A Framework for Dynamic Resource Management on the Grid

Jyotishman Pathak, Jem Treadwell, Raj Kumar, Philip Vitale, Fernando Fraticelli
HP Laboratories Palo Alto
HPL-2005-153
August 22, 2005*

Grid resource management, Dynamic adaptation, resource provisioning, SLA management

A grid computing environment enables sharing of loosely coupled resources and services required by various applications in a large-scale. In such an environment, one of the key challenges is to develop a flexible, scalable, and self-adaptive resource management system which would allow users to carry out their jobs by transparently accessing autonomous, distributed, and heterogeneous resources. In this paper, we discuss the research issues and conceptual architectural design of such a dynamic resource management framework, which leverages the open-source Globus Toolkit and commercially available HP OpenView Configuration Management Solutions software (Radia). Our approach provides adaptive and scalable middleware for static and dynamic resource provisioning, resource monitoring, virtual organization-wide authorization, and business policy management. The framework is based on automated, policy-driven change and configuration management functionality that can dynamically adjust the size, configuration and allocation of various resources that will be consumed in the environment.

* Internal Accession Date Only
Approved for External Publication

A Framework for Dynamic Resource Management on the Grid

Jyotishman Pathak[1,3,*] **Jem Treadwell**[3] **Raj Kumar**[2] **Philip Vitale**[3] **Fernando Fraticelli**[3]

[1]Artificial Intelligence Lab, Department of Computer Science, Iowa State University, Ames, IA 50011, USA

[2]Hewlett-Packard Laboratories & Office of Corporate Strategy and Technology, Palo Alto, CA 94304, USA

[3]Hewlett-Packard Company, Mount Laurel, NJ 08054, USA

{firstname.lastname}@hp.com

Abstract

A grid computing environment enables sharing of loosely coupled resources and services required by various applications in a large-scale. In such an environment, one of the key challenges is to develop a flexible, scalable, and self-adaptive resource management system which would allow users to carry out their jobs by transparently accessing autonomous, distributed, and heterogeneous resources. In this paper, we discuss the research issues and conceptual architectural design of such a dynamic resource management framework, which leverages the open-source Globus Toolkit and commercially available HP OpenView Configuration Management Solutions software (Radia). Our approach provides adaptive and scalable middleware for static and dynamic resource provisioning, resource monitoring, virtual organization-wide authorization, and business policy management. The framework is based on automated, policy-driven change and configuration management functionality that can dynamically adjust the size, configuration and allocation of various resources that will be consumed in the environment.

Keywords: Grid resource management, Dynamic adaptation, Resource provisioning, SLA management.

1 Introduction

The development of grid computing technologies[20] over the past several years has provided us a means of using and sharing heterogeneous resources over local/wide area networks, and geographically dispersed locations. This has resulted in the ability to form loosely coupled, high-performance computational environment comprising numerous scalable, fault tolerant, and platform-independent services across the entire Internet. The grid infrastructure provides a way to execute applications over autonomous, distributed and heterogeneous nodes by secure resource sharing among individuals and institutions. Typically, a user can submit jobs to a grid without necessarily knowing (or even caring) where it will be executed. It is the responsibility of the grid resource management system to distribute such jobs among a heterogeneous pool of servers, trying to optimize the resource usage and provide the best possible quality of service. However, the design and implementation of such systems where clients can seamlessly execute their applications pose several challenges. Firstly, the grid is a dynamic framework where resources are subjected to changes due to system performance degradation, node failure, allocation of new nodes in the infrastructure, etc. As a result, a grid resource management system should have the capability to adapt to these changes and take appropriate actions to improve performance of various computing applications. For example, if a node running an application crashes, then the resource management system should migrate the jobs running in that machine to an alternate one. Secondly, typical grid resource requirements for an application vary during the course of its execution. Similarly, user demand patterns may also fluctuate, as a result of which, some resources may be over-utilized while others under-utilized. In such cases, it might be beneficial to provision additional resources (if all resources are over-utilized) or migrate jobs from an under-utilized resource and de-allocate it, hence ensuring optimal resource usage[1]. At the same time, the decisions for resource provisioning and sharing should be made while maintaining the autonomy of their environments and geographical locations. Thus, the resource management framework should provide a highly scalable and configurable approach for provisioning and securely accessing the resources. Also,

[*]This work was done while Jyotishman Pathak was an intern at Hewlett-Packard. Additional email: jpathak@cs.iastate.edu

[1]In some situations allowing jobs to execute in an under-utilized node (resulting in faster execution) might be optimal, as opposed to migrating them. Such decisions could be made using various optimization heuristics[35, 37] or policy-based modeling[31].

lately there has been interest in using grid computing for e-Business and e-Commerce applications which demand stricter service guarantees. Typically, such guarantees are specified as part of agreements, which need to be monitored and assured by the grid resource management system.

Keeping these issues in mind, in this paper we investigate a framework for dynamic grid resource management using the open-source Globus Toolkit[19]—a software toolkit for building grid infrastructure—and commercially available HP OpenView Configuration Management Solutions software (Radia)[3]—an automated, policy-driven change and configuration management infrastructure for dynamic allocation of physical system and data resources. More specifically, we introduce the architecture of ***Globus-Radia Resource Management System*** **(GRRMS)**—a framework for static and dynamic resource provisioning, resource monitoring, virtual organization-wide resource sharing, SLA management and resource brokerage. The main goal of this system is to provide seamless access to users for submitting jobs to a pool of heterogeneous resources, and at the same time, dynamically monitoring the resource requirements for execution of applications and adapting to those needs in accordance with the changing load characteristics of the underlying resources. The architecture has been designed to be a pluggable core component for resource management processes which leverage various services provided by the existing technologies, namely the **Globus Toolkit** (hereafter ***Globus***) and **HP OpenView Configuration Management Solutions** software (also known as ***Radia***). GRRMS implements various WSRF-compliant[14] services to provide the dynamic resource management functionalities required in a typical grid computing environment.

The rest of the paper is structured as follows: in Section 2, we give a brief introduction to Globus and Radia software. Section 3 describes the architecture of GRRMS in detail. Related work is presented in Section 4, while Section 5 concludes with a summary of our work.

2 Existing Technologies

In what follows, we provide an overview about the main functionalities of Globus and Radia software.

2.1 Globus

The development of Globus began in the early 90's to support the building of distributed computing applications and infrastructures[19]. Globus has been used to build computational Grids and Grid-based applications by enabling sharing of computer power, data storage, and various other facilities online, spanning the Internet across institutional and geographic boundaries in a secured manner. The latest release of the software, Globus Toolkit version 4.0, provides significant Web services implementation advancements over its previous counterparts in terms of features, compliance to various emerging standards (e.g., WSRF[14]), and usability. The following is a brief description of some of its core components:

• **Security:** Globus security tools build on the Grid Security Infrastructure (GSI)[21] for authenticating users and services, secure communications, and authorization. These tools support functions such as managing user credentials, delegation of credentials and maintaining group membership information. Examples of the tools include Community Authorization Service, Delegation Service etc.

• **Execution Management:** The execution management tools enable initiation, monitoring, management, scheduling, and/or coordination of remote computations. Globus implements the Grid Resource Allocation and Management (GRAM) service for providing these functionalities. In order to address issues such as data staging, delegation of proxy credentials, and job monitoring and management, the GRAM server is deployed along with Delegation and Reliable File Transfer (RFT) servers.

GRAM typically depends on a local mechanism for starting and controlling the jobs. To achieve this, GRAM provides various interfaces/adapters to communicate with local resource schedulers (e.g., Condor[10], PBS[8], LSF[7]) in their native messaging formats. The job details to GRAM are specified using an XML-based job description language, known as Resource Specification Language (RSL). RSL provides syntax consisting of attribute-value pairs for describing resources required for a job, including memory requirements, number of CPU's needed etc.

• **Information Services:** The information services provide static and dynamic data about the various Grid resources. These services are mainly concerned with gathering, indexing, archiving, processing, and distributing information about the configuration and state of various resources and services over the Grid. The Globus component within this package is called Monitoring and Discovery Service (MDS). Globus uses its WSRF core and WS-Notification[13] interfaces to simplify the tasks of registering information sources and locating and accessing the required information. Other Globus components, such as GRAM and RFT, also de-

fine various resource properties, providing a basis for monitoring and discovery.

• **Data Management:** The data management package provides many utilities and tools for transmitting, storing and managing large amounts of data required for various Grid-based applications. The elements of this package include GridFTP service, Reliable File Transfer (RFT) service, Replication Location Service (RLS) etc.

2.2 Radia

Radia[3] implements an automated, policy-driven change and configuration management system for dynamically adjusting the size, configuration and allocation of physical system and data resources. It provides a highly adaptable, flexible, and automated approach to resource provisioning for an infrastructure. Radia adopts an object-oriented technology for transforming software and content from file-based media into self-aware, platform-independent, intelligent objects that can automatically assess the environment into which they are deployed, and have the capability to install, update and repair themselves accordingly. This deployment of resources is governed by a *distribution model* or *desired state* which records the identities and intended configurations of the various nodes (including deployment destinations e.g., PDAs, desktops) that are part of the Radia-managed infrastructure. Figure 1 shows the elements for such a distribution model, which we categorize and explain in detail in the remainder of this section.

The various components of the Radia Distribution Model can be divided into four main categories:

• **Radia Management Applications:** The Radia Management Applications are the client-side components that allow for automated deployment (of software), updates, repairs, and deletion activities. They also have the ability to inspect the hardware and software configurations of the client machine.

There are three types of Radia Management Applications: *Radia Application Manager* allows the administrator to control the distribution of applications. Using this software, the administrator can select, install, uninstall, and update a subscriber's software and content automatically in a seamless way. *Radia Software Manager* allows subscribers to install, delete, verify, and update their own elective software and content via a user interface. However, the administrator still decides which software and content each subscriber uses. The *Radia Inventory Manager Client* gathers information about software and hardware configurations automatically and stores them in a centralized server location. This information can be viewed as reports. Typical information gathered might be the amount of RAM in a computer, hard disk capacity, processor type, versions of OS running etc.

• **Radia Management Infrastructure:** This module is the heart of all Radia activities. It is used to maintain the desired state or distribution model, store software and content packages, automate software management activities, and perform various administrative functions. The core of this infrastructure consists of the *Radia Configuration Server* and *Radia Administrator Workstation*.

The Radia Configuration Server dynamically generates the distribution model based on situation-specific data, creating a software environment that automatically adapts to changes in a user or machine environment. The server synchronizes distribution of objects such as application packages, computer configurations, and policy relationships across the network automatically. Typically, a policy defines which services subscribers, client computers or managed devices are entitled to. These policies are stored in the Radia Configuration Server Database. It also stores the various software components (e.g., application packages) that Radia distributes and the security and access rules for Radia administrators.

The Radia Administrator Workstation provides various tools, namely Radia Publisher, Radia System Explorer, Radia Client Explorer and Radia Screen Painter, for centralized control of Radia objects and entitlements. These tools carry out basic Radia functions such as manage Radia database, prepare applications for deployment, view Radia client objects etc. The Radia Publisher provides an interface for packaging all software for deployment and post-deployment management. It determines what packages to install by scanning the destination node before and after installing the software. The Radia System Explorer allows the administrator to view and configure policy and application services stored in the Radia database. Using the explorer, the administrator can modify application packages after the initial publishing process or define new application service prerequisites and policies for application entitlements. The administrator also uses the Radia Client Explorer to manipulate (create new, view/edit existing) objects, and the Radia Screen Painter to create new customized dialog boxes.

• **Radia Extended Infrastructure:** The Radia Extended Infrastructure provides a scalable solution to software management services across an enterprise. Through distributed administrative capabilities, repli-

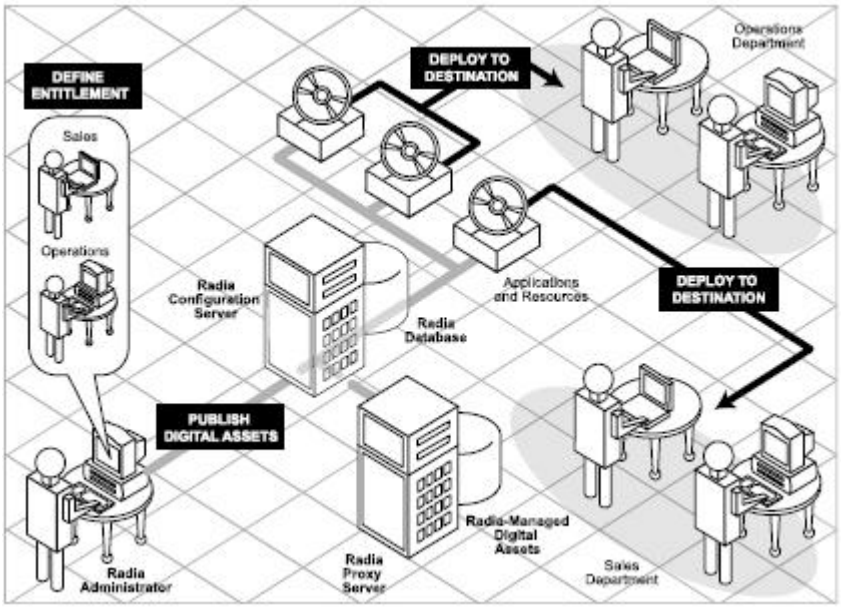

Figure 1. Elements in Radia Distribution Model[3]

cation services, and bandwidth conservation and metering, it gives an end-to-end management solution. This infrastructure comprises the following components:

Radia Distributed Configuration Server allows sharing of information about policies and management content in an enterprise-wide network comprising more than one Radia Configuration Server. It can automatically synchronize distributed Radia databases allowing managed applications and policy information to be shared across the enterprise.

The *Radia Multicast Server* enhances and simplifies data-transmission technology by reducing the number of transmissions necessary, optimizing the use of network bandwidth. Such an approach transmits the same data stream to many receivers simultaneously as opposed to transmitting individually. However, the server makes sure that only the participants eligible for reception receive the data and only the data requested is transmitted. The information required to determine whether a client is eligible is stored in the Radia database.

The *Radia Integration Server* integrates independent modules, giving them access to all the functions and resources under its control. This server comprises the following modules: *Radia Management Portal*, provides a web-based single access point from which Radia administrators can deploy Radia client components, detect the current status of some Radia components, manage the Radia database, and track the completion status of many tasks. It also exposes the following Web services - `GroupManagement` service for managing device groups, their attributes and memberships, `ServiceManagement` service for managing various Radia services, their availability and deployment, `PolicyManagement` service for managing the definition of policies in devices, groups etc., `JobManagement` service for managing jobs that are created and monitored and `EntityManagement` service for managing entities that can access the various Radia objects. The *Radia Inventory Manager Server* stores the hardware and software information collected from various machines by the inventory manager client. This server enables centralized reporting and administration based upon the discovery results. The *Radia Mobility Server* acts as an interface between the mobile device and the Radia Configuration Server for mobile devices distribution activities. The *Radia Proxy Server* localizes digital content enabling Radia-managed devices to receive application data over LANs as opposed to WANs, as a result reducing the overall network traffic. Such servers are beneficial in environments where many clients request the same resource to be deployed from the same location. Placement of proxy servers at strategic locations in a network improves its efficiency. *Radia Staging Server* also helps in reducing the network traffic. Typically, when a client first connects to the Radia Configuration Server to retrieve an application package, a copy of the software is sent from the Radia Configuration Server to the Radia Staging Server. The

next client that connects to the Configuration Server is re-directed (based on network locality-aware heuristics such as[36]) to obtain its application from the Staging Server. This also reduces the workload on the Configuration Server.

• **Radia Management Extensions:** These extensions provide enterprise integration and extended functionality. The following are the various extensions: *Radia Extensions for Windows Installer* is a management system which gives complete control over resource gathering and analysis of Windows installer packaging. It provides the ability to build, test, maintain, deploy and troubleshoot applications and installation packages. The *Radia Policy Server* is used for administration purposes. It integrates with LDAP servers, Novell NDS etc. to enable single source points of control for user authentication, access policies and subscriber entitlement. The Radia Configuration Server can be configured to query the Policy Server to determine what applications and software components should be distributed and managed for a particular subscriber. The *Radia Adapter for SSL* utilizes the latest encryption and security protocols for secured network and software distribution. The *Radia Publishing Adapter* is a tool that allows automated, unattended updates to application packages. It identifies a set of files and components, and publishes them, in a controlled, automated, and repeatable manner, to the Radia database where they are stored as objects (to be deployed).

3 Globus-Radia Resource Management System

This section describes the conceptual architectural design of Globus-Radia Resource Management System (GRRMS). It provides an automated, scalable, highly configurable and adaptable solution to dynamic resource management for a grid environment. One of the key features of this framework is the ability to automatically provision (and de-provision) resources for execution of applications based on the changing load characteristics of the environment, thereby ensuring optimal resource usage. The GRRMS architecture, shown in Figure 2, implements a set of WSRF-compliant services to provide various functionalities for authorization, resource discovery, job monitoring and management, SLA management, and resource provisioning as part of the Globus-Radia Broker (GRB). In what follows, we describe these components in detail.

3.1 Globus-Radia Broker

The Globus-Radia Broker (GRB) provides a gateway to clients for interacting with GRRMS. The aim of this module is to efficiently use various components of GRRMS and control the whole process of job and resource management. Typically, a client interacts with the broker by providing its own proxy credentials, which are authenticated and authorized by the Authorization Manager (Section 3.3). Once the client gains access to GRRMS, it can perform various actions (e.g., submit a job request for execution, query for job status, request for advance reservation of multiple resources). GRB provides generic interfaces for achieving these functionalities by communicating with different modules within GRRMS.

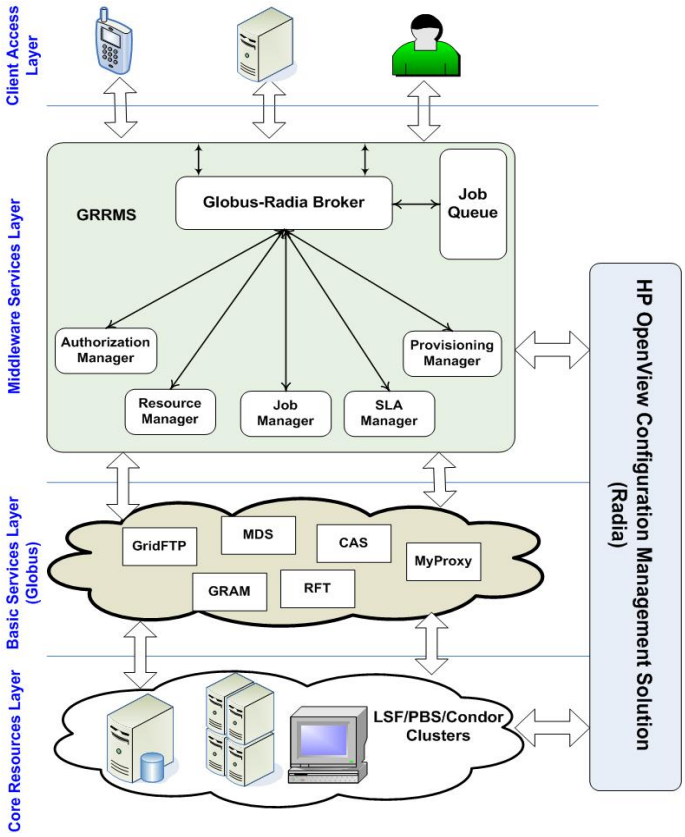

Figure 2. GRRMS Architecture

3.2 Job Queue

This module provides a secure (GSI-enabled[21]) `JobQueue` interface for GRRMS, using which clients can submit job requests. The module validates each incoming job description and puts the job in the job

queue. The job queue stores the jobs which are ready for execution and implements First In First Out (FIFO) strategy, although in principle, it can implement any alternative queue management algorithm (e.g., Shortest Job First, Negotiated Priorities). `JobQueue` also communicates with the `JobManager` (Section 3.5) via GRB, as a result of which clients can monitor the status of their jobs (either by querying or subscribing to notifications). The job descriptions in our framework are specified using Resource Specification Language (RSL), an XML-based job description language for Globus. However, the grid community is currently working on an (abstract) Job Submission Description Language[4] which is independent of language bindings and provides a rich set of features and attribute semantics. We are following these developments and will adhere to the standards as and when available.

3.3 Authorization Manager

Globus provides a secure framework for submitting and executing jobs and applications on various grid resources. It uses the GSI[21] infrastructure for authenticating users and services, secure communications and authorization. This infrastructure is based on a static mapping from the user's Distinguished Name (DN) to a local user-id using a *grid-mapfile*. While this approach provides the benefits of secure single-signon authentication, for our purposes we need a framework which is more scalable (support for several thousands of users), extensible (support for detailed authorization rules) and expressive (support for authorization policies between Virtual Organizations and communities).

GRRMS provides these features with the *`AuthorizationManager`* service. This service acts as a mediator between the clients and resource providers for authentication and authorization. It provides a registry (based on My-Proxy[19]) to which all the potential grid users and services/sites are registered. The various grid resources and sites belonging to a particular Virtual Organization (VO) or community register via a local identity. Typically, the registry stores information about user profiles, proxy credentials authorized by a Certificate Authority (CA), and user access control policies. Such policies describe 'who, what and when' issues related to user access, essentially specifying resources, users, and when users have permission to access the resources. These policies are represented using Extensible Access Control Markup Language (XACML)[2]. The `AuthorizationManager` service implements an authorization server based on Sun's open-source XACML engine[9]. The server grants or denies authorization to users based on their associated access policies (specified by the administrator) and generates a temporary proxy credential for resource access, valid for a particular time period. However, in a multi-VO configuration, it is possible that the requesting user does not have access to the referenced VO. In such situations, the `AuthorizationManager` service maps the user's credentials to a guest/template user credential to provide authorization. These guest accounts are also specified as part of the XACML policy and, in general, have limited access and authorization capabilities.

3.4 Resource Manager

The Resource Manager is responsible for finding and monitoring appropriate resources for execution of jobs and applications over the grid and provides a *`ResourceManager`* interface. It relies on static and dynamic data provided by a set of information and monitoring services. Static data is the information that does not vary during run-time (e.g., OS type), whereas dynamic data refers to variables affecting the actual resource availability (e.g., CPU usage). In our current framework, the `ResourceManager` communicates with Globus's Monitoring and Discovery Service (MDS) and Radia's Inventory Manager Server (IMS) for providing such information. Both these services provide centralized access to the various hardware and run-time configuration-related information about the grid resources. Typically in a grid environment one can envision multiple independent projects or application suites making use of many heterogeneous clusters and services. Similar to [32], we assume a hierarchically structured set of clustered resources, spanning multiple VOs, such that they deploy information providers (e.g., Ganglia[24], Hawkeye[10]) for local/host-level monitoring and allow MDS & IMS access to scheduler and cluster information. The `ResourceManager` filters the resources by comparing the dynamic resource information received from MDS & IMS and static data with the job requirements received from the client. Once a resource (or a set of candidate resources) is selected, the manager interacts with the `AuthorizationManager` interface to determine if the requested user has access and authorization to the resource.

However, in certain cases it might be that there are insufficient resources available for execution of a particular job (based on its requirements). In such situations, the `ResourceManager` communicates with the Provisioning Manager (Section 3.7) via GRB, for provisioning of new resources.

3.5 Job Manager

The Job Manager is responsible for job submission, monitoring, logging and rescheduling. The manager comprises several other subsystems which work cooperatively to provide these functionalities. It exposes a *JobManager* interface via which the Globus-Radia Broker (GRB) can interact. A particular job execution request from the client is forwarded to the *JobExecution* interface which provides the job launching facility. It implements customized methods for submitting jobs to various local resource managers of the grid (e.g., GRAM[19], Condor-G[10]). These local resource managers also provide job status change information, which can be either queried for or subscribed (for notifications[13]) via the interface. The Job Manager module is also responsible for logging job status-related information in a database together with some additional data (e.g., user id, job id, resource requirement, submission time, total execution time, wait time etc.). Access to such historical information not only helps clients in decision-making, but also allows implementation of advanced (meta) scheduling algorithms (e.g., prediction based scheduling). In addition, such information can be used for implementing a usage-based 'chargeback model' (e.g., GridBank[11]). Finally, the manager module implements a *ReScheduler* interface for providing dynamic adaptability to changes in the load characteristics of the resources on the grid. Specifically, it enables job/application migration between resources. To achieve this, `ReScheduler` leverages the checkpointing and job migration functionalities provided by the local resource management systems (e.g., Condor[10], LSF[7]). During the migration process, it communicates with the `AuthorizationManager` interface to ensure that the user has access and authorization to use the resource. `ReScheduler` implements three modes of migration: *on-demand, timeserving,* and *failure-based*[2]. During the on-demand mode, the client (via GRB) may send a request to migrate a job from one resource to another (e.g., because of pre-emption by a higher-priority job). In the timeserving mode, the `ReScheduler` interacts with `JobExecution` and `ResourceManager` (Section 3.4) interfaces to gather real-time data about job and available resource status. Based on this information, it may make a greedy choice to migrate selected job(s) such that its total execution time is reduced. Finally, in the case of failure-based migration, the `ReScheduler` migrates a job to an alternate resource due to a resource failure (e.g., server crash). Note that in each of these three migrating modes it might be possible that there are not adequate resources available to migrate a particular application. In such situations, the `ReScheduler` interacts with the Provisioning Manager (Section 3.7) to attempt to allocate additional resources.

[2]Failures solely correspond to the malfunctioning of resources, and not the various components of GRRMS.

3.6 SLA Manager

The increasing adoption of grid computing in the commercial space has shifted its principal of operation of what used to be a 'best effort approach'[30]. With this shifting focus, there is an emerging need to consider and incorporate *agreements* as part of the software building process. These agreements represent expectations and responsibilities of different parties involved about the functionality and performance of systems and services. They also help support automated adaptation and management in the presence of multiple interests[27]. Such contracts are called *Service Level Agreements* (SLAs). SLAs encompass both the functional and non-functional requirements of the resources. In addition, they cover the cost of consuming a resource, as well as a penalty for breaking the contract. In our framework, these SLA-related issues are handled by the SLA Manager. It exposes an *SLAManager* interface by which the clients can interact (via GRB). The clients formalize their requirements unambiguously using SLAs, which are published and stored using the *SLAPublishing* interface. The *SLANegotiator* engine acts as a broker between such demands of the client and the capabilities of the resources. This engine interacts with the `ResourceManager` interface (Section 3.4) to discover resource(s) which meet the SLA requirements. It also maintains an ordered list of resources based on their (historical) performance rating. Upon successful completion of a job, it requests client feedback (about the resources utilized for that particular job), which is used for rating the performance of the resources. Such a rating is used for assigning *best of breed* rankings among alternative resources for execution of a particular application.

However, simply having an SLA-based negotiation mechanism is not sufficient to guarantee that the requirements will be fulfilled. To assure correct behavior of the system and ensure that SLAs are not violated, we need to monitor the resources involved in order to detect failures and, if needed, migrate jobs to other available resources. This is done by the *SLAMonitoring* module. The module communicates with the `ResourceManager` and `JobManager` interfaces to determine the state of the jobs as well as the resources executing those jobs. Based on the status, it

identifies and takes action if there is any violation of the negotiated SLA between the client and the provider. For example, if there is a hardware resource failure, then the `SLAMonitoring` engine activates the 'failure-based' re-scheduling mode of the `ReScheduler` for migrating jobs to alternate resources. At the same time, the engine penalizes the failed resource by reducing its performance rating. In addition, the SLA Manager provides an *`SLAProvisioning`* interface using which the clients can determine appropriate resources (e.g., based on functional/non-functional capabilities, performance ratings etc.) for executing their application and make advance reservations (also called *static provisioning*). This interface can also be used for *dynamically provisioning* additional resources (based on the client's requirements) during run-time, if such resources are not available for migrating (and further executing) the client application. Similarly, if a particular job has exceeded its *wait time* (as specified in the SLA) in the job queue, the monitoring engine might invoke 'on-demand' provisioning of new resources for execution of the job. Such active monitoring capabilities are vital to ensure SLA compliance. To provide these functionalities, the interface interacts with the Provisioning Manager (Section 3.7).

The agreements in our framework are defined using the WS-Agreement specification[12] of Global Grid Forum. This specification defines a protocol for establishing agreement between two parties and provides an XML-based representation for specifying them.

3.7 Provisioning Manager

The Provisioning Manager is responsible for allocation and deployment of resources needed to execute a client application. It provides a *`ResourceProvision`* interface via which the various modules of GRRMS can interact. In our framework, we use Radia[3] as a tool for provisioning of hardware and software resources. Radia provides a Web services-based API (called Radia Management Portal) for basic management of resources and their deployment. As mentioned in Section 2.2, the provisioning of resources in Radia is determined by a *distribution model*. These models are created dynamically by the Radia Configuration Server (based on situation-specific data) and are specified (by the Radia administrator) using XML-based 'policies', which broadly defines *what* application needs to be installed/deployed *where*, *when* and *how*. They also specify software dependency, i.e., the deployment relationship with other software components, operating systems and hardware. The policies are stored in the Radia Database, and queried by the Configuration Server during run-time. Similar policies can also be defined for resource de-allocation, wherein applications are terminated and/or uninstalled.

Typically, the `ResourceProvision` interface receives resource allocation requests from various components of GRRMS—namely Resource Manager, Job Manager and SLA Manager. An example of such a request could be '*Allocate an HP-UX 11i node with 20GB of free hard disk space and 2Gigabit network connectivity into a local Condor pool.*' Upon receiving such requests, the interface interacts with the Radia Inventory Manager Server to determine an appropriate destination node with appropriate hardware requirements (in this case, 20GB free hard disk space and 2Gigabit network connection). Assuming that such a node is available (and hence selected), `ResourceProvision` submits an allocation request to the `JobManagement` Web service of the Radia Management Portal (Section 2.2) by providing destination node details, and the action to perform (in this case, installing Condor[10] daemons, setting the appropriate CONDOR_HOST, and starting the CONDOR_MASTER). The `JobManagement` service submits this request to the Radia Configuration Server, which creates an appropriate distribution model for resource allocation. The Configuration Server interacts with various other Radia components (e.g., Radia Publisher for packaging all the software for deployment, Radia Policy Server for securing access to the node etc.) during this process. The `JobManagement` service monitors the progress of the actions and updates the `ResourceProvision` interface. On successful provisioning, the Radia Inventory Manager Server is updated, as a result of which the Resource Manager (Section 3.4) becomes aware of the newly provisioned resource. In addition, the Provisioning Manager provides an interface for `ActiveMonitoring`. This service constantly communicates with the Resource Manager and Job Manager (Section 3.5) to gather up-to-date status information about the various jobs that are running as well as the health of the resources as a whole. Based on this run-time status and requirements of the infrastructure, the service determines whether to allocate or de-allocate additional resources. Such active monitoring results in optimal resource usage. For example, if the infrastructure is over-subscribed (e.g., CPU utilization exceeds a certain threshold) and still more jobs are waiting in the job queue, then `ActiveMonitoring` will direct the `ResourceProvision` service to allocate more resources.

GRRMS provides two basic modes of provisioning: *static* and *dynamic*. During the static mode, clients can request provisioning of resources before executing their application. To achieve this, `ResourceProvision`

provides various generic methods for creation, modification, monitoring, and cancellation of a provisioning request. In our framework, these methods implement the interfaces provided by the Maui Cluster Scheduler[5], but in principle any reservation manager framework [34] can be adopted. However, if a particular reservation cannot be fulfilled (e.g., due to non-availability of resources at the requested time), then `ResourceProvision` submits a request to Radia for allocation of additional resources based on the client's requirements. On the other hand, in the dynamic provisioning mode, the client is not aware apriori of the resource where its application will be executed. Instead, provisioning requests are coordinated and formulated at run-time by the various components of GRRMS.

4 Related Work

Over the past several years, there has been a lot of work towards the development of grid middleware technologies. A good survey of such approaches can be found in [23]. Due to space limitation, we discuss only a few other projects (which were not introduced in [23]).

UNICORE[16] is a vertically integrated grid computing environment that facilitates seamless, secure and intuitive access to resources in a distributed environment. It adopts a 3-tier architecture and provides a client-side graphical interface allowing them to create Abstract Job Object (AJO) represented as a serialized Java object. The UNICORE Network Job Supervisor (NJS) incarnates these AJOs into target system specific actions. NJS also manages the submitted jobs and performs user authorization (single sign-on through X.509 certificates). The UNICORE Target System Interface (TSI) accepts incarnated job components from NJS, and passes them to the local batch schedulers for execution.

Gridbus[11] is an open-source middleware toolkit for computational grids. It provides a service-oriented cluster and grid middleware for e-Business and e-Science applications. Its design and development is based around the notion of 'utility computing' and uses various economic models for efficient management and usage of shared resources. Gridbus provides an array of software for: Grid Economy and Scheduling (Economy Grid), Data Grid Brokering and Scheduling (Gridbus Broker), Cluster Economy and Scheduling (Libra), .NET-based Grid computing (Alchemi), Grid Market Directory and Service Publication (GMD), Grid Simulation (GridSim toolkit), Resource Usage and Accounting (GridBank) and Grid Portals (GridScape, GMonitor).

Another on-going project which provides a suite of software for various grid middleware functionalities is GridLab[15]. The GridLab framework is based around a Grid Application Toolkit (GAT), which implements a set of high-level APIs, using which clients and applications are able to call the underlying grid services.

NAREGI[25] aims to research and develop high-performance, scalable grid middleware for the Japanese national scientific computational infrastructure. The implementation of the NAREGI framework is divided into six R&D groups, also called 'Work Packages' (WPs): WP-1 focuses on resource management and managing information services; WP-2 works on basic parallel programming tools; WP-3 develops grid tools for end users; WP-4 deals with packaging and configuration management of the software products from the project; WP-5 investigates issues related to networking, security, user management; and WP-6 is in charge of grid-enabling various nanoscience applications.

The ASKALON[18] project provides a tool set for service-based performance-oriented development of grid applications. The tools implemented can be broadly categorized as: resource brokerage and monitoring, scalable discovery and organization of resources, workflow-based dynamic and fault-tolerant execution of activities, meta scheduling, performance prediction for estimation of job execution time, and performance monitoring, instrumentation and analysis of grid applications. The GLARE system[33], developed as part of the ASKALON framework, allows automatic deployment and on-demand provisioning of components for grid applications. In this work, the authors adopt a peer-to-peer architecture for resource management and consider 'activities' as generalized abstractions of grid tasks/jobs, which can be deployed during run-time on different computers and executed in a coordinated manner to accomplish a particular application goal.

GrADS[35] (Grid Application Development Software) is a framework designed to solve numerical applications over the grid. An important aspect of this framework is the ability to decide when to migrate jobs based on actual and predicted execution times. If the system detects minimal differences between the actual and predicted performance of the application, it avoids job migration. Such an approach makes those jobs which are about to finish execution in a short period of time less susceptible to suspension and migration.

Othman et al.[28] implemented an adaptive grid resource broker using reflection technology to facilitate adaptation during run-time to variations in system resources, such as load or CPU utilization of a particular job. They also implement prediction based modeling that is used as a basis for a decision as to whether

job migration is required to satisfy a pre-specified time constraint.

SmartFrog[22] is a framework for service configuration, description and lifecycle management. The framework adopts a component-based architecture and provides a declarative language for describing service configuration and provisioning. This language is used to design code templates which are executed by SmartFrog engines running on remote nodes. The SmartFrog component model enforces lifecycle management by transitioning components through various stages (installed, initiated, started, terminated, failed), which in turn allows the SmartFrog engines to redeploy components during a failure.

Apart from the above mentioned approaches, the Global Grid Forum CDDLM working group[1] is also working towards development of specifications for describing configuration of services, their deployment on the grid, and management of their life cycle. In addition, there is on-going work in the Open Grid Services Architecture (OGSA) Basic Execution Service Working Group[6] which focuses on many aspects of job execution and management. We are closely following these developments and will adhere to the specifications in future.

There has also been an increasing amount of work on grid resource management based on Service agreements. An overview of such approaches can be found in[27]. Sahai et al.[29, 30] proposed an architecture for SLA monitoring to ensure stricter service guarantees. Their architecture provides a language for specifying SLAs and relies on a network of communicating proxies, each maintaining SLAs committed within the administrative domain of the proxy. SLAs are either negotiated between or specified to management proxies, which are responsible for automated monitoring and triggering of appropriate actions for the agreements. The problem of resource allocation for SLA-constrained grid applications is discussed in [26]. In this system, an application is decomposed into simple tasks which exhibit precedence relationships. Then, a heuristic is used to optimally allocate resources for running the tasks by minimizing total cost and preserving execution time SLAs. Burchard et al.[17] introduced the Virtual Resource Manager which builds on existing resource manager systems and provides various components for QoS management. The granularity for the deployment of these components (decided by the local administrator) effects the overall QoS management offered by the system.

Our framework is also motivated by the approaches mentioned above. However, one of the key facets of our approach is the ability to dynamically monitor and adapt to the run-time resource requirements of the various applications that are executing. The Resource Manager, Job Manager, and SLA Manager constantly monitor the overall functioning of the infrastructure to ensure optimal resource usage as well as strict guarantee of service agreements. For example, if additional resources are needed for execution of a particular job, the Provisioning Manager gives GRRMS the ability to provision brand-new resources based on such requirements. To the best of our knowledge, none of the proposed approaches deals with this problem of requirements-based resource provisioning. At the same time, GRRMS implements an XACML-policy based authorization framework. Such an approach is not only scalable and flexible in a grid environment, but in a way realizes the true meaning of 'Virtual Organizations'[20].

5 Conclusion

In this paper, we introduced the conceptual architectural design of Globus-Radia Resource Management System (GRRMS)—a modular and extensible approach for dynamic resource allocation and management in a grid environment. The framework provides a pluggable component by leveraging a set of core services provided by two existing technologies: Globus and Radia. The novelty of this approach is the ability to provision resources based on the job requirements. Such automated and highly adaptive systems will play an important role in realizing next-generation grid applications. Apart from this, the framework also provides a flexible approach for SLA-based service management and an XACML-based VO-wide authorization mechanism, which are vital for realizing commercial grid applications. Our future work is targeted towards a prototype implementation of various components of GRRMS. We also plan to include additional features (e.g., prediction-based scheduling and migration, chargeback model for utility computing applications) into our framework.

Acknowledgment. The authors would like to thank Greg Astfalk from Hewlett-Packard for his encouragement and continued support in pursuing this research.

References

[1] Configuration, Deployment and Life Cycle Management of Grid Services, https://forge.gridforum.org/projects/cddlm-wg.

[2] Extensible Access Control Markup Language, http://www.oasis-open.org/committees/xacml.

[3] HP OpenView Configuration Management Solutions Software, http://managementsoftware.hp.com/solutions/ascm.

[4] Job Submission Description Language, https://forge.gridforum.org/projects/jsdl-wg.

[5] Maui Cluster Scheduler, http://www.clusterresources.com/products/maui/.

[6] OGSA Basic Execution Service Working Group, https://forge.gridforum.org/projects/OGSA-BES-wg.

[7] Platform's Load Sharing Facility, http://www.platform.com/products/LSF.

[8] Portable Batch Scheduler, http://www.openpbs.org.

[9] Sun's XACML Implementation, http://sunxacml.sourceforge.net.

[10] The Condor Project, http://www.cs.wisc.edu/condor.

[11] The Gridbus Project, http://www.gridbus.org.

[12] Web Services Agreement, https://forge.gridforum.org/projects/graap-wg.

[13] Web Services Notification, http://ws.apache.org/pubscribe.

[14] Web Services Resource Framework, http://ws.apache.org/wsrf.

[15] G. Allen, K. Davis, and et al. Enabling Applications on the Grid: A GridLab Overview. *Intl. Journal of High Performance Computing Applications*, 17(4):449–466, 2003.

[16] J. Almond and D. Snelling. UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce. *Future Generation Computer Systems*, 15(5-6):539–548, 1999.

[17] L.-O. Burchard, B. Linnert, F. Heine, M. Hovestadt, O. Kao, and A. Keller. A Quality-of-Service Architecture for Future Grid Computing Applications. In *IEEE Parallel and Distributed Processing Symposium*, 2005.

[18] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. S. Junior, and H.-L. Truong. ASKALON: A Tool Set for Cluster and Grid Computing. *Concurrency and Computation: Practice and Experience*, 17(2-4):143–169, 2005.

[19] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl. Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[20] I. Foster and C. Kesselman. *The GRID: Blueprint for a New Computing Infrastructure, 2nd Edition*. Morgan-Kaufmann, San Mateo, CA, 2004.

[21] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *5th ACM Conference on Computer and Communications Security*, 1998.

[22] P. Goldsack, J. Guijarro, A. Lain, G. Mecheneau, P. Murray, and P. Toft. SmartFrog: Configuration and Automatic Ignition of Distributed Applications. In *HP OpenView University Association Workshop, Hewlett-Packard Labs*, 2003.

[23] K. Krauter, R. Buyya, and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software - Practice and Experience*, 32(2):135–164, 2002.

[24] M. Massie, B. Chun, and D. Culler. The Ganglia Distributed Monitoring System: Design, Implementation and Experience. *Parallel Computing*, 30(7):817–840, 2004.

[25] S. Matsuoka, S. Shimojo, M. Aoyagi, S. Sekiguchi, H. Usami, and K. Miura. Japanese Computational Grid Research Project: NAREGI. *Proceedings of the IEEE*, 93(3):522–533, 2005.

[26] D. Menasce and E. Casalicchio. A Framework for Resource Allocation in Grid Computing. In *Intl. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, 2004.

[27] C. Molina-Jimenez, J. Pruyne, and A. van Moorsel. The Role of Agreements in IT Management Software. In *Architecting Dependable Systems III, LNCS 3549*, pages 36–58, 2005.

[28] A. Othman, P. Dew, K. Djemame, and I. Gourlay. Adaptive Grid Resource Brokering. In *IEEE Intl. Conference on Cluster Computing*, 2003.

[29] A. Sahai, A. Durante, and V. Machiraju. Towards Automated SLA Management for Web Services. In *HP Labs Technical Report, HPL-2001-310R1*, 2002.

[30] A. Sahai, S. Graupner, V. Machiraju, and A. van Moorsel. Specifying and Monitoring Guarantees in Commercial Grids through SLA. In *HP Labs Technical Report, HPL-2002-324*, 2002.

[31] C. A. Santos, A. Sahai, X. Zhu, D. Beyer, V. Machiraju, and S. Singhal. Policy-Based Resource Assignment in Utility Computing Environments. In *HP Labs Technical Report, HPL-2004-142*, 2004.

[32] J. Schopf, M. D'Arcy, N. Miller, L. Pearlman, I. Foster, and C. Kesselman. Monitoring and Discovery in a Web Services Framework: Funtionality and Performance of the Globus Toolkit's MDS4. In *Submitted to SuperComputing Conference*, 2005.

[33] M. Siddiqui, A. Villazon, J. Hofer, and T. Fahringer. GLARE: A Grid Activity Registration, Deployment and Provisioning Framework. In *Intl. Conference on High Performance Computing and Communications*, 2005.

[34] W. Smith, I. Foster, and V. Taylor. Scheduling with Advanced Reservations. In *Intl. Parallel and Distributed Processing Symposium*, 2000.

[35] S. Vadhiyar and J. Dongarra. Self Adaptivity in Grid Computing. *Concurrency and Computation: Practice and Experience*, 17(2):235–257, 2005.

[36] B. Zhao, A. Joseph, and J. Kubiatowicz. Locality-aware Mechanisms for Large-scale Networks. In *Workshop on Future Directions in Distributed Computing*, 2002.

[37] S. Zhuk, A. Chernykh, A. Avetisyan, S. Gaissaryan, N. Kuzjurin, and A. Pospelov. Comparison of Scheduling Heuristics for Grid Resource Broker. In *3rd Intl. Conference on Parallel Computing Systems*, 2004.