Chapter 13
Facilitating e-Science Discovery Using Scientific Workflows on the Grid

Jianwu Wang, Prakashan Korambath, Seonah Kim, Scott Johnson, Kejian Jin, Daniel Crawl, Ilkay Altintas, Shava Smallen, Bill Labate, and Kendall N. Houk

Abstract e-Science has been greatly enhanced from the developing capability and usability of cyberinfrastructure. This chapter explains how scientific workflow systems can facilitate e-Science discovery in Grid environments by providing features including scientific process automation, resource consolidation, parallelism, provenance tracking, fault tolerance, and workflow reuse. We first overview the core services to support e-Science discovery. To demonstrate how these services can be seamlessly assembled, an open source scientific workflow system, called *Kepler*, is integrated into the *University of California Grid*. This architecture is being applied to a computational enzyme design process, which is a formidable and collaborative problem in computational chemistry that challenges our knowledge of protein chemistry. Our implementation and experiments validate how the Kepler workflow system can make the scientific computation process automated, pipelined, efficient, extensible, stable, and easy-to-use.

13.1 Introduction

"e-Science is about global collaboration in key areas of science and the next generation of infrastructure that will enable it."[1] Grid computing "coordinates resources that are not subject to centralized control by using standard, open, general-purpose

[1] John Taylor, Director General of Research Councils, Office of Science and Technology, UK.

J. Wang (✉), D. Crawl, I. Altintas, and S. Smallen
San Diego Supercomputer Center, UCSD, 9500 Gilman Drive, MC 0505, La Jolla, CA 92093, USA
e-mail: jianwu@sdsc.edu

P. Korambath, K. Jin, and B. Labate
Institute for Digital Research and Education, UCLA, 5308 Math Sciences, Los Angeles, CA 90095, USA

S. Kim, S. Johnson, and K.N. Houk
Department of Chemistry and Biochemistry, UCLA, Los Angeles, CA 90095, USA

protocols and interfaces, and deliver nontrivial qualities of service" [1]. For over a decade, Grid techniques have been successfully used to enable or facilitate domain scientists on their scientific computational problems by providing federated resources and services. Yet the software that creates and manages Grid environments, such as the Globus toolkit,[2] gLite,[3] and Unicore,[4] alone is not sufficient to manage the complex job control and data dependencies for many domain-specific problems. Such problems require combining more than one complex computational code into flexible and reusable computational scientific processes [2–4]. Scientific workflow systems [5, 6] enable researchers to design computational experiments that span multiple distributed computational and analytical models, and in the process, store, access, transfer, and query information. This requires the integration of a variety of computational tools, including the domain-specific software, database programs as well as preparation, visualization, and analysis toolkits [2, 3].

In this chapter, we explain how scientific workflow systems can facilitate the e-Science discovery in Grid environments by providing features such as scientific process automation, resource consolidation, parallelism, provenance tracking, fault tolerance, and workflow reuse. The chapter is organized as follows. In Sect. 13.2, we summarize the core services needed for e-Science discovery. Section 13.3 demonstrates an assembly of these services by integrating the Kepler workflow system[5] into the University of California Grid (UC Grid).[6] In Sect. 13.4, an application for a theoretical enzyme design computation process is explained using the integrated architecture. We concluded the chapter in Sect. 13.5.

13.2 The Core Services to Support e-Science Discovery

Nowadays, various services have been provided by the infrastructure to support e-Science discovery. e-Science problems build upon increasingly growing scientific data and require large-scale computational resources. Commonly, data and computation resources are distributed in geographically sparse locations and have a variety of usage modes. A typical computational experiment involves various tasks. By providing pipeline tools to connect the tasks and automate their execution, scientific process automation is becoming increasingly important to help scientists easily and efficiently utilize the data and computation resources to solve their domain-specific scientific problems. The infrastructure for e-Science should also enable scientists to interact with during the whole experiment lifecycle, such

[2] http://www.globus.org/toolkit/
[3] http://glite.web.cern.ch/glite/
[4] http://www.unicore.eu/
[5] http://kepler-project.org
[6] http://www.ucgrid.org/

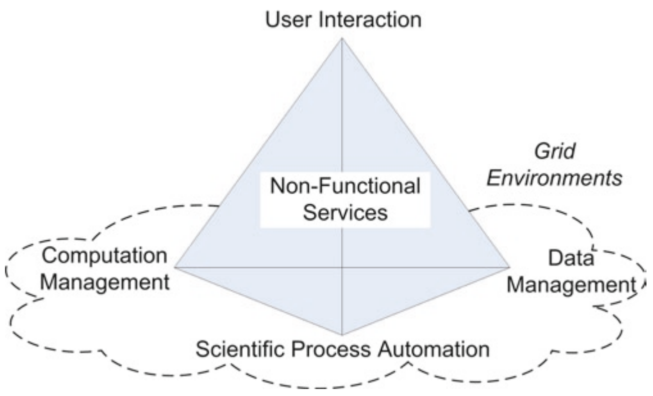


Fig. 13.1 A core service classification to support e-Science discovery

as triggering and adjusting the processes, monitoring their execution, and viewing the resulting data. Furthermore, nonfunctional services, such as security and failure recovery, are also important to ensure the whole process to be secure and fault tolerant. As shown in Fig. 13.1, computation management, data management, scientific process automation, user interaction, and nonfunctional services are the core service categories to support e-Science discovery. Also these services are complementary to each other, and are often integrated in many e-Science projects.

In this section, we describe the main services in each category, discussing their purposes, challenges along with the approaches and tools to enable them.

13.2.1 Computation Management

Over the past decade, Grid computation techniques have been successfully used to assist domain scientists with their scientific computational problems. Widely used Grid software includes the Globus toolkit, gLite, Unicore, Nimrod/G,[7] and Condor-G.[8] More details on Grid computation management can be found in [7, 8].

13.2.1.1 Service-Oriented Computation

A Web service is "a software system designed to support interoperable machine-to-machine interaction over a network".[9] Original Web services, also called big Web

[7] http://messagelab.monash.edu.au/NimrodG

[8] http://www.cs.wisc.edu/condor/condorg/

[9] http://www.w3.org/TR/ws-gloss/#webservice

services, are based on standards including XML, Web Service Definition Language (WSDL), and Simple Object Access Protocol (SOAP). Another set of Web services, called RESTful Web services, are simple Web services implemented based on HTTP protocol and the principles of Representational State Transfer (REST) [9]. Heterogeneous applications can be virtualized and easily interoperate with each other at Web service level by following the standards and protocols. The Open Grid Services Architecture [10] defines uniform exposed service semantics for Grid components, called *Grid services*, such that Grid functionalities can be incorporated into a Web service framework.

Through the introduction of Web and Grid services, a large number of computational resources in different scientific domains, e.g., bioinformatics, are becoming easily usable, and in turn introducing new challenges including service semantics, discovery, composition, and orchestration [11].

13.2.1.2 Local Resource Management

A compute cluster resource is a collection of compute nodes connected through a private fast interconnect fabric, e.g., Infiniband, Myrinet, or even local area network (LAN), and operated by an organization such as a university. A *local resource manager* is an application that is aware of the resources in a cluster and provides an interface for users to access them. The job submission and execution on a cluster is usually managed through resource manager software, such as the Torque,[10] Sun Grid Engine (SGE, renamed as Oracle Grid Engine recently),[11] or Load Sharing Facility (LSF).[12] A *resource scheduler,* on the other hand, can simply allocate jobs in a first in first out (FIFO) basis, or follow some complex scheduling algorithms (e.g., preemption, backfilling, etc.) such as the Maui,[13] Moab,[14] or SGE scheduler. A resource manager can use multiple schedulers. For example, Torque can be integrated with either the Maui or Moab.

The cluster owner sets policies on resource consumption such as which groups have access to it, how many resources can be allocated, whether they can run parallel jobs or only serial jobs, etc. This information is fed into the resource manager and is used when the scheduler executes jobs. A scheduler constantly monitors the cluster status and recalculates job priorities according to changes in the cluster environment.

Detailed techniques on local resource management can be found in [12, 13].

[10] http://www.clusterresources.com/products/torque-resource-manager.php

[11] http://www.sun.com/software/sge/

[12] http://www.platform.com/workload-management

[13] http://www.clusterresources.com/products/maui-cluster-scheduler.php

[14] http://www.clusterresources.com/products/moab-cluster-suite.php

13.2.1.3 Resource Allocation

Resource allocation is the process of assigning resources associated with a Grid for a user application. It is a key service since there are usually many available local resources and many user applications to be executed within one *Virtual Organization* (VO), where many different groups share resources through a collaborative effort.

One challenge here, called *resource scheduling*, is how to get a resource allocation result that satisfies user requirements, since resources in the Grid are not exclusive and may meet competing user requirements. It is already proved that the complexity of a general scheduling problem is NP-Complete [14]. So many approximation and heuristic algorithms are proposed to achieve suboptimal scheduling on the Grid [15]. Scheduling objectives are usually classified into *application centric* and *resource centric* [15]. The former one targets the performance of each individual application, such as makespan, economic cost, and quality of service (QoS). The latter one targets the performance of the resource, such as resource utilization and economic profit. Although it is an active research area, the proposed solutions are not ready to be widely deployed in production environments and it is still common that users or communities provide their own simple resource allocation strategies using Grid interfaces to contact each local resource.

13.2.2 Data Management

Data management on the Grid, sometimes called Data Grid, can be seen as a specialization and extension of the Grid as an integrating infrastructure for distributed data management [16]. This includes data acquisition, storage, sharing, transfer, archiving, etc. Representative Data Grid software includes the Globus toolkit, OGSA-DAI,[15] Storage Resource Broker (SRB),[16] and its recent successor called integrated Rule Oriented Data System (iRODS).[17] More comprehensive scientific data management and Data Grid surveys can be found in [16–20].

13.2.2.1 Data Acquisition

In general, scientific data may be created either from computations such as scientific calculations and image processing, or through data collection instruments such as an astronomy telescope, earthquake-monitoring devices, meteorology sensors, etc.

[15] http://www.ogsadai.org.uk/

[16] http://www.sdsc.edu/srb/index.php

[17] https://www.irods.org/

Once the experimental data is collected, it needs to be stored and transferred to the location where computing models can be run using that data to interpret the relationship or predict future events. The data often need to be shared among many researchers.

Data acquisition in large-scale observing systems, such as the National Ecological Observatory Network (NEON),[18] is an emerging application area. These systems accommodate a broad spectrum of distributed sensors and continuously generate very large amount of data in real-time. Heterogeneous sensor integration [21] and data stream processing [22, 23] are two main challenges [24]. The details of these two problems are not in the scope of this chapter.

13.2.2.2 Data Storage

Reliability, failover, and input/output (I/O) throughput are critical factors for large datasets storage. Typical solutions include storing the data through RAID[19] to achieve storage reliability by providing redundancy, and employing distributed parallel file systems using metadata tables, such as Lustre[20] and Parallel Virtual File System (PVFS)[21] to get higher I/O throughput.

One challenge on the Grid is how to provide a logical and simple view for researchers to access various types of geographically distributed data storage across a Grid environment. This is commonly handled by data storage abstraction techniques. For example, a logical data identifier, rather than its physical location, is provided to users to realize uniformed and easy data access. One tool that provides data abstractions is the SRB, which is a client-server middleware that provides a uniform interface for connecting to heterogeneous federated data resources over a network. The Replica Location Service (RLS)[22] in the Globus toolkit also support data abstraction. Additionally, both SRB and RLS support data replica functionality to manage the multiple copies of the same data, which will get better response time for user applications by accessing data from locally "cached" data stores.

13.2.2.3 Data Transfer

A data transfer moves data between two physical locations. It is necessary to share data within the VO or realize better computation balance and performance. Challenges here include performance, security, and fault tolerance.

[18] http://www.neoninc.org/

[19] http://en.wikipedia.org/wiki/RAID

[20] http://www.lustre.org/

[21] http://www.pvfs.org/

[22] http://www.globus.org/toolkit/data/rls/

There are many data transfer tools, e.g., FTP (File Transfer Protocol), scp (secure copy), GridFTP, SRB, and others. FTP [25] is one of the universally available file transfer application, which functions over a network using TCP/IP-based communication protocol such as the current Internet. scp [26] is a simple shell command that allows users to copy files between systems quickly and securely. Using the above two tools does not need the expertise in Grid systems. GridFTP is built on top of FTP for usage in Grid computing with the data encryption through the Globus Grid Security Infrastructure (GSI).[23] Additionally, GridFTP can provide third party transfer, parallel streams, and fault tolerance. The SRB also provides strong security mechanisms supported by fine-grained access controls on data, and parallel data transfer operations.

13.2.2.4 Metadata

Metadata is usually defined as "data about data," which is regarded as "structured data about an object that supports functions associated with the designated object" [27]. Metadata is useful to understand, access, and query the designated object. The metadata structures vary for different targets and usages [28]. Commonly used metadata categories in e-Science projects consist of dataset metadata (size, creator, format, access method, etc.), application metadata (applicable operation system information, license, etc.), resource metadata (node number, CPU speed, memory size, disk capacity, etc.), and workflow metadata (creator, language, etc.).

Semantics and ontology are more sophisticate techniques to describe and process metadata [29]. A domain ontology represents the particular meanings of terms as they apply to that domain. For example, the myGrid ontology helps the service discovery and composition in bioinformatics domain [30].

13.2.3 Scientific Process Automation

Scientific workflows are a common solution to realize scientific process automation [31, 32]. Workflow is a higher-level "language" in comparison to classic programming languages, such as scripting and object-oriented languages. The advantages of using workflow languages for scientific process include: (1) Many workflow systems support intuitive process construction by "dragging and dropping" via a graphical user interface (GUI). (2) The components or sub-workflows in workflow are easy to share and reuse. (3) Many workflow languages support task parallelism intuitively (see Sect. 13.2.3.2). (4) Workflow systems usually have built-in provenance support (see Sect. 13.2.3.4). (5) Some workflow systems are able to dynamically optimize process execution in Grid or other distributed execution environments.

[23] http://www.globus.org/security/overview.html

Widely used scientific workflow systems include Kepler, Pegasus,[24] Taverna,[25] Triana,[26] ASKALON,[27] and Swift.[28] More detailed scientific workflow surveys can be found in [5, 6, 31, 32].

13.2.3.1 Workflow Model

Although there are different languages for representing scientific workflows [33–35], workflows commonly include three types of components: tasks, control dependencies, and data dependencies [36]. For example, in Fig. 13.2, the tasks T2 and T3 will be executed under different conditions. Additionally, T4 needs to get data from either T2 or T3 to start its execution.

The tasks in the scientific computation process need to follow certain dependency logic to be executed. Usually the dependency logic can be described using *control flow*, *data flow*, or a *hybrid* of both. In control flows, or control-driven workflows, explicit control structures (including sequence, loop, condition, and parallel) describe the dependency. In data flows, or data-driven workflows, data dependencies describe the relationships among tasks. Two tasks are only linked if the downstream task needs to consume data from the output of the upstream task. The hybrid method uses both control and data dependencies to enable powerful and easy logic description. Many scientific workflow systems, e.g., Kepler, Triana and Taverna, use hybrid methods.

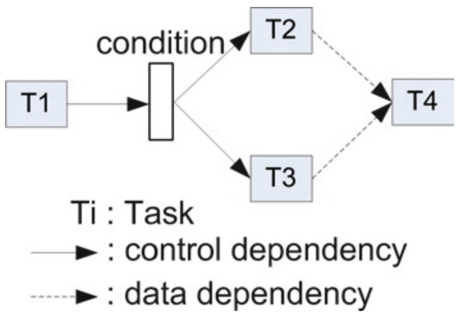


Fig. 13.2 An example workflow composed of tasks and dependencies

[24] http://pegasus.isi.edu/

[25] http://www.taverna.org.uk

[26] http://www.trianacode.org

[27] http://www.dps.uibk.ac.at/projects/askalon/

[28] http://www.ci.uchicago.edu/swift/

13.2.3.2 Task Parallelism

Task parallelism occurs when the tasks in one workflow can execute in parallel, providing a good execution performance. The task parallelism patterns can be classified into three basic categories: *simple parallelism*, *data parallelism*, and *pipeline parallelism* [37]. Simple parallelism happens when the tasks do not have data dependency in a data-driven workflow, or are in the same parallel control structure in a control-driven workflow. Data parallelism describes parallel execution of multiple tasks while different tasks processing independently on different parts of the same dataset. This employs the same principle as single instruction multiple data (SIMD) parallelism [38] in computer architecture. Pipeline parallelism describes a set of data that are processed simultaneously among a group of sequential tasks, each task processing one or more data elements of the set.

The difference between data parallelism and pipeline parallelism is illustrated in Fig. 13.3. In Fig. 13.3a, multiple instances of *Task 1* can be executing in parallel, each task consuming one-third of input data set, meanwhile other tasks have to wait for their input data. In Fig. 13.3b, *Task 1* can only consume one portion of all the input data for each execution. After *Task 1* finishes its processing on the initiate input data, the output data of *Task 1* will trigger the execution of its downstream tasks and meanwhile *Task 1 w*ill continue to process its next input data. So all the tasks in the Fig. 13.3 can be executing simultaneously, each processing its own data.

13.2.3.3 Workflow Scheduling

Workflow scheduling maps the tasks of a process and its data to proper computational resources, in order to meet expected performance, such as minimal execution

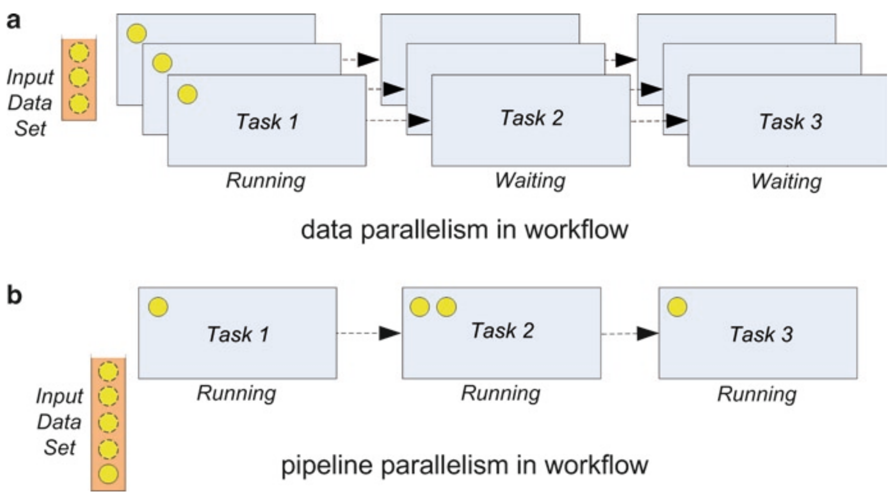


Fig. 13.3 Data and pipeline parallelism in workflow

time. After scheduling, tasks in one workflow might be executed on many local resources in parallel on the Grid.

Workflow scheduling belongs to the application-centric scheduling category of resource allocation (see Sect. 13.2.1.3), and focuses on the scheduling of process-based applications. Scheduling can be done either statically or dynamically based on whether it is flexible to diverse resource environments. Many heuristic scheduling approaches, such as genetic algorithm, are used to achieve suboptimal solutions [39, 40]. The local computation resource capacity, its advanced reservation capability, and real-time status are usually needed to make decisions during the scheduling.

13.2.3.4 Provenance Management

Provenance plays a critical role in scientific workflows, since it allows or helps "determine the derivation history of a data product, starting from its original sources" [41]. Comprehensive surveys on provenance management can be found in [41, 42].

A major challenge of provenance management in Grid environments is how to efficiently store provenance information and easily query it in the future. There are three typical approaches to collect the execution information for data provenance: *centralized*, *decentralized*, and *hybrid* [43]. Centralized approach stores all data generated during a workflow's distributed execution in one centralized center. However, storing the data content of all distributed nodes in a single, centralized center is inefficient, especially when the dataset size is large. In a decentralized approach, provenance data is stored on the local node when it is generated (i.e., no data needs to move to a centralized center). While it is efficient to separate data storage to each distributed node locally, it becomes difficult to query and integrate this data in the future. In the hybrid approach, provenance data are stored locally in distributed nodes, and a centralized provenance catalog is employed to maintain the metadata and location information. After finding the needed data endpoint at the provenance catalog, users can get the data content from the corresponding nodes. In the hybrid approach, the burden for data transfer is reduced in comparison to the centralized provenance system, and it is easier than the decentralized approach to do future provenance tracking. Note that one risk in both decentralized and hybrid approaches is that users may not be able to access the distributed nodes after the workflow execution, which is true when a resource is only open to some users for a limited time.

13.2.4 User Interaction

13.2.4.1 User Authentication

Grid credentials are commonly used to identify users to Grid resources. All certificate signing authorities (called CAs) need to have a policy to authenticate users before they issue user credentials. A user requests a Grid certificate only once to be authorized to use Grid resources. In some organizations such as UC Grid, all users can be positively identified as members of an organization using a Security Assertion Markup Language (SAML)[29] assertion from a Shibboleth Identity Provider (IdP).[30] Typically, IdPs will provide sufficient information to issue a Grid credential such as first name, last name, unique identifier, e-mail address, etc.

Some CAs let users keep credentials in their custody whereas other organizations maintain them in a credential management server such as MyProxy.[31] In the latter case, the credentials never leave the signing machine and only the short-lived credentials (called delegated proxy credentials) are provided to users. Users can always check out credentials from those servers. The delegated proxy credentials usually have a short lifetime of less than 8 h and will be destroyed when they expire. Many CAs additionally have an annual renewal policy for their certificate, so when users are no longer associated with the original project, their certificates can be revoked. Their identities are published in a certificate revocation list, which is then distributed to all organizations where their certificates are trusted.

13.2.4.2 Portal

A Grid portal is a web server that provides web interfaces to Grid services such as job submission, file transfer, job status check, and resource information monitoring. Some Grid portals provide generic capabilities that can be used by many types of users whereas others, such as Science Gateways,[32] typically target a specific community of users. A Grid portal stores user information such as login identifier, the Grid resources the user is entitled to, the status of their submitted jobs, etc. When users login to a portal, they are redirected to a credential management server, which allows the portal to authenticate Grid services on behalf of the user through a delegated proxy credential. Since a Grid portal holds some information about the role of the user, it can also authorize some Grid services such as job submission.

One challenge here is to allow users to access multiple distributed resources and applications without separate authentications, which are usually supported by single sign-on techniques. GridShib[33] allows a portal to sign proxy certificates for the uniquely identifying subject assertion from a federated single sign-on service such as Shibboleth.[34] Shibboleth implements a federated identity standard based on SAML to provide identification and attributes of members in a federation. The primary purpose of Shibboleth is to avoid multiple passwords for multiple applications, interoperability within and across organizational boundaries, and enabling service providers to control access to their resources.

[29] http://saml.xml.org/

[30] http://shibboleth.internet2.edu/about.html

[31] http://grid.ncsa.illinois.edu/myproxy/

[32] https://www.teragrid.org/web/science-gateways/

[33] http://gridshib.globus.org/

[34] http://shibboleth.internet2.edu/

13.2.4.3 Job Monitoring

Users often like to know the overall load status and availability of resources. They also want to know the current execution status, e.g., which tasks are executing on which computers with which data. Therefore, job monitoring services are also important for user interaction.

Some general job status information can be retrieved when the job is submitted to a Grid resource. For example, the Scheduler Event Generator (SEG) service in the Globus toolkit gets the status of jobs such as pending, active, or done. The SEG service queries local cluster resource managers such as SGE or PBS, and the job status information can be retrieved through the command-line or optionally pulled into a Grid portal or other GUI interfaces to display to users using a subscription application programming interface (API).

To view overall job statistics, typically cluster monitoring systems such as Ganglia[35] or Nagios[36] are deployed and display queue information, node information, and the load on the cluster. A Grid monitoring service is also deployed to collect information from each resource's cluster monitoring tool, summarize it, and display it. This provides overall Grid job statistics that can be used by managers to ensure users' needs are being met. Such services are typically modeled after the Grid Monitoring Architecture [44], which was defined by Global Grid Forum. It includes three main parts: (1) a *producer*, a process that produces events and implements at least one producer API; (2) a *consumer*, a process that receives events and implements at least one consumer API; and (3) a *registry*, a lookup service that allows producers to publish the event types and consumers to search them. Examples of Grid monitoring tools include the Globus Monitoring and Discovery Service (MDS).[37]

13.2.4.4 Data Visualization

Data visualization is the creation of a visual representation of data, meaning "information which has been abstracted in some schematic form, including attributes or variables for the units of information" [45]. Through the presentation of visualized data, it is easy for scientists to study, analyze, and communicate with one another. The primary reference model is called the filter-map-render pipeline [46]. The filter stage includes data selection, extraction, and enrichment; the map stage applies a visualization algorithm to generate a viewable scene; and finally the render stage generates a series of images from the logical scene. To optimize the performance of data visualization, especially for large data set, many parallel visualization algorithms have been developed [47–49].

[35] http://ganglia.sourceforge.net/

[36] http://www.nagios.org/

[37] http://www.globus.org/toolkit/mds/

13.2.5 Nonfunctional Services

We categorize a service to be "*nonfunctional*" if it is not usually explicitly used by users, but useful to ensure a certain property of the e-Science discovery, such as security. These services are often transparently provided in the whole lifecycle of user usage. These services are also orthogonal to other services in that they can be integrated with each of them.

13.2.5.1 Security

Several security concerns in specific services have been discussed in Sects. 13.2.2.3 and 13.2.4.1. A key challenge of security management in Grid environments is that there is no centrally managed security system [50]. The GSI provides secure communication through authenticated services among resources and is widely used in many Grid systems and projects. It operates on the single sign-on concept; that is, a single user credential will be valid among multiple organizations that trust the certificate authority (CA) of that credential. In GSI, every user needs to have a pair of X509 certificates. One is the private key that is used to encrypt the data, and the other is public key that is used to decrypt the data once it reaches the target resource. This kind of process is called asymmetric encryption because the keys that encrypts and decrypts are not the same. In practice, all Grid communications including GridFTP use short-lived proxy certificates that allow the process to act on behalf of the user. The bearer of this proxy certificate has exactly the same capabilities as the original X-509 certificate holder until the proxy certificate expires. In a Grid infrastructure, these proxy certificates are often leased from a certificate management server, such as MyProxy. The advantage of having proxy certificates is that a Grid user can securely store the long-lived private key in a secure machine and release only short-lived proxy credentials on a publically accessible certificate management server during Grid computing.

13.2.5.2 Failure Recovery

Due to the large number of resources in a Grid, there is a high probability of a single resource component failure during an application execution. Further, providing fault tolerance in a Grid environment can be difficult since resources are often distributed and under different administrative domains. Failures may occur in every level in the Grid architecture [51], which includes: (1) computational platforms, e.g., selected nodes for a job may run out of memory or disk space while the job is still running; (2) network, e.g., a local resource is temporarily unreachable during job execution; (3) middleware, e.g., a Grid service fails to start the job; (4) application services, e.g., domain-specific application gets an execution exception due to unexpected input; (5) workflow system, e.g., infinite loops during workflow execution; or (6) user, e.g., a user credential expires during workflow execution.

Many of these issues fall into the domain of system and network administrators, who must design infrastructure to provide redundant components. Here, we address only parts at the workflow level, where *redundancy*, *retry*, and *migration* are the main fault tolerance policies. Using simultaneous execution on redundant resources, workflow execution will have a lower chance to fail. The workflow system can also retry a failed resource or service after a certain amount of time. In migration, a workflow system will restart on a different resource. For the latter two cases, checkpoint and intermediate data usually need to be recorded so that only a sub-workflow rather than the whole workflow will be re-executed.

13.3 Integrating the Kepler Workflow System in the University of California Grid

To demonstrate how the services in Sect. 13.2 can be seamlessly assembled, Kepler scientific workflow system is integrated into the UC Grid. The interoperation of these services and the characteristics of our integration will be discussed. We will also demonstrate how the capabilities of Kepler, including resource consolidation, task parallelism, provenance tracking, fault tolerance, and workflow reuse, can facilitate UC Grid on scientific process automation. Almost all the implementations in this architecture are open source or follow open standards.

13.3.1 University of California Grid

The UC Grid is a federation of all participating compute resources among UC campuses. Any organization that is willing to accept or follow the established policies and trust relations of UC the Grid policy management authority can also be an autonomous member of UC Grid. Typically, the UC Grid resources are open to all faculty, staff, and students who are members of any UC campus. The owners of the resources determine when and how much of those resources are allocated to the UC Grid resource pool.

As shown in Fig. 13.4, the UC Grid architecture mainly consists of four parts.

The UC Grid campus portal. This is the web interface, front-end to the UC Grid in a single campus. It provides the user interface and serves as a single point of access for users to all campus computing resources. It communicates with the Grid appliances in a single campus. Because the users are known only at the campus level, all applications for a Grid account starts at the campus portal. The campus portal takes the request from the users and authenticates them through a Shibboleth-based campus service, and if the authentication process is successful, it sends a request to the UC Grid portal to sign an X-509 certificate for the user.

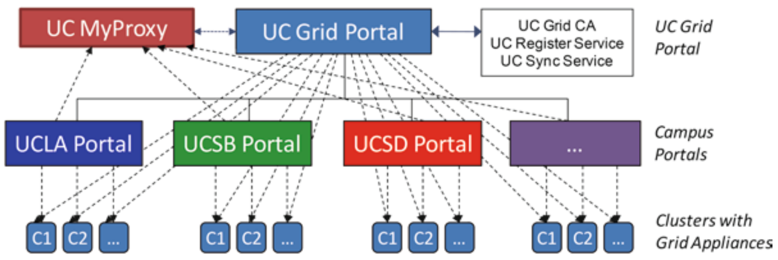


Fig. 13.4 The architecture of University of California Grid

The UC Grid portal. This is the web interface where users can login to access all the clusters in all ten campuses of the University of California. This is the super portal of all campus Grid portals and issues certificates for the entire UC Grid through the UC Grid CA. Once issued to a user, a certificate is pushed to the UC Grid MyProxy server that leases a corresponding short-lived credential every time the user logs in to the Grid portal. Any resource updates on campus portal will be updated on the UC Grid portal immediately through a Web service.

The UC MyProxy server. The primary purpose of this server is to store user credentials and release them to the UC Grid portal when users login to the portal.

Grid appliance nodes. These nodes are functionally equivalent to head nodes of a compute cluster where the UC Grid portal software is deployed. These nodes need to be open only to the Grid portals and compute nodes inside the cluster. Additionally, these nodes need to be a job submission host to the local scheduler such as Torque, LSF, or SGE.

The UC Grid is based on Globus toolkit 4.0.7, using its GridFTP, GRAM, MyProxy, and MDS services. We have also implemented several services to meet our own requirements. Two important ones are the UC Sync Service and the UC Register Service. The UC Sync Service makes sure all database on both the campus portal and UC wide portal are synchronized so that they have updated user, cluster, and application information. The UC Register Service is an automated process to authenticate, authorize, and issue credentials for new users on the UC Grid portal. For authentication purpose, it relies either on a Shibboleth-based service or, in the absence of such services, make use of a secure shell (SSH) based authentication mechanism. During the authorization process, a cluster administrator has to verify the Unix user identifier of new user on a cluster.

The UC Grid portal consists of application software that runs a portlet that is pluggable user-interface components in a portal to provide services such as Grid services. We employ Gridsphere[38] as our portlets container, which guarantees interoperability among portlets and portals through standard APIs.

[38] http://www.gridsphere.org

The usage modes are quite different for users who already have access to some compute clusters and users who do not have access to any clusters. So users are classified into different user categories in the UC Grid: *cluster user* and *pool user*. Cluster users are those users with a Unix user identifier on one of the clusters who can access their resources directly without the Grid portal web interface. Pool users on the other hand are those users who do not have an account on any of the clusters or on the cluster where they want to run an application. The jobs submitted by pool users are run through guest accounts on all participating clusters when unused cycles are available on that cluster. Pool users can access the resources only by authorizing through their Grid credentials. Currently, pool users can only submit precompiled application jobs that are deployed in advance by the cluster administrator, as the Grid portal does not have a mechanism to allow the pool users to upload their own binary files and guarantee the right run time architecture for that job. Some of the applications that pool users regularly use are Mathematica, Matlab, Q-Chem, NWChem, Amber, CPMD, etc. Typically, pool users need not worry about the target cluster as it is determined by the Grid portal depending on the dynamic resource availability, and also the application availability on any of the clusters.

13.3.2 Kepler Scientific Workflow System

The Kepler project aims to produce an open source scientific workflow system that allows scientists to easily design and efficiently execute scientific workflows. Inherited from Ptolemy II,[39] Kepler adopts the *actor-oriented modeling* [52] paradigm for scientific workflow design and execution. Each actor is designed to perform a specific independent task that can be implemented as *atomic* or *composite*. Composite actors, or sub-workflows, are composed of atomic actors bundled together to perform complex operations. Actors in a workflow can contain *ports* to consume or produce data, called *tokens*, and communicate with other actors in the workflow through communication channels via *links*.

Another unique property inherited from Ptolemy II is that the order of execution of actors in the workflow is specified by an independent entity called *director*. The director defines how actors are executed and how they communicate with each other. Since the director is decoupled from the workflow structure, a user can easily change the computational model by replacing the director using the Kepler graphical user interface. As a consequence, a workflow can execute sequentially, e.g., using the Synchronous Data Flow (SDF) director, or in parallel, e.g., using the Process Network (PN) director [53].

Kepler provides an intuitive graphical user interface and an execution engine to help scientists to edit and manage scientific workflows and their execution. In the Kepler GUI, actors are dragged and dropped onto the canvas, where they can be

[39] http://ptolemy.eecs.berkeley.edu/ptolemyII

customized, linked, and executed. Further, the Kepler execution engine can be separated from the user interface, thereby enabling the batch mode execution.

Currently, there are over 200 actors available in Kepler, which largely simplify the workflow composition. We will briefly describe the main distinctive actors that are used in this chapter.

Local execution actor. The *External Execution* actor in Kepler is the wrapper for executing commands that run legacy codes in a workflow. Its purpose is to call the diverse employed external programs or shell-script wrappers.

Job submission actors. Kepler provides two sets of actors that can submit jobs to two typical distributed resources: Cluster and Grid. Each set has actors to be used for different job operations, e.g., create, submit, and status check.

Data transfer actors. There are multiple sets of data transfer actors in Kepler to support moving data from one location to another by different ways, e.g., FTP, GridFTP, SRB, scp, and others.

Fault tolerance actor. An actor, called *contingency* actor, is provided in Kepler to support fault tolerance for workflow execution [54]. This actor is a composite actor that contains multiple sub-workflows, and supports automatic exception catching and handling by re-trying the default sub-workflow or executing the alternative sub-workflows.

Besides the above actors, Kepler also provides the following characteristic capabilities.

Inherent and implicit parallelism. Kepler supports inherent and implicit parallelism since it adopts a dataflow modeling approach [55]. Actors in Kepler are independent from each other, and will be triggered once their input data are available. Kepler workflow execution engine will parallelize actor execution automatically at runtime according to their input data availability. In addition, workflow composition in Kepler can be greatly simplified, since explicit parallel primitives, such as parallel-for, are not needed.

Pipeline parallelism. The execution of the tokens in the token set can be independent and parallel. Kepler supports pipeline parallelism by token streaming, blocking, and buffering techniques [37].

Provenance tracking. Kepler provenance framework [56] supports collection and query on workflow structure and execution information in both local and distributed environments [57].

13.3.3 Integrated Architecture

By integrating Kepler into UC Grid, an overall architecture is presented in Fig. 13.5. Most of the services described in Sect. 13.2 are supported here.

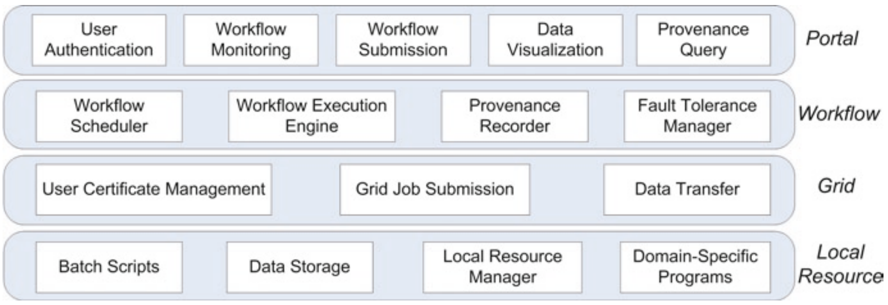


Fig. 13.5 A layered architecture to facilitate e-Science discovery

13.3.3.1 Portal Layer

The portal layer mainly provides the user interaction services described in Sect. 13.2.4 to enable users to interact with cyberinfrastructure and accomplish their e-Science discoveries.

User authentication. In order for the Grid portal to execute Grid services on behalf of users, it must be provided with a delegated credential from a credential management server (MyProxy server is used here). When users login to the portal, they enter their username and corresponding MyProxy password. The portal will retrieve and store the short-lived delegation credentials so that users can interact with Grid services. When the users log out, the delegated credentials are destroyed.

Workflow submission. Users select a workflow by browsing the workflow repository in the Grid portal. The Grid portal then presents a web interface for users to configure the workflow parameters and upload the required input files that need to be staged on the target computing resources. Users are also allowed to upload their own workflows to the portal, but require authorization to avoid intentional or unintentional security problems. For example, malicious code might be inserted into a workflow by executing file delete commands in the Kepler External Execution actor. Once approved, the workflow is uploaded to the workflow repository and made available to other users through the Grid portal.

Workflow monitoring. After a workflow is submitted through the Globus GRAM service for execution, the portal will monitor the progress and completion of the workflow execution by querying the Globus SEG service running on the target resource. Users can log in anytime to check the status, or the portal will send a notification e-mail when the execution is done.

Data visualization. Users can either download the output data to their local computer to visualize the data locally or choose one of the deployed application visualization services, e.g., Jmol,[40] to visualize the data through the portal itself.

[40] http://jmol.sourceforge.net/

Provenance query. During and after workflow execution, users can check provenance information via the query user interface in the portal. The query can utilize the provenance information from all previous workflow runs. For example, a user may want to understand how a parameter change influenced the results of one workflow, or how workflows are connected by a certain dataset.

13.3.3.2 Workflow Layer

The workflow layer provides the scientific process automation services described in Sect. 13.2.3.

Workflow scheduler. Currently, static workflow scheduling is supported by explicitly describing the scheduling policy in a separate workflow. Globus GRAM jobs are submitted through the workflow to initiate workflow task execution on resources. For workflow task execution on remote resources, data needs to be staged in before execution and staged out after execution. The capabilities of the resources must be known to achieve better overall load balancing. Sophisticated dynamic workflow scheduling capability is being studied as future work, such as optimal input data distribution on multiple resources based on the resources' capability and real-time load status.

Workflow execution engine. Once an invocation request is received from the workflow scheduler along with the corresponding workflow specifications, the workflow execution engine will parse the specification, and execute the actors based on their dependencies and director configuration.

Provenance recorder. During workflow execution, the provenance recorder listens to the execution messages and saves corresponding information generated by the workflow execution engine, such as an actor execution time and its input/output contents. It also saves a copy of the workflow for future verification and re-submission. The provenance will be stored locally and optionally merged into a centralized provenance database upon the workflow execution completion.

Fault tolerance manager. The fault tolerance manager will be triggered by exception messages generated from workflow execution engine, and will check whether the source of the exception is contained within a Contingency actor. If so, the alternative sub-workflows defined in the Contingency actor will be invoked with the corresponding configuration policy. Otherwise, the exception will be passed to the next higher-level Contingency actor, until the top level of the workflow is reached, where the workflow execution failure message will be reported and workflow execution will stop.

13.3.3.3 Grid Layer

The Grid layer consolidates multiple resources and manages their computation and data resources, providing unified services for the portal and workflow layer.

This layer is where the Globus toolkit software, such as GRAM, GridFTP, and MyProxy Server, is located.

User certificate management. There should be at least one CA, which stores the long-lived credentials for users. The short-lived credentials are then pushed into a MyProxy server. The portal gets the delegated credential when users login to the portal. A workflow can also get the delegated credential through a MyProxy actor.

Grid job submission. A GRAM service will enable job submission on any accessible resources. A workflow execution can invoke multiple GRAM services on different resources to realize parallel computation.

Data transfer. GridFTP permits direct transfer of files between the portal and the cluster resources or vice versa. Third party transfers can also be made between two local resources. A GridFTP actor is used to transfer data during workflow execution.

13.3.3.4 Local Resource Layer

The services provided by each compute cluster resource are located in the local resource layer. The compute nodes are not accessible directly from a portal. UC Grid communicates with the cluster through its Grid Appliance Node (see its details at Sect. 13.3.1) with both public and private interfaces.

Batch scripts. The batch job script syntax and terminologies vary according to the type of scheduler. The Globus GRAM service executes a job manager service at the target to create the job submission script using information such as executable name, number of processors, memory requirement, etc. It is also useful to create "shims" in the workflow [58], such as creating job scripts in accordance with the scheduler configuration of the host cluster.

Data storage. Each cluster provides data storage service for their users up to a certain size limitation. Cluster users can store their data permanently on the clusters they can access, whereas pool users must save their data in portal provided storage servers. The data generated on the cluster by pool users must be downloaded; otherwise, the data gets cleaned up periodically to make room for other pool users.

Local resource manager. A local resource manager is used to manage the job submission in a cluster. Several local resource managers are supported by the GRAM, such as SGE and PBS, which schedule jobs to the nodes in the cluster.

Domain-specific programs. Domain-specific programs are deployed to clusters to accomplish certain domain-specific computation problems. The applications widely used in UC Grid include Mathematica, Matlab, Q-Chem, NWChem, Amber, CPMD, etc.

13.4 Application in Computational Chemistry

In this section, we demonstrate how the services and integrated architecture described in Sects. 13.2 and 13.3 can facilitate e-Science discovery by applying them to a challenging application in computational chemistry. The detailed information about the application can be found at [59].

13.4.1 Theoretical Enzyme Design Process

Enzymes are nature's protein catalysts that accelerate and regulate all metabolic reactions. To design new catalysts computationally and then to make them with molecular biological techniques will be a breakthrough in technology. An inside-out design approach has been developed [60]. In the process, quantum mechanical calculations give a *theozyme* [61], or *theoretical enzyme*, which is theoretical optimum catalyst. Numerous protein scaffolds are then screened to determine which can be used to display the side chains to mimic the geometry of the theozyme; this procedure generates a large library of potential catalysts that are evaluated for fidelity to the theozyme. The best of these are subjected to mutations to obtain a design that will both fold and catalyze the reaction. Typically, a theozyme with active sites is matched at least once per scaffold (226 protein scaffolds so far) to potentially accommodate the orientation of the model functional groups. The computation process needs to be repeated for many times with different theozymes and calculation options. The goal of this application is to accelerate and facilitate this important computation- and data-intensive process for chemists.

13.4.2 Conceptual Enzyme Design Workflow

As shown in Fig. 13.6, the conceptual enzyme design workflow takes quantum mechanical theozymes as inputs, and goes through three discrete steps before validation by experiments. The goal of this workflow is to standardize the enzyme design process through automation, and eliminate the need for unnecessary human interaction. Sequences of tasks in the enzyme design process are repeated using the same series of programs, which must be executed to design and evaluate an enzyme

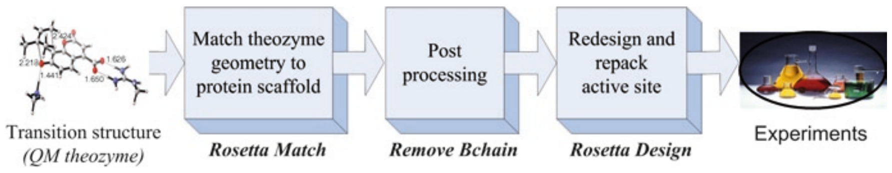


Fig. 13.6 Conceptual workflow of enzyme design process

for a new chemical reaction. For example, a theozyme with active sites is matched at least once per scaffold to potentially accommodate the orientation of the model functional groups.

Computational methodology for enzyme design has been developing using quantum mechanics and molecular dynamics. We locate structures of catalytic sites (theozyme) for the aromatic Claisen rearrangement of an allyl coumarin ether using density functional theory. The resultant theozymes are incorporated into existing stable protein scaffolds using the Rosetta programs of Zanghellini et al. [60]. The residues in the vicinity of active site are then optimized with RosettaDesign [62, 63]. The designs are further evaluated using molecular dynamics.

The entire enzyme design process can be performed independently for different scaffolds, and the total computation time for each scaffold can vary. The number of matches per theozyme is about 100–4,000, and computation time for a single scaffold on one CPU core is usually 1–3 h. The number of enzyme designs generated by RosettaDesign per match is about 100–15,000, and computational time on one CPU core is usually 0.5–2 h. One whole computation time required for all 226 scaffolds could take months on one single CPU core and the total number of generated enzyme designs is about seven million.

13.4.3 Enzyme Design Workflow in Kepler

The conceptual enzyme design workflow in Sect. 13.4.2 is composed in Kepler to make it executable. We will explain the details of the composition in this section.

13.4.3.1 Workflow for Execution on Local Cluster Resources

As shown in Fig. 13.7, a workflow is implemented to utilize the tens to hundreds of computing CPU cores in one cluster. The top-level workflow structure is the same as the conceptual workflow in Sect. 13.4.2. The connection links between the actors describe their dependencies, which determine that each input data, namely each scaffold, has to go through the three processing tasks sequentially. Once the output data of one actor is generated, it will trigger the execution of downstream actors. For instance, the RemoveBChain actor will start processing once it gets output data from the RosettaMatch actor.

Inside a composite actor, such as RosettaMatch, the sub-workflow dynamically creates job scripts according to a user's inputs and submits them to a job scheduler such as SGE on the cluster using Kepler job actors. By distributing the jobs for all scaffolds through the job scheduler on the cluster, the jobs can be concurrently executed on many nodes on the cluster and the concurrency capability is limited only by the node capacity of the cluster.

One computational characteristic in the enzyme design process is that the execution time for different scaffolds varies greatly, which could be minutes or hours.

Therefore, we adopt a set as the input of the workflow and elements of the input sets will be processed independently. With pipeline parallel support in Kepler, one scaffold does not need to wait for the completion of the other scaffolds before being processed by the downstream tasks.

13.4.3.2 Workflow for Execution on UC Grid

By adopting Globus as the Grid security and service infrastructure, the workflow shown in Fig. 13.8 is used to scheduling application execution among two cluster resources in UC Grid. For a local cluster, we execute the Kepler workflow shown in Fig. 13.7 through the Globus GRAM service deployed on the cluster.

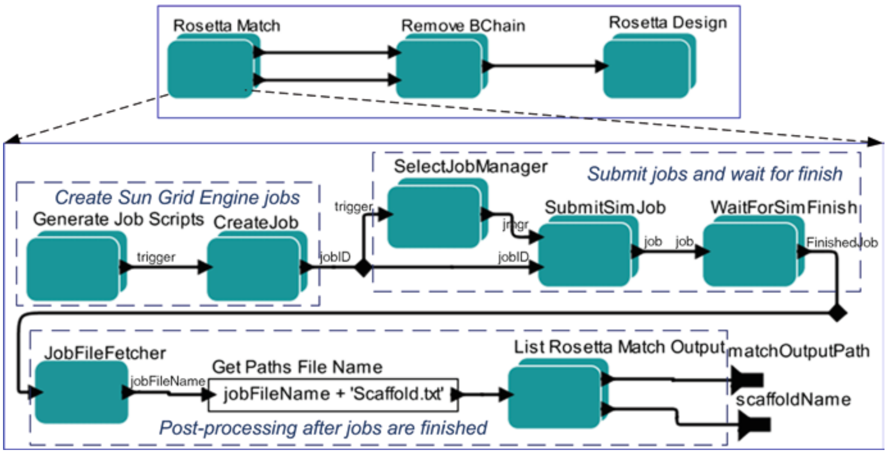


Fig. 13.7 Kepler workflow for enzyme design processes on one cluster

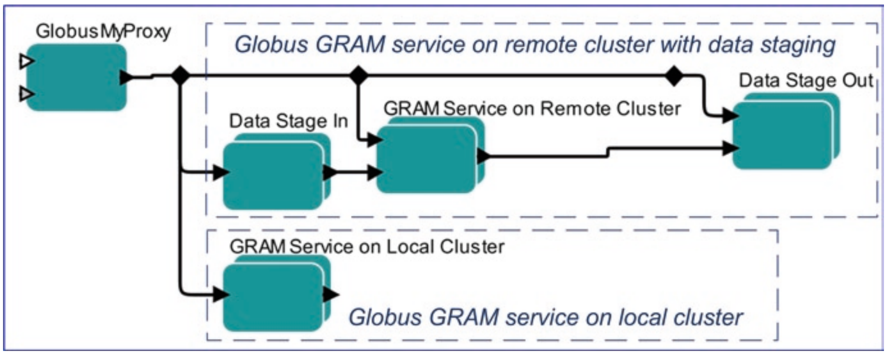

GlobusMyProxy
Globus GRAM service on remote cluster with data staging
Data Stage In
GRAM Service on Remote Cluster
Data Stage Out
GRAM Service on Local Cluster
Globus GRAM service on local cluster

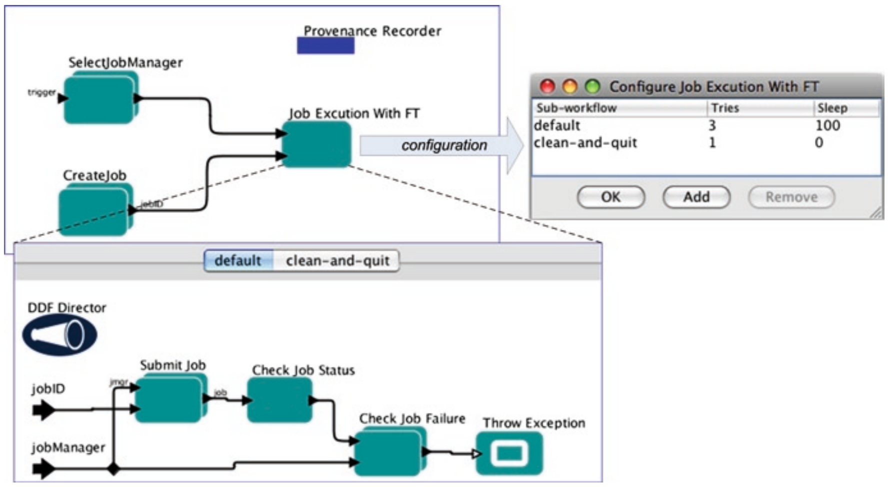
Fig. 13.8 Kepler workflow for enzyme design processes on UC Grid. The GRAM service actor will invoke the workflow shown in Fig. 13.7 on the target cluster

For the remote cluster, besides the workflow execution, two extra tasks need to be added: (1) Input data needs to be staged in to the remote clusters in advance of the workflow execution. (2) Output data needs to be staged out from remote cluster back to the local cluster. We employ the GridFTP to do the data stage in and out. The computations on the two clusters are independent of each other, so there is no control or data flow connections among the actors for these executions in the workflow, and the Kepler engine will run them in parallel. The workflows in Sect. 13.4.3.1 are easily reused in the two-level workflow structure. One challenge for this workflow is how to optimize the load balance on multiple clusters and the data stage in/out overhead.

13.4.3.3 Provenance and Fault Tolerance Support in the Workflow

While using more clusters increases computational capacity, it also increases the probability of failure. Although these exceptions happen only occasionally, the whole execution of the enzyme design workflow will crash without fault tolerance support.

To support fault tolerance at the workflow level, we adopt the Contingency actor for some sub-workflows. A simplified example is shown in Fig. 13.9, where the JobExecutionWithFT actor is a customized Contingency actor. There are two sub-workflows (shown in the bottom part of Fig. 13.9) in the actor, namely *default* and *clean-and-quit*, which will be invoked according to the configurations of the JobExecutionWithFT actor. The default sub-workflow submits a job, checks the job status, and throws an exception if the job fails. As specified in its configuration



Sub-workflow	Tries	Sleep
default	3	100
clean-and-quit	1	0

Fig. 13.9 Fault tolerance and provenance support in Kepler workflow

(shown in the right part of Fig. 13.9), after catching the exception, the JobExecutionWithFT actor will re-execute the default sub-workflow after sleeping for 100 s. If the default sub-workflow still gets exceptions after three retries, the clean-and-quit sub-workflow, which cleans intermediate data and stops the whole execution, will be executed with the same input tokens.

The exception handling logic can be easily expressed with the Contingency actor; no explicit loop and conditional switch is needed in the workflow. Further, the input data for sub-workflow retry does not need to be explicitly saved since they are automatically recorded by the provenance recorder and will be fetched for re-execution.

Besides fault tolerance, Kepler provenance also supports collection and query on workflow structure and execution information in both local and distributed environments. Each enzyme design workflow execution will generate millions of designs, and chemists may need the workflow to be executed many times with different input models. Kepler provenance can help chemists to track the data efficiently in the future, such as querying which input model was used to generate one particular design.

13.4.4 Experiment

To measure the speedup capabilities of the workflows, we chose two clusters in UC Grid as test bed where Globus toolkit 4.0.7 and Sun Grid Engine job scheduler are deployed. Cluster 1 has 8 nodes, each with 24 GB of memory and two 2.6 GHz quad-core CPUs. Cluster 2 has 30 nodes, each with 4 GB of memory and two 2.2 GHz single-core CPUs.

Our first experiment executed the enzyme design workflow described in Sect. 13.4.3.1 on cluster 2 with different usable CPU core configurations. We also executed another workflow that only had the RosettaMatch part of the enzyme design workflow in order to determine the speedup difference. We ran the workflows with different inputs. As shown in Table 13.1, all these tests, no matter the differences of workflow structures and inputs, have good scalability and speedup when the usable core number grows.

Table 13.1 Workflow execution on one local resource with different CPU cores

Workflow	Workflow input	Output data	Total job number	Workflow execution time (unit: hour)		
				1 core	25 core	60 core
RosettaMatch	10 scaffold	0.29 GB	10	3.38	0.69	0.63
RosettaMatch	226 scaffold	27.96 GB	226	128.61	5.52	3.06
RosettaMatch + RemoveBChain + RosettaDesign	10 scaffold	10.92 GB	296	533.61	29.32	11.24

Table 13.2 Workflow execution on UC Grid with different local resources

Workflow	Workflow input	Total job number	Workflow execution time (unit: hour) Cluster 1 (64 core)	Cluster 2 (60 core)	Cluster 1 and 2
RosettaMatch	10 scaffold	10	0.33	0.63	0.36
RosettaMatch	226 scaffold	226	1.52	3.06	1.33
RosettaMatch + RemoveBChain + RosettaDesign	10 scaffold	296	6.17	11.24	4.21

We also tested the workflows in Sects. 13.4.3.1 and 13.4.3.2 to know the concurrence performance on the Grid which is shown in Table 13.2. From the experiment data of the workflow execution only on cluster 1 and 2 (listed at the fourth and fifth column of Table 13.2), we know cluster 1 is about twice as fast. So approximately twice as many inputs are distributed to cluster 1 and cluster 1 is set as the local cluster when the workflows are executed on the two clusters, and its experiment results are listed at the sixth column. The experiment data demonstrates the good concurrence performance in the second and third tests. The poor performance in the first test is because there are too few jobs in comparison to the number of CPU cores. We can also see the speedup ratios are not as good as those in the first experiment. The reasons are twofold: (1) It is hard to realize good load balance on multiple clusters since the execution time for different scaffold varies. (2) The data stage in and out phases for remote cluster may cause a big overhead if the size of transferred data is very large.

13.5 Conclusions

Increasingly e-Science discoveries are being made based on the enhanced capability and usability of cyberinfrastructure. In this chapter, we have summarized the core services to support e-Science discovery in Grid environments. The five service dimensions, namely computation management, data management, scientific process automation, user interaction, and nonfunctional services, are all important constituents and complementary to each other. To demonstrate how these services can be seamlessly assembled, we explained an integration of the Kepler workflow system with the UC Grid, and its application in computational chemistry. The implementation and experiments validate the capability of this integrated architecture to make a scientific computation process automated, pipelined, efficient, extensible, stable, and easy-to-use. We believe that, as the complexity and size of scientific problems grow larger, it is increasingly critical to leverage workflow logic and task distribution across federated computing resources to solve e-Science problems efficiently.

Acknowledgments The authors would like to thank the rest of the Kepler and UC Grid community for their collaboration. We also like to explicitly acknowledge the contribution of Tajendra Vir Singh, Shao-Ching Huang, Sveta Mazurkova, and Paul Weakliem during the UC Grid architecture design phase. This work was supported by NSF SDCI Award OCI-0722079 for Kepler/CORE, NSF CEO:P Award No. DBI 0619060 for REAP, DOE SciDac Award No. DE-FC02-07ER25811 for SDM Center, and UCGRID Project. We also thank the support to the Houk group from NIH-NIGMS and DARPA.

References

1. Foster I (2002) What is the Grid? – a three point checklist. GRIDtoday, Vol. 1, No. 6. http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf
2. Sudholt W, Altintas I, Baldridge K (2006) Scientific workflow infrastructure for computational chemistry on the Grid. In: Proc. of the 1st Computational Chemistry and Its Applications Workshop at the 6th International Conference on Computational Science (ICCS 2006):69–76, LNCS 3993
3. Tiwari A, Sekhar AKT (2007) Workflow based framework for life science informatics. Computational Biology and Chemistry 31(5–6):305–319
4. Yang X, Bruin RP, Dove MT (2010) Developing an End-to-End Scientific Workflow: a Case Study of Using a Reliable, Lightweight, and Comprehensive Workflow Platform in e-Science. Computing in Science and Engineering, 12(3):52–61, May/June 2010, doi:10.1109/MCSE.2010.61
5. Taylor I, Deelman E, Gannon D, Shields M (eds) (2007), Workflows for e-Science. Springer, New York, Secaucus, NJ, USA, ISBN: 978-1-84628-519-6
6. Yu Y, Buyya R (2006) A Taxonomy of Workflow Management Systems for Grid Computing. J. Grid Computing, 2006 (3):171–200
7. Foster I, Kesselman C (eds) (2003) The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, The Elsevier Series in Grid Computing, ISBN 1558609334, 2nd edition
8. Berman F, Fox GC, Hey AJG (eds) (2003) Grid Computing: Making The Global Infrastructure a Reality. Wiley. ISBN 0-470-85319-0
9. Richardson L, Ruby S (2007) RESTful Web Services. O'Reilly Media, Inc., ISBN: 978-0-596-52926-0
10. Foster I, Kesselman C, Nick J, Tuecke S (2002) The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. www.globus.org/research/papers/ogsa.pdf
11. Singh MP, Huhns MN (2005) Service-Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons
12. Buyya R (ed.) (1999) High Performance Cluster Computing: Architectures and Systems. Volume 1, ISBN 0-13-013784-7, Prentice Hall, NJ, USA
13. Buyya R (ed.) (1999) High Performance Cluster Computing: Programming and Applications. Volume 2, ISBN 0-13-013785-5, Prentice Hall, NJ, USA
14. El-Rewini H, Lewis TG, Ali HH (1994) Task Scheduling in Parallel and Distributed Systems, ISBN: 0130992356, PTR Prentice Hall
15. Dong F, Akl SG (2006) Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical Report No. 2006-504, Queen's University, Canada, http://www.cs.queensu.ca/TechReports/Reports/2006-504.pdf
16. Chervenak A, Foster I, Kesselman C, Salisbury C, Tuecke S (2000) The data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Journal of Network and Computer Applications. 23(3): 187–200. July 2000, doi:10.1006/jnca.2000.0110

17. Gray J, Liu DT, Nieto-Santisteban M, Szalay A, DeWitt DJ, Heber G (2005) Scientific data management in the coming decade, ACM SIGMOD Record, 34(4):34–41, doi://10.1145/1107499.1107503
18. Shoshani A, Rotem D (eds) (2009) Scientific Data Management: Challenges, Existing Technology, and Deployment, Computational Science Series. Chapman & Hall/CRC
19. Moore RW, Jagatheesan A, Rajasekar A, Wan M, Schroeder W (2004) Data Grid Management Systems. In Proc. of the 21st IEEE/NASA Conference on Mass Storage Systems and Technologies (MSST)
20. Venugopal S, Buyya R, Ramamohanarao K (2006) A taxonomy of Data Grids for distributed data sharing, management, and processing. ACM Comput. Surv. 38(1)
21. Yick J, Mukherjee B, Ghosal D (2008) Wireless sensor network survey. Computer Networks, 52(12): 2292–2330, DOI: 10.1016/j.comnet.2008.04.002.
22. Fox G, Gadgil H, Pallickara S, Pierce M, Grossman RL, Gu Y, Hanley D, Hong X (2004) High Performance Data Streaming in Service Architecture. Technical Report. http://www.hpsearch.org/documents/HighPerfDataStreaming.pdf
23. Rajasekar A, Lu S, Moore R, Vernon F, Orcutt J, Lindquist K (2005) Accessing sensor data using meta data: a virtual object ring buffer framework. In: Proc. of the 2nd Workshop on Data Management for Sensor Networks (DMSN 2005): 35–42
24. Tilak S, Hubbard P, Miller M, Fountain T (2007) The Ring Buffer Network Bus (RBNB) Data Turbine Streaming Data Middleware for Environmental Observing Systems. eScience 2007: 125–133
25. J. Postel and J. Reynolds, File Transfer Protocol (FTP), Internet RFC-959 1985
26. secure copy, http://linux.die.net/man/1/scp
27. Greenberg J (2002) Metadata and the World Wide Web. The Encyclopedia of Library and Information Science, Vol.72: 224–261, Marcel Dekker, New York
28. Wittenburg P, Broeder D (2002) Metadata Overview and the Semantic Web. In Proc. of the International Workshop on Resources and Tools in Field Linguistics
29. Davies J, Fensel D, van Harmelen F. (eds.) (2002) Towards the Semantic Web: Ontology-driven Knowledge Management. Wiley
30. Wolstencroft K, Alper P, Hull D, Wroe C, Lord PW, Stevens RD, Goble C (2007) The myGrid Ontology: Bioinformatics Service Discovery. International Journal of Bioinformatics Research and Applications, 3(3):326–340
31. Ludäscher B, Altintas I, Bowers S, Cummings J, Critchlow T, Deelman E, Roure DD, Freire J, Goble C, Jones M, Klasky S, McPhillips T, Podhorszki N, Silva C, Taylor I, Vouk M (2009) Scientific Process Automation and Workflow Management. In Shoshani A, Rotem D (eds) Scientific Data Management: Challenges, Existing Technology, and Deployment, Computational Science Series. 476–508. Chapman & Hall/CRC
32. Deelman E, Gannon D, Shields MS, Taylor I (2009) Workflows and e-Science: An overview of workflow system features and capabilities. Future Generation Comp. Syst. 25(5): 528–540
33. Brooks C, Lee EA, Liu X, Neuendorffer S, Zhao Y, Zheng H (2007), Chapter 7: MoML, Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II), EECS Department, University of California, Berkeley, UCB/EECS-2007-7, http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-7.html
34. Scufl Language, Taverna 1.7.1 Manual, http://www.myGrid.org.uk/usermanual1.7/
35. SwiftScript Language Reference Manual. http://www.ci.uchicago.edu/swift/guides/historical/languagespec.php
36. Wang J, Altintas I, Berkley C, Gilbert L, Jones MB (2008) A High-Level Distributed Execution Framework for Scientific Workflows. In: Proc. of workshop SWBES08: Challenging Issues in Workflow Applications, 4th IEEE International Conference on e-Science (e-Science 2008):634–639
37. Pautasso C, Alonso G (2006) Parallel Computing Patterns for Grid Workflows, In: Proc. of Workshop on Workflows in Support of Large-Scale Science (WORKS06) http://www.iks.ethz.ch/publications/jop_grid_workflow_patterns

38. Flynn MJ (1972) Some Computer Organizations and Their Effectiveness. IEEE Trans. on Computers, C–21(9):948-960
39. Wieczorek M, Prodan R, Fahringer T (2005) Scheduling of scientific workflows in the ASKALON grid environment. SIGMOD Record 34(3): 56–62
40. Singh G, Kesselman C, Deelman E (2005) Optimizing Grid-Based Workflow Execution. J. Grid Comput. 3(3–4):201–219
41. Simmhan YL, Plale B, Gannon D (2005). A survey of data provenance in e-science. SIGMOD Record, 34(3):31–36
42. Davidson SB, Freire J (2008) Provenance and scientific workflows: challenges and opportunities. In: Proc. of SIGMOD Conference 2008:1345–1350
43. Wang J, Altintas I, Berkley C, Gilbert L, Jones MB (2008) A High-Level Distributed Execution Framework for Scientific Workflows. In: Proc. of the 2008 Fourth IEEE International Conference on e-Science (e-Science 2008):634–639
44. Tierney B, Aydt R, Gunter D, Smith W, Swany M, Taylor V, Wolski R (2002) A Grid Monitoring Architecture. GWDPerf-16–3, Global Grid Forum http://wwwdidc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf
45. Friendly M (2009) Milestones in the history of thematic cartography, statistical graphics, and data visualization. Toronto, York University, http://www.math.yorku.ca/SCS/Gallery/milestone/milestone.pdf
46. Haber RB, McNabb DA (1990) Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. IEEE Visualization in Scientific Computing:74–93
47. Singh JP, Gupta A, Levoy M (1994) Parallel Visualization Algorithms: Performance and Architectural Implications, Computer, 27(7):45–55 doi:10.1109/2.299410
48. Ahrens J, Brislawn K, Martin K, Geveci B, Law CC, Papka M (2001) Large-scale data visualization using parallel data streaming. IEEE Comput. Graph. Appl., 21(4):34–41
49. Strengert M, Magallón M, Weiskopf D, Guthe S, Ertl T (2004) Hierarchical visualization and compression of large volume datasets using GPU clusters. In: Proc. Eurographics symposium on parallel graphics and visualization (EGPGV04), Eurographics Association: 41–48
50. Welch V, Siebenlist F, Foster I, Bresnahan J, Czajkowski K, Gawor J, Kesselman C, Meder S, Pearlman L, Tuecke S (2003) Security for grid services. In: Proc. of the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12). IEEE Press
51. Plankensteiner K, Prodan R, Fahringer T, Kertesz A, Kacsuk PK (2007). Fault-tolerant behavior in state-of-the-art grid workflow management systems. Technical Report. CoreGRID, http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0091.pdf
52. Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee E, Tao J, Zhao Y (2005) Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience, 18 (10):1039–1065
53. Brooks C, Lee EA, Liu X, Neuendorffer S, Zhao Y, Zheng H (2007) Heterogeneous Concurrent Modeling and Design in Java (Volume 3: Ptolemy II Domains), EECS Department, University of California, Berkeley, UCB/EECS-2007-9, http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-9.html
54. Mouallem P, Crawl D, Altintas I, Vouk M, Yildiz U (2010). A Fault-Tolerance Architecture for Kepler-based Distributed Scientific Workflows. In: Proc. of 22nd International Conference on Scientific and Statistical Database Management (SSDBM 2010):452–460
55. Lee EA, Parks T (1995) Dataflow Process Networks. In: Proc. of the IEEE, 83(5):773–799
56. Altintas I, Barney O, Jaeger-Frank E (2006) Provenance Collection Support in the Kepler Scientific Workflow System. In: Proc. of International Provenance and Annotation Workshop (IPAW2006):118–132
57. Wang J, Altintas I, Hosseini PR, Barseghian D, Crawl D, Berkley C, Jones MB (2009) Accelerating Parameter Sweep Workflows by Utilizing Ad-hoc Network Computing Resources: an Ecological Example. In: Proc. of IEEE 2009 Third International Workshop on Scientific Workflows (SWF 2009) at Congress on Services (Services 2009):267–274

58. Radetzki U, Leser U, Schulze-Rauschenbach SC, Zimmermann J, Lussem J, Bode T, Cremers AB (2006) Adapters, shims, and glue-service interoperability for in silico experiments. Bioinformatics, 22(9):1137–1143
59. Wang J, Korambath P, Kim S, Johnson S, Jin K, Crawl D, Altintas I, Smallen S, Labate B, Houk KN (2010) Theoretical Enzyme Design Using the Kepler Scientific Workflows on the Grid, In: Proc. of 5th Workshop on Computational Chemistry and Its Applications (5th CCA) at International Conference on Computational Science (ICCS 2010):1169–1178
60. Zanghellini A, Jiang L, Wollacott AM, Cheng G, Meiler J, Althoff EA, Röthlisberger D, Baker D (2006) New algorithms and an in silico benchmark for computational enzyme design. Protein Sci. 15(12):2785–2794
61. Tantillo DJ, Chen J, Houk KN (1998) Theozymes and compuzymes: theoretical models for biological catalysis. Curr Opin Chem Biol. 2(6):743–50
62. Dantas G, Kuhlman B, Callender D, Wong M, Baker D (2003) A Large scale test of computational protein desing: Folding and stability of nine completely redesigned globular proteins. J. Mol. Biol. 332(2):449–460
63. Meiler J, Baker D (2006) ROSETTALIGAND: Protein-small molecule docking with full side-chain flexibility. Proteins 65:538–548