The QuakeSim Portal and Services: New Approaches to Science Gateway Development Techniques

Marlon E. Pierce[1], Xiaoming Gao[1,2], Sangmi L. Pallickara[1], Zhenhua Guo[1,2], Geoffrey C. Fox[1,2,3]
[1]Community Grids Laboratory
[2]Department of Computer Science
[3]Department of Informatics
Indiana University, Bloomington IN 47404

Abstract: Traditional techniques in building science portals and gateways are being challenged by new techniques such as Web 2.0 and Cloud Computing. This paper discusses some of our efforts to evaluate these techniques as we evolve the QuakeSim architecture. We believe that architecturally both traditional and newer approaches for Gateways are very similar, thus giving us a path for moving to hybrid approaches. In this paper, we specifically evaluate techniques for building interactive user interfaces that rely on remote services; architectural approaches for managing massive job submissions that can include both parallel and serial jobs; and an architectural prototype for building component based containers compatible with emerging standards.

1 Introduction: Gateways in Transition

Science gateways and portals have been developed by numerous groups to support scientific communities and provide them with Web accessible user interfaces to data and applications. In the United States, for example, the TeraGrid Science Gateway program [1] includes over 25 registered projects. Although often initially targeted at research communities, the greatest success for many of these gateways has been to support educational users [2].

The QuakeSim gateway [3] is designed to provide several capabilities to users, including

- Access to real-time and archival Global Positioning System (GPS) data;
- Services for analyzing GPS time series data streams using RDAHMM [4];
- Services for analyzing deformation properties of faults using Okada-based methods [5] and GeoFEST [6];
- Services accessing the QuakeTables fault model database [7];
- KML-generating services for rendering application inputs and outputs; and

- Web user interfaces for interacting with the above services.

Gateways are commonly built following the service-oriented architecture approach [8]. The key idea of this approach is to break the gateway's functionality into a set of well-defined Web services. The strength of this approach is its flexibility, allowing multiple user interfaces to be built and also allowing multiple backend services for data and application management to be integrated into the system.

User-interface components function as clients to these network services. This approach also enables a modular approach to user interface design, as client components can be relatively self-contained and aggregated using containers. Commonly these user interfaces are Web portals, but desktop applications and workflow composers [9][10][11][12] are also popular. The various approaches are not mutually exclusive if implemented correctly with services. Client components to these services can be built using a number of approaches. Portlets are a popular approach for Web portal-based gateway.

The service layer also acts as an abstraction and buffer layer over multiple backend resources. For example, application management services and user interfaces can be designed and built to run applications on test-bed resources for initial testing. Following testing, the service implementations can be changed to use high performance computing facilities. By keeping the service's invocation interface unchanged, the user interface components can remain unchanged.

Many of the founding implementation assumptions of science gateways discussed above are currently being challenged. Science gateways themselves are typically built out of "enterprise" standards (such as Java portlets), but the innovations in Web computing have been driven by insurgent technologies such as AJAX and social networking. As we have demonstrated previously, science gateways will benefit from the adoption of both Web 2.0 technologies and development approaches [13]. Gateways have also been built on a foundation of Grid computing middleware and high end computing at academic computing centers. Cloud computing is challenging this conception of middleware [14], and for-fee computing and storage services provide an alternative to computing grids such as the TeraGrid [15], Open Science Grid [16], and the Enabling Grids for E-Science (EGEE) project [17].

We believe however that the overall service architecture remains the same in any case, even if the components of the architecture diagram get relabeled or (more likely) become hybrids. Cloud services create virtual machines and clusters that themselves can host Web services. Portlets as a specification don't preclude AJAX-style interactivity, and while this particular component standard is probably growing obsolete, the general component-container model is not and is being replaced by gadget/widget containers such as iGoogle and Facebook. The Open Social Application Programming Interface

(API) [19] makes it possible for additional container developers to make applications that are compliant with massively used social networking sites such as MySpace, LinkedIn, and Orkut. It remains to be seen how the current non-interoperability between the Open Social and the Facebook APIs [20] will be resolved, and internationalization with popular non-English social network sites is also uncertain. API details aside, the general capabilities of these social networking sites should be useful for providing simple sharing mechanisms for scientific datasets and online capabilities among collaborators and with general audiences.

This paper reviews some of the advanced features and future developments for the QuakeSim portal as we adjust to changes from Web 2.0 and Cloud computing. In Section 2 we examine different strategies for increasing interactivity in map-based user interfaces to GPS station data. In Section 3, we describe the architecture for both single and massive job submission and management in Grid resources, with a path towards Cloud computing interoperability. In Section 4, we describe the architecture for an open source gadget container that we will use as the next generation of the QuakeSim portal. We conclude the paper in Section 5.

2 Components for Real-time and Archived GPS Data Analysis

This section describes updates and improvements to the QuakeSim GPS access and RDAHMM analysis portlets and services. In brief summary, we use the Geospatial Resources Web Service [22] developed by the NASA REASoN project [21] to access historical GPS data, which are then analyzed with the RDAHMM application via network services. This analysis is updated daily and is accessible via a web map interface. The map interface can be used to explore state changes in GPS networks. We have also developed a real-time GPS infrastructure in collaboration with the REASoN team that is also analyzed with RDAHMM. Analysis results are published every thirty minutes. Earlier versions of this work are described in more detail in [4][18].

New features described here include a state change summary service for the archival data and a movie generation service that can display the time evolution of the overall network state. All user interfaces are built with interactive maps, since we need to make these services as interactive as possible. We evaluate different strategies for improving interactivity and response time.

2.1 Daily RDAHMM – Plot of Number of Stations with State Changes

Extending our previous work on Daily RDAHMM Analysis (DRA) of the archived GPS data collected at 442 GPS stations in California, we have added a new function to the DRA Service for plotting the number of stations with state changes on each day from January 1, 1994 (the earliest GPS data records available) to the current date. The plot provides a comprehensive view of all

stations' states in the time interval and can help analyze the relationship between stations' state changes and geological events. For example, on 1999-10-16, when the Hector Mine Earthquake happened at Barstow, California, there were 22 stations changing their states around that area in our analysis.

The relationship between the plotting function and the DRA service and portlet is shown in Figure 1. After performing daily RDAHMM Analysis on all stations, the DRA service generates a file containing information about the number of stations with state changes on each day between 1994-01-01 and "today", and calls the plotting service to draw the plot, which is integrated and presented in the Daily RDAHMM portlet.

Figure 1 shows plot of the number of stations with state changes.

2.2 Daily RDAHMM – Station State Change Video Maker Service

In addition to the plot of state change numbers, we have added a station state change video maker service to the service architecture of DRA, as shown in Figure 2. After the daily RDAHMM analysis is performed on all stations, the video maker service is called by the DRA service to generate a video that shows a movie of all stations' dynamic state changes through the whole time between 1994-01-01 and the current date, which can then be downloaded from the DRA portlet. This video is updated every day, and Figure 3 shows a sample frame of the movie.

Figure 2 also gives the workflow of the video maker service:

1. Draw a series of pictures for a recent time range, such as 2 months, with one picture per day. Each picture shows the map of California, with

markers in different colors for all stations, indicating their state change information on the corresponding day;

2. Generate a video of this time range from all these pictures with encoder;
3. Merge this newly created video with a stable historical video to get a complete video for the whole history since 1994-01-01. E.g., merge the newly created video for the time between 2008-08-01 and 2008-09-06 with the video for 1994-01-01 to 2008-07-31. We generate the complete video in this way in consideration of performance: merging two videos is much faster than creating a new one from the pictures;
4. Periodically update the historical video to a more recent date. The archival data sets for the current data are not all immediately available from GRWS, so we use a two-week buffer. This eliminates false state changes caused by stations not yet reporting.

Figure 2 Workflow of Station State Change Video Maker Service

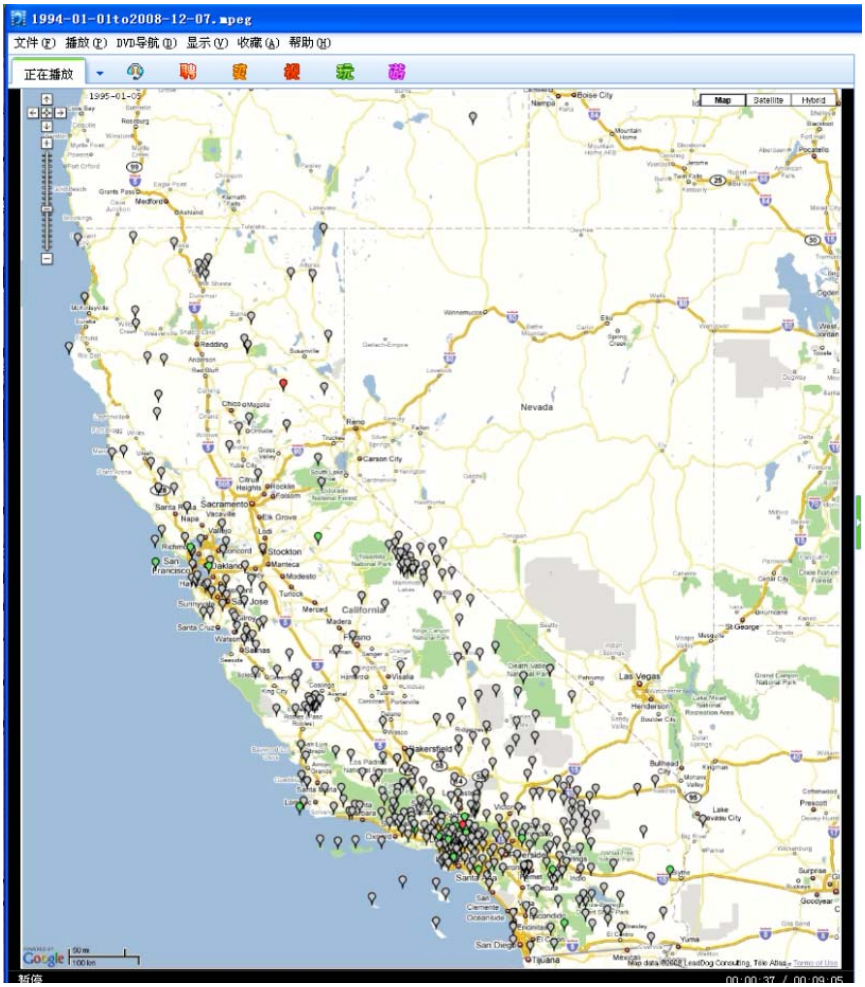


Figure 3 Screen capture of a generated movie of GPS stations' state changes between 1994-01-01 and the current date. Data is obtained from the REASoN project's Geophysical Resources Web Service. Push-pin icons represent GPS station locations. The color of the stations indicates the stations' state.

2.3 Daily RDAHMM – Results Analysis Service Performance

As described in [4], every day the RDA service goes through all the California GPS stations, obtains their latest available position data, performs an RDAHMM evaluation on each station, analyzes the evaluation results and stores their state change information and missing data information into a XML formatted result file. The DRA portlet can retrieve this file and present the evaluation results with interactive Google maps by analyzing the content of the file. An example is shown in Listing 1.

```
<xml>
    <output-pattern>
        <server-url>
            http://156-56-104-131.dhcp-bl.indiana.edu:8080//rdahmmexec
        </server-url>
        <pro-dir>daily_project_{!station-id!}_2007-12-18</pro-dir>
        <AFile>daily_project_{!station-id!}.A</AFile>
        <BFile>daily_project_{!station-id!}.B</BFile>
        <InputFile>daily_project_{!station-id!}_2007-12-18.input</InputFile>
        <LFile>daily_project_{!station-id!}.L</LFile>
        <XPngFile>daily_project_{!station-id!}_2007-12-18.all.input.X.png</XPngFile>
        <YPngFile>daily_project_{!station-id!}_2007-12-18.all.input.Y.png</YPngFile>
        <ZPngFile>daily_project_{!station-id!}_2007-12-18.all.input.Z.png</ZPngFile>
        <PiFile>daily_project_{!station-id!}.pi</PiFile>
        <QFile>daily_project_{!station-id!}_2007-12-18.Q</QFile>
        <MaxValFile>daily_project_{!station-id!}.maxval</MaxValFile>
        <MinValFile>daily_project_{!station-id!}.minval</MinValFile>
        <RangeFile>daily_project_{!station-id!}.range</RangeFile>
    </output-pattern>
    <station-count>442</station-count>
    <station>
        <id>7odm</id>
        <lat>30.3421</lat>
        <long>-144.3342</long>
        <status-changes>2006-10-03:5to2;2006-10-02:3to5;…</status-changes>
        <change-count>16</change-count>
        <time-nodata>2008-12-09to2008-11-23;…</time-nodata>
    </station>
    …
</xml>
```

Listing 1 Example summary file for results of Daily RDAHMM processing.

To serve the DRA portlet, we created a Daily RDAHMM Result Analysis Service to analyze the XML result file generated by the DRA service, and answer queries about stations' state change information from the portlet. This service provides three methods, shown in Table 1. In older versions, this task was handled by Java Server Faces "Managed Beans", which are a feature of the JSF framework. This updated web service has the following merits compared with the Managed Bean solution:

(1) **Efficiency:** since calling a web service method does not require refreshing the web page, which is necessary for calling a method of a Managed Bean, updating all stations' states information for a specific date is much faster when the web service method is used. Table 3

shows the tested page loading time and stations' states updating time following four different kinds of solution strategies that we have analyzed. Table 2 gives a detailed explanation of these solution strategies. Obviously, the web service strategy produces both shorter loading time and shorter updating time than the Managed Bean solution;

(2) **Scalability:** Since hidden HTML controls are needed to pass parameters for calling the methods of Managed Beans, one managed bean is created for each HTTP session (that is, each distinct user). This leads to redundant work of loading and analyzing the XML result file in each managed bean. When the number of clients is large, the web server will have to maintain too much state information about the clients. On the other hand, the web service is stateless; further, the XML file is only loaded once at the creation of the web service, and updated only once per day thereafter. We tested the response time of the Daily RDAHMM Result Analysis Service under different work-loads with JMeter [23] and the results are shown in Table 4. We can see that the service can handle 30*5=150 requests per second with an average response time of 379.5ms and a maximal response time of 1476 ms.

We did not use the pure JavaScript or JPS + JavaScript strategies because of their long loading time or large page size.

Table 1 Methods provided by the Daily RDAHMM Result Analysis Service. These provide data used in the map plots such the snapshot shown in Figure 2.

Method	Description
String getDataLatestDate()	Calculates the latest date when any station has some input data.
String getLatLongForStation (String stationId)	Returns the string representation of latitude and longitude of the station specified by stationId.
String calcStationColors (String date)	Calculates the string representation of the colors for all stations on a specific date. The colors represent different states of the stations on that date.

Table 2 Four Strategies for Implementing the DRA Portlet

Strategy	Description
Pure JavaScript	Analyze the XML result file and store data needed for map computation with client side JavaScript; stations' states and their markers' colors are computed with JavaScript at client side.

JSP + JavaScript	Analyze the XML result file with server side JSP codes, and JavaScript the data needed for map computation at client side with JavaScript objects; stations' states and their markers' colors are computed with JavaScript at client side. Since we do XML file analysis on the server side but do stations' states computation at client side, we have to generate a lot of JavaScript codes for directly storing the state change and missing data information, such as direct assignment sentences for each station, which result in a large html file transmitted to the client side.
Managed Bean + JavaScript	Analyze the XML result file and store the data needed for map computation with server side managed session beans; stations' states and their markers' colors are also computed by session beans.
Web Service + JavaScript	Analyze the XML result file and store the data needed for map computation with web service; stations' states and their markers' colors are also computed by web service.

Test environment configuration: the web server runs on a Linux server machine with 8 Intel Xeon 2.33GHz processors and 8G memory, and the Firefox client runs on a Linux desktop with a 2-core Intel Pentium 4 3.40GHz CPU and 2G memory. Each strategy is tested 5 times and the average results are given. Network connections are over 1 Gbps internal networks.

Table 3 Single Client test of performance of the four strategies

Method	Page size	Loading time	Map update time
Pure JavaScript (only state changes)	289KB	13.0s	4.4s
JSP + JavaScript	2.78MB	7.4s	2.8s
Managed Bean + JavaScript	766KB	8.4s	5.4s
Web Service + JavaScript	972KB	5.2s	3.4s

Test environment configuration: the test is done with JMeter 2.3.2, the server and client machine configuration is the same as Table 3. N is the number of client threads created per second. Each client sends 5 requests to the web service after started. RT means "Response Time".

Table 4 Web Service Scalability Test. N is the number of client threads interacting with the user interface each second. RT is the response time. Units are in milliseconds.

N	Avg RT(ms)	Min RT(ms)	Max RT (ms)
1	81.0	80.0	85.0
10	85.0	76.0	136.0
20	191.5	77.0	750.0
30	379.5	78.0	1476.0
40	596.5	80.0	2937.0
50	784.5	79.0	3485.0

2.4 Real-time RDAHMM – Two Phase RDAHMM Analysis Service

We have expanded the previous real-time RDAHMM analysis model to a two-phase analysis model, whose workflow is shown in Figure 4. In brief summary, data streams from the California Real Time Network are obtained and processed using a publish/subscribe-style filter chain system. A wrapped RDAHMM application represents a link in the processing chain. Raw data is generated at the rate of 1 Hz. Previous work provided a performance evaluation of the GPS stream filter management system, but to begin obtaining geophysical information, we need to train the RDAHMM filters first on an incoming data set. Classifications of states in the real time data are then made using the trained filters.

In the first phase, the service builds an RDAHMM model for each GPS station with its real-time data collected in 1-2 days. In the second phase, the service collects real-time data for each station, performs an RDAHMM evaluation for each station every 30 minutes based on their models, plots the evaluation results, and saves the state change information of each station within last 2 hours. Then the real-time RDAHMM portlet can access this information and present the plots of each station, and mark the stations on Google map with different colors to indicate their state change information. Adding a remodeling phase where the RDAHMM models are periodically rebuilt is part of our future work. The problem here is that building a model with 1-2 days' real-time data is computationally intensive. Strategies for solving these problems are described in the following section.

Figure 4 Workflow of Two-Phase Real-time RDAHMM Analysis Service

3 SWARM: Infrastructure for Scheduling Large-Scale Job Clusters

Many QuakeSim applications need substantial backend computing power. Applications such as GeoFEST need high-performance computing resources to realistically model faults, and real-time RDAHMM processing requires frequently updated training models that (although trivially parallelizable for each station) can be computationally demanding. These two scenarios are in fact very general and point out important differences between the current state of the art in Grid computing (which focuses on federating high end computing resources) and Cloud computing (which focuses on providing highly available virtual computer clusters). To address both of these issues, we have developed the Swarm service.

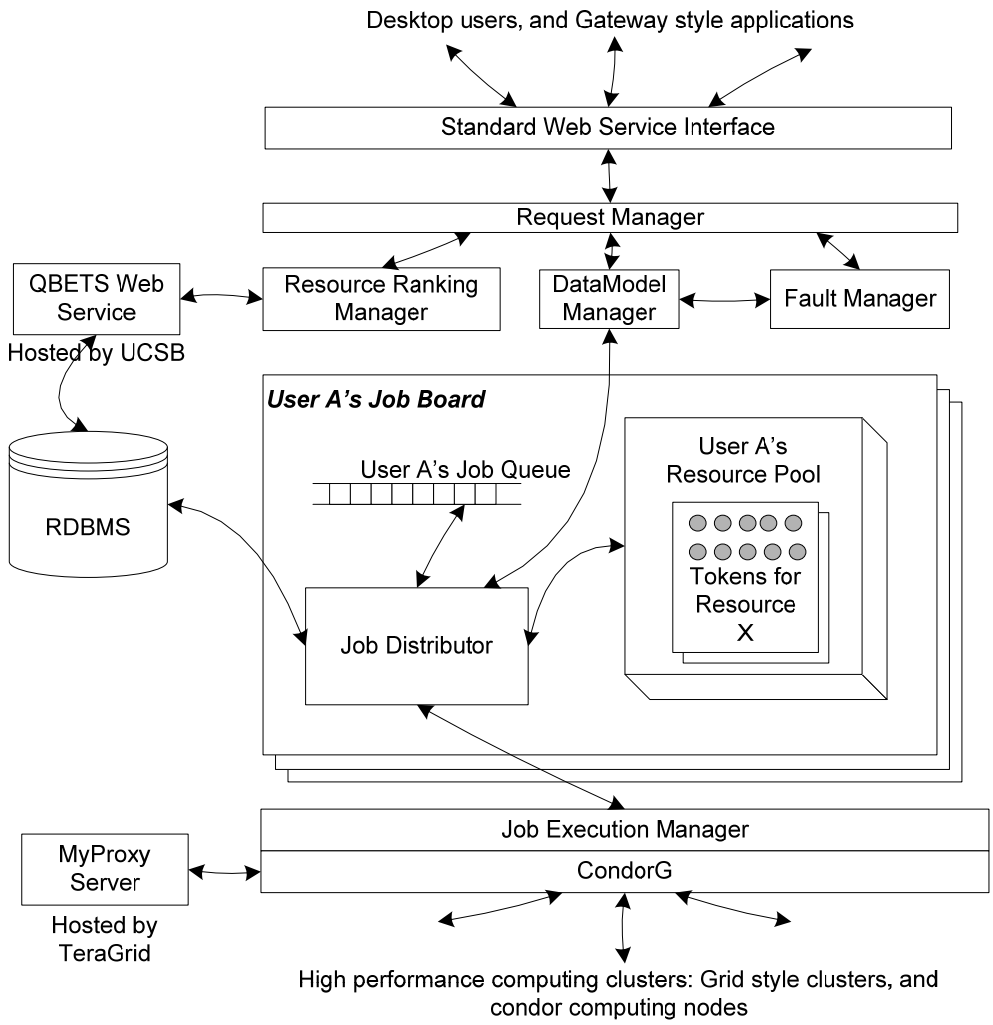
Swarm is a high-level job-scheduling infrastructure for large-scale jobs. Swarm has been developed for scientific applications that need to submit a massive number of high-throughput jobs to highly distributed computing clusters (include virtual clusters) and high performance computers. The jobs that may be submitted by Swarm include both serial and parallel jobs that are normally submitted to the batch job systems provided by high-performance computing clusters. The Swarm service is itself designed to be extensible, lightweight, and easily installable on a desktop or small server. Derivative services (such as services with a GeoFEST-specific API) based on Swarm can be integrated in a straightforward fashion with other applications including Web portals and science gateways.

There have been several approaches to high-level job scheduling in the grid

environment. We review this related work to put Swarm in context. GridWay [24], and PanDa [25] projects provide a job submission environment over multi-site resources. On top of the scheduling functionality, Swarm incorporates a resource prioritizing feature, which searches the batch queue system with the minimum wait time. Falkon [26] and myCluster [27] enable users to access provisioned resources (typically implemented by holding slots in a queuing system) to submit large-scale scientific jobs. Instead of provisioning resources, Swarm provides a user-based resource pool, which limits the maximum number submissions to a given batch queue system at any given time. Jobs are held on the Swarm server until more slots on the actual cluster queuing systems become available. This enables Swarm to incorporate policies from different batch queue systems more flexibly.

Pegasus [28] provides a workflow generation and mapping environment for grid computing environments. It generates a workflow plan based on artificial intelligence reasoning techniques. The workflow plan is transformed into a directed acyclic graph that is then passed to the Condor [29] DAGMan system and then executed on the grid environment. Swarm utilizes Globus [30] job submission services (running on the backend clusters) along with the Condor-G and Birdbath as the foundation job submission mechanism. Condor-G is an optional configuration of Condor that can act as a universal client to various types of Grid middleware. Birdbath is Condor's Web Service interface. It is also a configuration option for standard Condor and has a collection of Java client libraries that simplify the process of creating a client. Instead of using DAGMan, Swarm provides a simple built-in workflow manager that submits individual jobs through Condor-G and maintains the status of the submitted jobs.

We chose Condor-G as a universal client because of its flexibility. In addition to submitting jobs to various versions of the Globus toolkit, Condor also provides bridges to interact with PBS and LSF queuing systems directly. An "Amazon" Grid type has been added in Condor version 7.1, allowing Condor-G to also interact with Amazon's Elastic Computing Cloud services. Thus the Swarm service inherits these features.



.

Figure 5: Swarm architecture. Client applications interact with the Swarm WSDL using standard Web service tools. In practice, we extend Swarm to make problem-specific services that inherit Swarm capabilities but provide a code-specific WSDL.

Our goal with Swarm's architecture is to provide a highly extensible base for domain specific application extensions instead of a general purpose, "out of the box" job submission service. Science gateways or Web portals such as QuakeSim can integrate Swarm with their required data model and fault-handling scheme. The service is designed to be installable on single servers and can be hosted separately from the Grid middleware. In Figure 5, the Request Manager, Resource Ranking Manager, Data Model Manager, Fault Manager, Job Board, and Job Execution Manager are all components of a single service. We typically install Condor-G on the same host server. Globus and other middleware services such as provided by the TeraGrid are on separate hosts. These components are now discussed in detail.

3.1 Architecture

Swarm is a set of Web services and local servers, as depicted in Figure 5. Gateway style applications access Swarm via standard Web service interfaces. We have also provided simple, example command-line clients for desktop users. Each of the operations and parameters are defined in the WSDL associated with the services.

To provide the capability to track a large number of jobs, Swarm provides a simple structure for the submissions. Users submit jobs to Swarm in groups. Group sizes containing up to one million individual jobs have been tested. The requests for group job submissions are delivered to Swarm's *Request Manager*. The *Request Manager* creates a 128-bit universally unique identifier ticket for the series of jobs. An individual job is identified by its ticket and an internal ID. Here, the internal ID is the identity of the job, which is unique within the job group. This structure is designed to support multiple experiments launched by multiple users through the Web service.

As seen in Figure 5, the job submission process interacts with the *Resource Ranking Manager*, which prioritizes the resources over which the job is submitted to optimize the job execution process. With Swarm, users are allowed to specify multiple backend resources (computing clusters) to submit the job group. To prioritize the resources listed in the user's job description, Swarm interacts with the QBETS batch queue prediction service [31]. The QBETS service provides queue delay predictions. The Wall Clock Time and number of nodes are key factors to get the predicted delay. Wall Clock Time and the number of nodes are specified in the input job description and Resource Ranking Manager passes that information to the QBETS Web service and gets the result of predicted wait-time in the batch queue.

The *Data Model Manager* specifies the data model for the input, output and temporary files during the process. The temporary files include log files, error messages, and security related files such as proxy certificates. The data model provides a directory structure for the output files from a large number of jobs. In addition, the location for the privacy sensitive files can be specified in the data model. Besides using the standard data model, users or applications can implement their own data model to satisfy their application specific requirement.

The *Fault Manager* decides how to respond to the faults encountered during the job submission and execution. Swarm's current implementation categorizes faults into two categories: *fatal faults* and *recoverable faults*. A fatal fault is defined as a fault that cannot be recovered without new inputs from the users or relocating the jobs on different computing clusters. Examples of fatal faults include,

- Erroneous arguments, e.g. the supplied path to the input data is wrong;
- Hardware and software failures in the computing clusters; and

- Failures resulting from the policy of the computing cluster.

Recoverable faults are faults that can possibly be recovered without contacting the user. These are commonly related to resource specifications such as expected execution time or memory requirements. When Swarm detects that the fault is due to insufficient resource specifications, the jobs are resubmitted with modified arguments. Similar to the data model manager, the users or application developers can implement their own response mechanism.

Under the *Request Manager and Resource Ranking Manager,* there is a group of software components referred to as the *Job Board*. Swarm maintains a *Job Board* for each user. Each *Job Board* contains *a Job Queue, Job Distributor,* and *Resource Pool*. Users do not share any of these components. Matchmaking between the jobs and the resources are done in the user's *Job Board.*

When the Job Distributor finds a match with an available remote resource, the Job Execution Manager submits the job through Condor-G's Web service APIs. The user's Grid certificate (based on the X.509) is retrieved by means of interacting with the MyProxy service [32] and used to access to the Globus GRAM job manager. In addition, users are allowed to submit jobs to ordinary Condor computing nodes through Swarm.

3.2 Performance Evaluation

We have developed a prototype of the Swarm framework, in Java, based on Apache Axis2. The server was hosted on a machine with 3.40GHz Intel Pentium 4 CPUs and 1GB RAM. For the measurement, we ran the client software on the same machine to submit jobs. The machine involved in the benchmark was hosted on 1 Gbps network. As a group of HPC clusters, TeraGrid network was used for testing jobs.

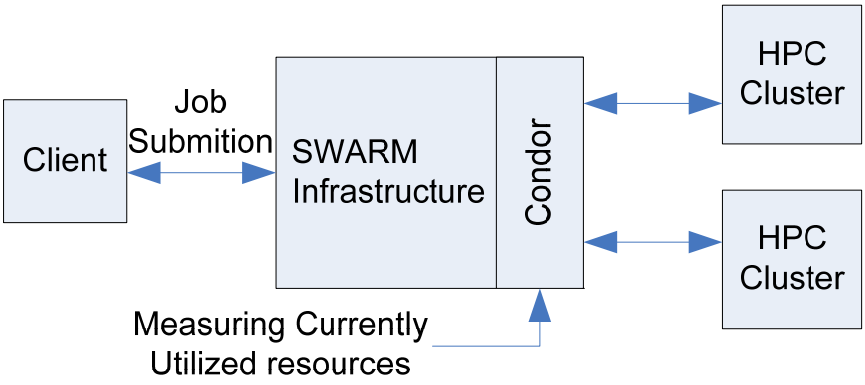


Figure 6 The testing scenario: The same group of jobs is submitted by client to Swarm and the Resource Utilization Rate is measured.

Our benchmark measured the resource utilization rate with various size of resource pool. Here the size of resource pool is defined as the total number of

jobs that we allow in the batch queue system concurrently (that is, we do not swamp the TeraGrid batch queues with jobs). Each of the clusters may have different policies about the number of jobs in the batch queue at a given time. Swarm controls the job submission so as not to exceed the number of concurrent jobs by means of configuring the size of resource pool for each of clusters.

The resource utilization rate shows how efficiently swarm utilizes the resources specified in the resource pool. We defined the resource utilization rate as

Resource Utilization Rate = (Total number of currently running jobs) / (Size of the resource pool)

As illustrated in Figure 6, the number of the concurrently running jobs is measured by executing condor command that returns status of jobs either running or idle. Figure 7 provides the resource utilization rate for the different size of resource pool. The time to reach close to maximum resource utilization rate (1.0 in Figure 7) is increased as the size of resource pool is increased. The job queue is scanned every minute, which is also configurable based on the characteristics of jobs to be submitted. The interval of the scanning caused the stairway pattern.

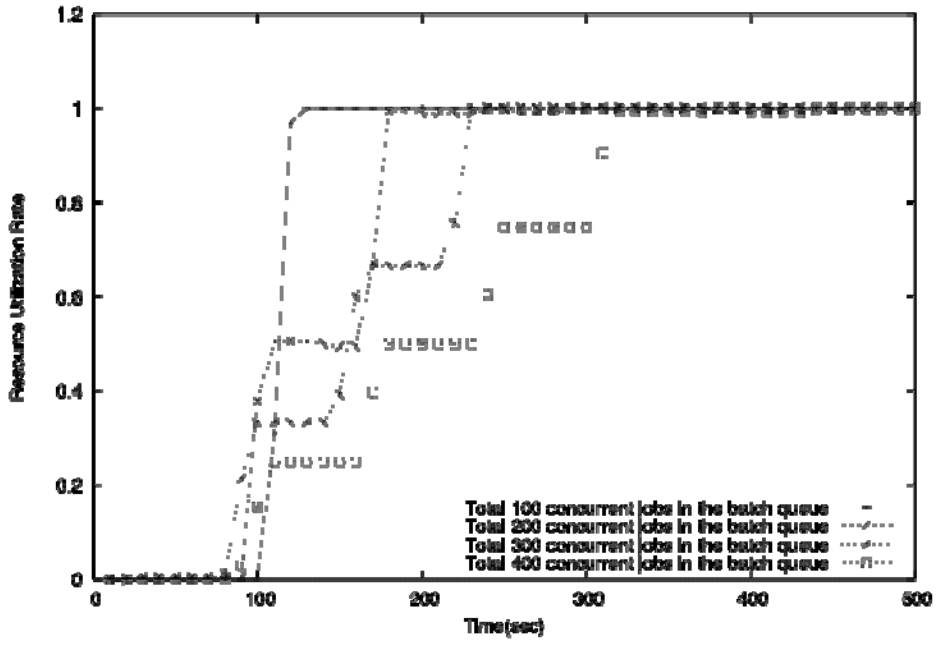


Figure 7 The resource utilization rate for the different size of resource pool

4 Developing a Gadget Container for Gateways

As discussed in the introduction, component-based portal systems have been a popular architecture for many years but are being challenged by open gadget APIs provided by iGoogle, Netvibes, and other popular “start pages”. These public containers simplify the process for developers to add new components, perhaps for very specialized groups of users. Integration may be very loose (essentially the third party content is delivered as an HTML IFrame), requiring no modification to the gadget’s source site except perhaps sizing. Integration between the component and the container may also be more tightly coupled through the use of JavaScript libraries supplied by the container provider.

Social networking portals have also been a major trend in Web 2.0 systems. Google has attempted to merge gadget-container systems and social networking in the Open Social consortium (which includes most prominent U.S.-based social networking sites except Facebook). Open Social extends Google’s gadget JavaScript API to enable the gadgets to interact with backend social network database information. Gadget developers can use this information to make asynchronously collaborative applications, such as shared calendars. Open Social compatible gadgets can run without modification in any Open Social compatible container.

Although one may want to use one of the prominent Open Social containers to build a science gateway such as QuakeSim, we also believe many of these gateways will want to provide their own containers. There are several reasons for this: the security model for the public containers may not be adequate for sharing scientific data; users may want to cleanly separate research work social networks from more casual networks of friends and family; the existing containers may need to be extended to support more container services (such as custom login and layout modules); and we may wish to experiment with advanced, unsupported capabilities such as real-time collaboration. By adopting the Open Social standard, we can develop gadget components that can be moved back and forth between (for example), iGoogle and a gadget-based QuakeSim. Shindig, and Apache incubator project, is nominally the Open Social container reference implementation, but its code base is very unstable, and it in any case is primarily designed to test the evolving API rather than serve as a production portal. Given the situation, we have decided to investigate and implement an open architecture version of a generic gadget container.

Our prototype gadget container is a system by which developers can download, build, and run their own layout containers. The current prototype consists of user authentication system, user administration system, and gadget management system. The user authentication system allows the system to accept new user registration and login of existing users. The user

administration system provides a way for administrators to manage all user accounts and profiles. The gadget management system allows users to manage gadgets in convenient way.

4.1 Architecture

As we have discussed above, portals are divided into containers and gadgets. Containers handle universal chores such as login, layout, and user management. Gadgets implement more specific functionality. In Open Social, there are actually *two* containers for each gadget: the display container (such as iGoogle) and the social container (such as Orkut, LinkedIn or MySpace). The former controls the display and layout of the gadgets, while the latter controls the social context that the gadget operates in (that is, it has access the gadget user's network and groups and can provide information about the network members' and groups' states). This architecture is depicted in Figure 8.

Although it is possible to run these two containers on the same server, this is not required. To simplify the interactions between the two separate containers, and to simplify account management generally, we include OpenID support [33]. OpenID allows portals and similar sites to establish trust in the authentication process of another portal. Thus a user can use Portal A to log into Portal B if he/she has an account on both and binds the relationship. OpenID is also useful for transmitting user profile information (such as the user's full name, email address, and contact information). We use OpenID's Simple Registration Extensions format [34] to do this.

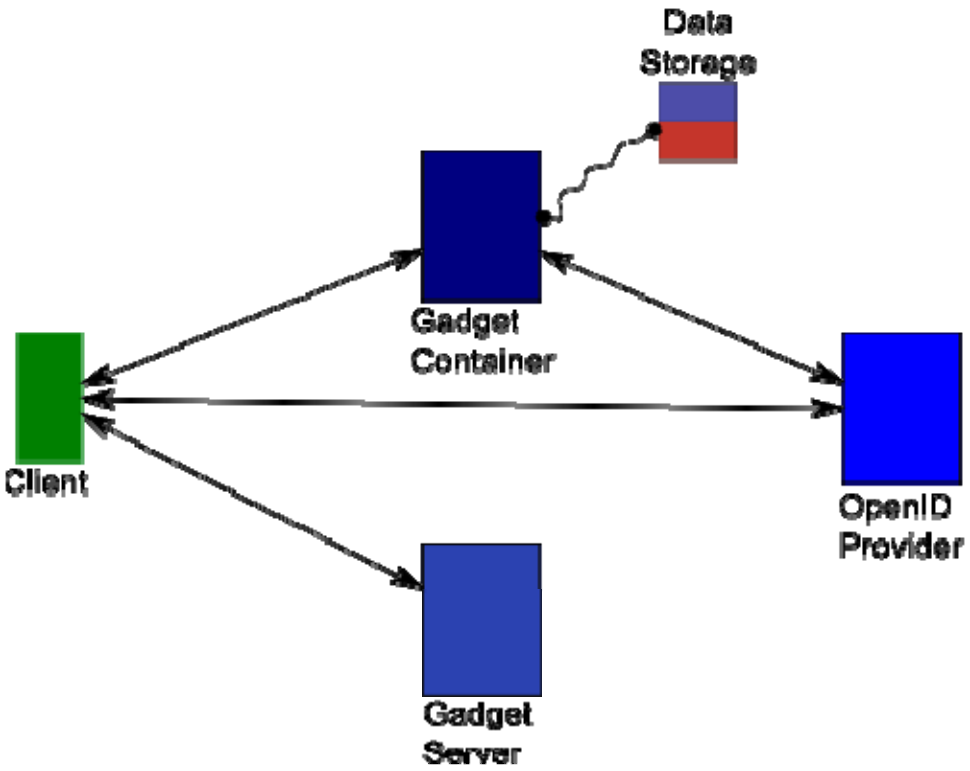


Figure 8 Gadget container architecture. Arrows represent HTTP-based communications. The curved line represents a JDBC connection.

When a new user wants to create a new account, he/she can do it in usual way by providing username, password and additional personal information. In addition, the user can bind the account to OpenID. Binding only needs to be done once and after that the user can log in to the portal using her/his associated OpenID. When a user logs in using OpenID, he/she first is directed to OpenID provider's side. If the user has not logged in, OpenID provider prompts the user for login. After that, the user is asked to accept or deny request from the third-party application (in our case, the third-party application is our system). The user will be redirected to his/her main gadget container page after successful OpenID authentication and authorization.

In our initial JavaScript-based layout manager implementation, gadgets are organized into tab panels. Each tab panel is divided into three columns, each

of which may contain a variable number of gadgets that are chosen by the user. Gadgets can be reorganized using drag and drop. Users can add and remove both gadgets and tab panels. Content of a gadget is embedded in an HTML IFrame element that points to the address (URL) of a rendered gadget from iGoogle server.

4.2 Other Features

We summarize other design features below.

- To make the system open, we use REST-style web services. Any client program that conforms to our calling interface can interface with our gadget management server.
- To make the server side program independent of underlying relational database, object-relational mapping (ORM) techniques are used to convert data between object-oriented programming languages and incompatible type systems of relational database. Hibernate is used in our system.
- The format of messages transmitted between client and server is Java Script Object Notation (JSON). We chose JSON over XML because of JSON's simplicity and relative compactness.
- On the client side, Asynchronous JavaScript and XML technique is used to provide an interactive interface. After comparing different JavaScript libraries, ExtJS was chosen. Currently, the tab layout is supported. In the future, desktop-like layout and tree layout may be implemented.

5 Summary and Conclusion

Previous approaches to building science gateways, such as portlets and Grid services, are being challenged by Web 2.0 and Cloud computing. This paper reviews some of our efforts within the QuakeSim project to evaluate these approaches and their use with the QuakeSim portal. Specific challenges that we have addressed include investigating techniques for building more interactive user interface, integrating high-end supercomputing capabilities for mass job management, and adopting new techniques for managing user interface components based on Open Social gadgets. In particular, we evaluated strategies for optimizing user interactivity with GPS data analysis results. We also described and evaluated an architectural solution for managing large-scale job submissions encountered in QuakeSim and other gateways. Finally, an architectural approach and early implementation details for an Open Social compatible version of the QuakeSim portal (or any gateway) were also presented.

Several interesting problems remain. The Swarm service and the prototype gadget container are relatively new projects. Although Swarm is being tested with realistic submission scenarios, scaling and optimizing the system to

millions of jobs or more will introduce new technical problems. Also, more interestingly, we find that some large job clusters (such as the RDAHMM evaluations of GPS data) can contain jobs ranging from a few seconds execution time to much longer, parallelizable computations, thus providing an excellent experiment for merge Cloud Computing approaches (such as Amazon's EC2) with Grid computing.

The Open Social prototype container system is also in early stages of development. As our next step we plan to evaluate its suitability for building a fully functional version of QuakeSim, and by extension many other gateways. We also plan to integrate the Open Authorization (OAuth) service and clients. This is a compatible feature to OpenID which can be used to determine if the identified user has the authorization to reach particular capabilities.

This work is supported by NASA through the Advanced Information Systems Technology (AIST) program (Andrea Donnellan, PI) and ACCESS program (Yehuda Bock, PI).

6 References

[1] Nancy Wilkins-Diehr: Special Issue: Science Gateways - Common Community Interfaces to Grid Resources. Concurrency and Computation: Practice and Experience 19(6): 743-749 (2007).

[2] Gerhard Klimeck, Michael McLennan, Mark S. Lundstrom, George B. Adams III. "nanoHUB.org - future cyberinfrastructure serving over 75,000 users today", IEEE Symposium on Massive Storage Systems and Technologies (MSST), Baltimore, September 22-24, 2008.

[3] Marlon E. Pierce, Geoffrey C. Fox, Galip Aydin, Zhigang Qi, Andrea Donnellan, Jay Parker and Robert Granat. QuakeSim: Web Services, Portals, and Infrastructure for Geophysics. 2008 IEEE Aerospace Conference, March 1-8 2008, Big Sky MT.

[4] Robert Granat, Galip Aydin, Marlon E. Pierce, Zhigang Qi, Yehuda Bock: Analysis of streaming GPS measurements of surface displacement through a web services environment. CIDM 2007: 750-757

[5] Okada, Yoshimitsu, SURFACE DEFORMATION DUE TO SHEAR AND TENSILE FAULTS IN A HALF-SPACE, 1985, BSSA, vol 75, no. 4, pp 1135-1154.

[6] Jay Parker, Gregory Lyzenga, C. Norton, E. Tisdale, Andrea Donnellan: A community faulted-crust model using PYRAMID on cluster platforms. CLUSTER 2004: 491

[7] Chen, A., Donnellan, A., McLeod, D., Fox, G., Parker, J., Rundle, J., Grant, L., Pierce, M., Gould, M., Chung, S., and Gao, S., Interoperability and Semantics for Heterogeneous Earthquake Science Data, International

Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, Sanibel Island, FL, October 2003.

[8] Jay Alameda, Marcus Christie, Geoffrey Fox, Joe Futrelle, Dennis Gannon, Mihael Hategan, Gopi Kandaswamy, Gregor von Laszewski, Mehmet A. Nacar, Marlon E. Pierce, Eric Roberts, Charles Severance, Mary Thomas: The Open Grid Computing Environments collaboration: portlets and services for science gateways. Concurrency and Computation: Practice and Experience 19(6): 921-942 (2007)

[9] Geoffrey Charles Fox, Dennis Gannon: Special Issue: Workflow in Grid Systems. Concurrency and Computation: Practice and Experience 18(10): 1009-1019 (2006)

[10] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, Y. Zhao, “Scientific Workflow Management and the Kepler System,” Concurrency and Computation: Practice & Experience, 18(10), pp. 1039-1065, 2006

[11] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia C. Laity, Joseph C. Jacob, Daniel S. Katz: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming 13(3): 219-237 (2005)

[12] Thomas M. Oinn, R. Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole A. Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip W. Lord, Matthew R. Pocock, Martin Senger, Robert Stevens, Anil Wipat, Chris Wroe: Taverna: lessons in creating a workflow environment for the life sciences. Concurrency and Computation: Practice and Experience 18(10): 1067-1100 (2006)

[13] Marlon E. Pierce, Geoffrey C. Fox, Jong Y. Choi, Zhenhua Guo, Xiaoming Gao, and Yu Ma. Using Web 2.0 for Scientific Applications and Scientific Communities. Concurrency and Computation: Practice and Experience Special Issue for 3rd International Conference on Semantics, Knowledge and Grid SKG2007 Xian China October 28-30 2007.

[14] “An EGEE Comparative study: Clouds and grids: Evolution or Revolution?” EGEE Technical Document EGEE-II INFSO-RI-031688. Available from https://edms.cern.ch/file/925013/4/EGEE-Grid-Cloud-v1_2.pdf.

[15] Catlett, C. et al. "TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications," HPC and Grids in Action, Ed. Lucio Grandinetti, IOS Press 'Advances in Parallel Computing' series, Amsterdam, 2007.

[16] The Open Science Grid, http://www.opensciencegrid.org/.

[17] Enabling Grids for E-Science (EGEE), http://egee1.eu-egee.org/.

[18] Galip Aydin, Zhigang Qi, Marlon E. Pierce, Geoffrey C. Fox, Yehuda Bock. Architecture, Performance, and Scalability of a Real-Time Global Positioning System Data Grid 17 January 2007, Special issue on Computational Challenges in Geosciences in PEPI (Physics of the Earth and Planetary Interiors) 163 (2007) 347-359, DOI.

[19] OpenSocial, http://www.opensocial.org/

[20] Facebook Developers, http://developers.facebook.com/

[21] The NASA REASoN Project, http://geoinfo.sdsu.edu/reason/

[22] The REASoN Geophysical Resource Web Service (GRWS), http://reason.scign.org/scignDataPortal/grwsSummary.jsp

[23] Apache JMeter, http://jakarta.apache.org/jmeter/

[24] Eduardo Huedo, Rubén S. Montero, Ignacio Martín Llorente, “A framework for adaptive execution in grids,” Soft., Pract. Exper. 34(7): 631-651, 2004

[25] Tadashi Maeno, “PanDA: distributed production and distributed analysis system for ATLAS,” Journal of Physics: Conference Series, 119 062036. 2008

[26] Ioan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster, Mike Wilde, “Falkon: a Fast and Light-weight tasK executiON framework,” in the IEEE/ACM SuperComputing, 2007

[27] Edward Walker, J. P. Gardner, V. Litvin, and E. P. Turner, “Personal Adaptive Clusters as Containers for Scientific Jobs,” Cluster Computing, vol. 10(3), September, 2007

[28] Ewa Deelman, Yolanda Gil, “Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges,” Workflow in e-Science, e-Science 2006, Ammsterdam, December 4-6, 2006.

[29] Douglas Thain, Todd Tannenbaum, Miron Livny: Distributed computing in practice: the Condor experience. Concurrency - Practice and Experience 17(2-4): 323-356 (2005).

[30] Ian T. Foster: Globus Toolkit Version 4: Software for Service-Oriented Systems. J. Comput. Sci. Technol. 21(4): 513-520 (2006)

[31] Daniel Nurmi, John Brevik, Richard Wolski, “QBETS: queue bounds estimation from time series,” SIGMETRICS, 2007: 379-380.

[32] Jim Basney, Marty Humphrey, Von Welch: The MyProxy online credential repository. Softw., Pract. Exper. 35(9): 801-816 (2005)

[33] The OpenID Foundation, http://openid.net/foundation.

[34] The OpenID Simple Registration Extensions, http://openid.net/specs/openid-simple-registration-extension-1_1-01.html

[35] Apache Shindig Incubator Project, http://incubator.apache.org/shindig/