource local schedulers about resource provision and makes agreement on the availability of resources. For a list of discovered candidate resources, MetaAlloc requests reservation for resources during task's EW and this is considered as a negotiation process. If local schedulers grant this request, MetaAlloc chooses the one that can provide the earliest EW for the task. A reservation ID is returned which will be used to

access the reservation. If no reservation could be made for all the candidates, grace periods are added to the EW and MetaAlloc requests reservation again for other wall-clock periods within the EW until a reservation is made. If MetaAlloc cannot reserve resources for the task, metascheduler stops on this task and forwards the partial EP to EPExec to launch the job. During job execution, MetaAlloc attempts the resource allocation steps periodically for this task until decisions are made.

3.2.2 GridDAG Workflow System

GridDAG is our event-driven workflow system able to coordinate the execution of dependent tasks of a workflow job. Events are notification about status change of jobs or file transfers, data availabilities, or other situations defined by users for resource accounting and monitoring purpose. The event producers detects certain situation or change of status, generate the corresponding event messages and distributes them. The event consumers receive an event and then take certain actions or invoke event handlers. The GridDAG event mechanism is developed using WS-Notification standard [38] and the concepts of event, event producers and consumers map to the situation, NotificationProducer, and NotificationConsumer in WS-Notification specification. A Subscriber is an entity that acts as a service requester, sending the subscribe request message to a NotificationProducer.

As shown in Figure 2, four components are designed in GridDAG to support the eventing mechanisms, event chain builder, chain deployer, GridDAG agent, and DepResolver. The chain builder reads job execution plan forwarded from metascheduler and generate the event chains according to the EP. An event chain is a sequence of the events flowing between the participating producer and consumers in the predefined order. The chain builder also decides who are the producer, consumer, and Subscriber; and what events are going to be generated by each producer and to be received by each consumer. The chain deployer sends subscription requests to producers. A Subscription represents the relationship between a consumer, producer, and related event messages. These relationships constitutes the runtime event chains of a GridDAG job. GridDAG agents installed on each grid resources coordinate the runtime event activities in a distributed fashion by playing several roles at the same time. First, as the event producer, detects events occurred on the host resources, and generates and sends out event message. Secondly, as a consumer, receives messages about the availability of dependent files on remote resources or locally, and takes actions accordingly, such as pulling files. Another GridDAG module, DepResolver is configured to received all event notifications and keeps track of the states of tasks' dependencies (a task may have more than one dependencies). When all dependencies of a tasks are resolved, DeResolver takes certain actions, which are typically sending requests to EPExec for job submission or control.

3.2.3 EPExec Runtime system for Job Execution and Management

EPExec executes workflow jobs by carry out its Execution Plan. EPExec has three main functional modules, EXEpre (Execution Preparation), sJSCS (simple Job Submission and Control Service), and RTadj (Runtime Adjuster). EPExec's EXEpre fills in job EP the required information for job submission and workflow control. sJSCS is a simple utility answering requests from EPExec to submit single-executable jobs and control them. EPExec RTadj identifies differences between the job execution and the job EP, and makes certain adjustments on the execution so that it follows the EP.

When a job EP is forwarded to EPExec for execution, EPExec first calls EXEpre to setup execution related details; and then calls sJSCS to submit the job of the first task to its allocated resources, thus begins the execution cycle of the workflow job. Task job is submitted using its resource reservation ID and the task EWstarttime are set in the resource local scheduler. A successful submission returns a global job ID, which EPExec uses for job monitoring and control. During job execution, GridDAG agent and DepResolver work together to handle task dependencies. For data dependencies, file transfer can be in either destination-pull or source-push mode. In destination-pull mode, GridDAG events on source resources sends events about file availabilities to destination GridDAG agent, which then fetches files and sends events to GridDAG DepResolver about file arrivals. For source-push mode, when files are available, source GridDAG agent transfers them to the destination resources and send events to DepResolver indicating that the intermediate files have been transferred. As the overall coordinator of

dependency handling, DepResolver keeps track of the status of the dependencies of all tasks and decides whether dependencies of a task are all resolved.

If job execution does not follow the EP, RTadj is responsible to adjust the mismatch to make sure that tasks are executed on the allocated resources during the reserved time frame. RTadj categorizes job execution into four situations depending on how much task executions deviate from the EP. In the first situation, tasks are executing within their EWs and no adjustment is needed. In the second situation, a job completes after its EW, but the difference is within the grace periods of its child tasks so that they all can be started within their EW. In the third situation, the task's late completions go beyond the grace periods of their child tasks and cause their execution not to complete within the resource reservation time frame. Instead of killing those jobs, we configure local scheduler to allows them to finish. Currently, RTadj does not make adjustment for the situation two and three and relies on the allocated buffer time in task EW to automatically repair this. In the fourth situation, the tasks' late completions cause the expiration of reservation of its child tasks. In this case, EPExec submits these jobs without using reservation, yet the jobs may be held in the resource local queues. So, after submitting them, RTadj will request metascheduler to discover other resource for these tasks. If some resources are discovered and allocated, EPExec submits another copies of these tasks to these resources. During execution, EPExec kills the one that it thinks will be completed later than another one. In doing this, RTadj tries its best to make up the lost time in past job execution and minimizes the negative impacts on the execution of later tasks. If cannot make up these delays and it is almost impossible to follow the original plan, RTadj will consider re-scheduling for the rest of tasks. RTadj forwards the sub graph of job GridDAG to Metascheduler to do re-planning and re-allocating. Re-scheduling may cause low resource usage or wasting because of the cancellation of the reservation that has already been made. Metascheduler should avoid such cancellation by scheduling other jobs onto these reservations.

4 Conclusions

AQF is a typical application that requires several computational resources simultaneously and collaboratively available to produce air quality forecasting results in timely fashion. While a grid environment has potential to satisfy such requirement, lots of efforts are still needed to fill the gap between grid community and domain scientists. This paper presents our efforts to provide solutions for domain scientists to enable their applications on grid. Driven by AQF application and based on our past experiences of grid deployment in UH campus grid, the ongoing Gracce project attempts to provide an end-to-end solution for automatic application execution on grid environments. Using middlewares of Gracce, domain scientists are only required to specify their application logic structures and major resource requirements, Gracce is responsible to allocate grid computational resources for application tasks, launch the application and deliver the results back to users.

There are two major efforts in Gracce project in current stage: GAMDL and Gracce metascheduler. GAMDL provides a very intuitive means to model an application for grid computing. GAMDL's dataflow style in describing an application reflects the application original logics, requiring no efforts from users to extract application control-flow. The mvproperty concept makes GAMDL to be very powerful in describing similar application entities and a GAMDL description document is very concise and readable. Gracce metascheduling effort researches grid scheduling issues for workflow jobs, defines the term "Grid metascheduler" and proposes a workflow-orchestrated metascheduling architecture. The architecture integrates solutions to scheduling related issues in grid area, such as resource co-allocation, service negotiation and resource reservation, and workflow execution planning.

References

[1] A. Bose, B. Wickman, and C. Wood *MARS: A Metascheduler for Distributed Resources in Campus Grids,* Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing, 2004

[2] B.M. Chapman, P. Raghunath, B. Sundaram, Y. Yan, *Air Quality Prediction in a Production Quality Grid Environment,* Engineering the Grid: status and perspective, edited by J. Dongarra, H. Zima, A. Hoisie, L. Yang and B.D. Martino, Spring 2005

[3] B.M. Chapman, H. Donepudi, J. He, Y. Li, P. Raghunath, B. Sundaram and Y.Yan, *Grid Environment with Web-Based Portal Access for Air Quality Modeling,* Parallel and Distributed Scientific and Engineering Computing, Practice and Experience, 2003

[4] B.M. Chapman, Y. Li and B. Sundaram, and J. He, *Computational Environment for Air Quality Modeling in Texas,* Use of High Performance Computing in Meteorology, World Scientific Publishing Co, 2003

[5] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor and I. Wang, *Programming Scientific and Distributed Workflow with Triana Services,* Special Issue of Concurrency and Computation: Practice and Experience? 2005

[6] D. W. Byun, J. Pleim, R. Tang, and A. Bourgeois, *Meteorology-Chemistry Interface Processor (MCIP) for Models-3 Community Multiscale Air Quality (CMAQ) Modeling System,* Washington, DC, U.S. Environmental Protection Agency, Office of Research and Development, 1999.

[7] D. W. Byun, K. Schere, *EPA's Third Generation Air Quality Modeling System: Description of the Models-3 Community Multiscale Air Quality (CMAQ) Model,* Journal of Mech. Review, 2004

[8] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny, *Pegasus: Mapping Scientific Workflows onto the Grid,* Across Grids Conference 2004, Nicosia, Cyprus

[9] G. Grell, J. Dudhia, and D. Stauffer, *A Description of the Fifth-Generation Penn State/NCAR Mesoscale Model (MM5) NCAR/TN-398+STR,* NCAR Tech Notes http://www.mmm.ucar.edu/mm5/

[10] G. Mateescu, *Quality of Service on the Grid Via Metascheduling with Resource Co-Scheduling and Co-Reservation,* International Journal of High Performance Computing Applications, Vol. 17, No. 3, 209-218 (2003)

[11] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, *A Java Commodity Grid Kit,* Concurrency and Computation: Practice and Experience, vol. 13, no. 8-9, pp. 643-662, 2001, http:/www.cogkits.org/

[12] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit,* International Journal of Supercomputer Applications, Summer 1997.

[13] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation.* Intl Workshop on Quality of Service, 1999

[14] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, *A Security Architecture for Computational Grids,* ACM Conference on Computers and Security, 1998, 83-91.

[15] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations,* International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001.

[16] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, *GridFlow: Workflow Management for Grid Computing,* In Proceedings of 3rd International Symposium on Cluster Computing and the Grid, at Tokyo, Japan, May 12-15, 2003, p. 198.

[17] K. Czajkowski, I. Foster, and C. Kesselman, *Resource Co-Allocation in Computational Grids,* Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8), pp. 219-228, 1999.

[18] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, *SNAP: A Protocol for negotiating service level agreements and coordinating resource management in distributed systems,* Lecture Notes in Computer Science, 2537:153-183, 2002.

[19] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, *A Resource Management Architecture for Metacomputing Systems,* Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.

[20] R. Buyya, D. Abramson, and J. Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid,* The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), May 2000

[21] T. Fahringer, J. Qin and S. Hainzer, *Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language,* Proceedings of Cluster Computing and Grid 2005 (CCGrid 2005)

[22] W. F. Dabberdt, M. A. Carroll, D. Baumgardner, G. Carmichael, and R. Cohen *Meteorological research needs for improved air quality forecasting,* Report of the 11th Prospectus Development Team of the U.S. Weather Research Program, 2004.

[23] W. Smith, I. Foster, and V. Taylor, *Scheduling with Advanced Reservations.* Proceedings of the IPDPS Conference, May 2000.

[24] Z. Adelman and M. Houyoux, *Processing the National Emissions Inventory 96 (NEI96) version 3.11 with SMOKE,* The Emission Inventory Conference: One Atmosphere, One Inventory, Many Challenges, 1-3 May, Denver, CO, U.S. Environmental Protection Agency, 2001.

[25] BPEL4WS: Business Process Execution Language for Web Services Version 1.0, http://www.106.ibm.com/developerworks/webservices/library/wsbpel

[26] Community Scheduler Framework, http://www.platform.com/products/Globus/

[27] DAGMan (Directed Acyclic Graph Manager), http://www.cs.wisc.edu/condor/dagman.

[28] Gracce: Grid Application Coordination, Collaboration and Execution, http://www.cs.uh.edu/~yanyh/gracce

[29] Grid Resource Allocation Services Package (GRASP), http://www.moredream.org/grasp.htm

[30] Grid Scheduling Architecture Research Group, https://forge.gridforum.org/projects/gsa-rg

[31] High Performance Computing Center, University of Houston, http://www.hpcc.uh.edu

[32] Load Sharing Facility, Resource Management and Job Scheduling System, http://www.platform.com/products/HPC/

[33] Maui Moab Grid Scheduler (Silver), http://www.clusterresources.com/products/mgs/

[34] NCEP ETA analysis and forecast http://www.emc.ncep.noaa.gov, http://www.emc.ncep.noaa.gov/data

[35] Sun Grid Engine, Sun Microsystems, http://www.sun.com/software/gridware

[36] The Globus Resource Specification Language RSL v1.0, http://www-fp.globus.org/gram/rsl_spec1.html

[37] UH IMAQs - Institute for Multidimensional Air Quality Studies, http://www.imaqs.uh.edu

[38] Web Services Notification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

[39] XScufl Language Reference, http://taverna.sourceforge.net/docs/xscuflspecification.html

Appendix: GAMDL Description for AQF Application

```
<gridApp uid="uhaqf05" xmlns="http://.../gamdl"
     xsi:schemaLocation="... gamdl.xsd">
 <name>UH-AQF-2005</name>

 <mvproperty file="uhaqf.mvproperties"/>
 <mvproperty name="mdName($md,$dmsz,$day)">
  <value>uhaqf-$md-$dmsz-$day</value>
 </mvproperty>
 <include href="aqfexe.xml"/>
 <include href="aqffiles.xml"/>
 <include href="aqfmd.xml"/>

 <appExecutables>
  <executable uidRef="uhaqf-$md"/>
 </appExecutables>

 <appDataFiles>
  <file uidRef="$mdName($md,$dmsz,$day)-in1"/>
  <file uidRef="$mdName(mm5,$dmsz,$day)-in2"/>
  <file uidRef="$mdName(mm5,$dmsz,$day)-in3"/>
  <file uidRef="$mdName(cmaq,$dmsz,$day)-in2"/>
  <file uidRef="$mdName(cmaq,$dmsz,$day)-in3"/>
  <file uidRef="$mdName(cmaq,$dmsz,$day)-in4"/>
  <file uidRef="$mdName($md,$dmsz,$day)-out1"/>
  <file uidRef="$mdName(mm5,$dmsz,$day)-out2"/>
  <file uidRef="$mdName(mm5,$dmsz,$day)-out3"/>
  <file uidRef="$mdName(smoke,$dmsz,$day)-out1"/>
  <file uidRef="$mdName(smoke,$dmsz,$day)-out2"/>
  <file uidRef="uhaqf-postpv-$day-in1"/>
  <file uidRef="uhaqf-postpv-$day-out1"/>
 </appDataFiles>

 <appModules>
  <module uidRef="$mdName($md,$dmsz,$day)"/>
  <module uidRef="uhaqf-postpv-$day"/>
 </appModules>

 <appMdDeps>
  <CPsRship uid="smoke-mm5-$dmsz-$day-CPs">
```

```
 <childMd uidRef="$mdName(smoke,$dmsz,$day)"/>
 <parentMd uidRef="$mdName(mm5,$dmsz,$day)">
  <viaPipe>
   <pipeInFile uidRef="$mdName(mm5,$dmsz,$day)-out1"/>
   <pipeOutFile uidRef="$mdName(smoke,$dmsz,$day)-in1"/>
  </viaPipe>
 </parentMd>
</CPsRship>
<CPsRship uid="cmaq-smoke-$dmsz-$day-CPs">
 <childMd uidRef="$mdName(cmaq,$dmsz,$day)"/>
 <parentMd uidRef="$mdName(smoke,$dmsz,$day)">
  <viaPipe>
   <pipeInFile uidRef="$mdName(smoke,$dmsz,$day)-out1"/>
   <pipeOutFile uidRef="$mdName(cmaq,$dmsz,$day)-in1"/>
  </viaPipe>
  <viaPipe>
   <pipeInFile uidRef="$mdName(smoke,$dmsz,$day)-out2"/>
   <pipeOutFile uidRef="$mdName(cmaq,$dmsz,$day)-in2"/>
  </viaPipe>
 </parentMd>
</CPsRship>

<CPsRship uid="mm5-12k-36k-$day-CPs">
 <childMd uidRef="uhaqf-mm5-12k-$day"/>
 <parentMd uidRef="uhaqf-mm5-36k-$day">
  <viaPipe>
   <pipeInFile uidRef="uhaqf-mm5-36k-$day-out3"/>
   <pipeOutFile uidRef="uhaqf-mm5-12k-$day-in3"/>
  </viaPipe>
 </parentMd>
</CPsRship>
<CPsRship uid="mm5-4k-12k-$day-CPs">
 <childMd uidRef="uhaqf-mm5-4k-$day"/>
 <parentMd uidRef="uhaqf-mm5-12k-$day">
  <viaPipe>
   <pipeInFile uidRef="uhaqf-mm5-12k-$day-out3"/>
   <pipeOutFile uidRef="uhaqf-mm5-4k-$day-in3"/>
  </viaPipe>
 </parentMd>
</CPsRship>
<CPsRship uid="cmaq-12k-36k-$day-CPs">
 <childMd uidRef="uhaqf-cmaq-12k-$day"/>
 <parentMd uidRef="uhaqf-cmaq-36k-$day">
  <viaPipe>
   <pipeInFile uidRef="uhaqf-cmaq-36k-$day-out2"/>
   <pipeOutFile uidRef="uhaqf-cmaq-12k-$day-in4"/>
  </viaPipe>
 </parentMd>
</CPsRship>
<CPsRship uid="cmaq-4k-12k-$day-CPs">
 <childMd uidRef="uhaqf-cmaq-4k-$day"/>
 <parentMd uidRef="uhaqf-cmaq-12k-$day">
  <viaPipe>
   <pipeInFile uidRef="uhaqf-cmaq-12k-$day-out2"/>
   <pipeOutFile uidRef="uhaqf-cmaq-4k-$day-in4"/>
  </viaPipe>
 </parentMd>
</CPsRship>

<CPsRship uid="mm5-2d-1d-$dmsz-CPs">
 <childMd uidRef="uhaqf-mm5-$dmsz-2d"/>
 <parentMd uidRef="uhaqf-mm5-$dmsz-1d">
  <viaPipe>
   <pipeInFile uidRef="uhaqf-mm5-$dmsz-1d-out2"/>
   <pipeOutFile uidRef="uhaqf-mm5-$dmsz-2d-in2"/>
  </viaPipe>
 </parentMd>
</CPsRship>
<CPsRship uid="cmaq-2d-1d-$dmsz-CPs">
 <childMd uidRef="uhaqf-cmaq-$dmsz-2d"/>
 <parentMd uidRef="uhaqf-cmaq-$dmsz-1d">
  <viaPipe>
   <pipeInFile uidRef="uhaqf-cmaq-$dmsz-1d-out1"/>
   <pipeOutFile uidRef="uhaqf-cmaq-$dmsz-2d-in3"/>
  </viaPipe>
 </parentMd>
</CPsRship>

<CPsRship uid="postpv-cmaq4k-$day-CPs">
 <childMd uidRef="uhaqf-postpv-$day"/>
 <parentMd uidRef="uhaqf-cmaq-4k-$day">
  <viaPipe>
   <pipeInFile uidRef="uhaqf-cmaq-4k-$day-out2"/>
   <pipeOutFile uidRef="uhaqf-postpv-$day-in1"/>
```

```
    </viaPipe>
   </parentMd>
  </CPsRship>
 </appMdDeps>

 <startMdUid>uh-aqf-mm5-36k-1d</startMdUid>
</gridApp>
```