QuakeSim Portal and Services

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

1 Introduction: Gateways in Transition

Science gateways and portals have been developed by numerous groups to support scientific communities. 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] [4] is designed to provide several capabilities to users, including

- Real-time and archival Global Positioning System data from the California Real Time Network and Southern California Integrated GPS Network;
- Services for analyzing GPS data streams using RDAHMM;
- Services for analyzing deformation properties of faults using Disloc, Simplex, and GeoFEST;
- Services accessing the QuakeTables fault model database;
- 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 [5]. 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 [6][7][8][9] are also popular. The various approaches are not mutually exclusive if implemented correctly. 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 local 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 assumptions of science gateways are 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. Science gateways will benefit frm the adoption of both Web 2.0 technologies and development approaches [6]. Gateways also 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 [11], and for-fee computing and storage services provide an alternative to computing grids such as the TeraGrid, Open Science Grid, and EGEE.

We believe however that the overall service architecture remains the same in any case. Cloud services create virtual machines and clusters that themselves can host Web services. Portlets as a specification don't preclude AJAX-style interactivity. 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 API potentially 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.

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 to Grid and Cloud resources. 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 developed by the NASA REASoN project to access historical GPS data, which we then analyze with the RDAHMM application. This analysis is updated daily and 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 wth the REASoN team that is also analyzed with RDAHMM. Analysis results are published every thirty minutes. This work is described

in more detail in [4][12].

This section describes our latest improvements to the system. New features 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 438 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 1994-01-01 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 doing 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 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 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 done to 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 frame of the movie playing.

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 reason that we don't update it to "today" is that stations' states are not stable for the time around "today". E.g., one station does not have GPS input for the date of "today", but after two weeks its input might be available for this date, and thus its state on this date will change from "no-data" to "state change on this day" or "state change within the last 30 days before this date".

Figure 2 Workflow of Station State Change Video Maker Service

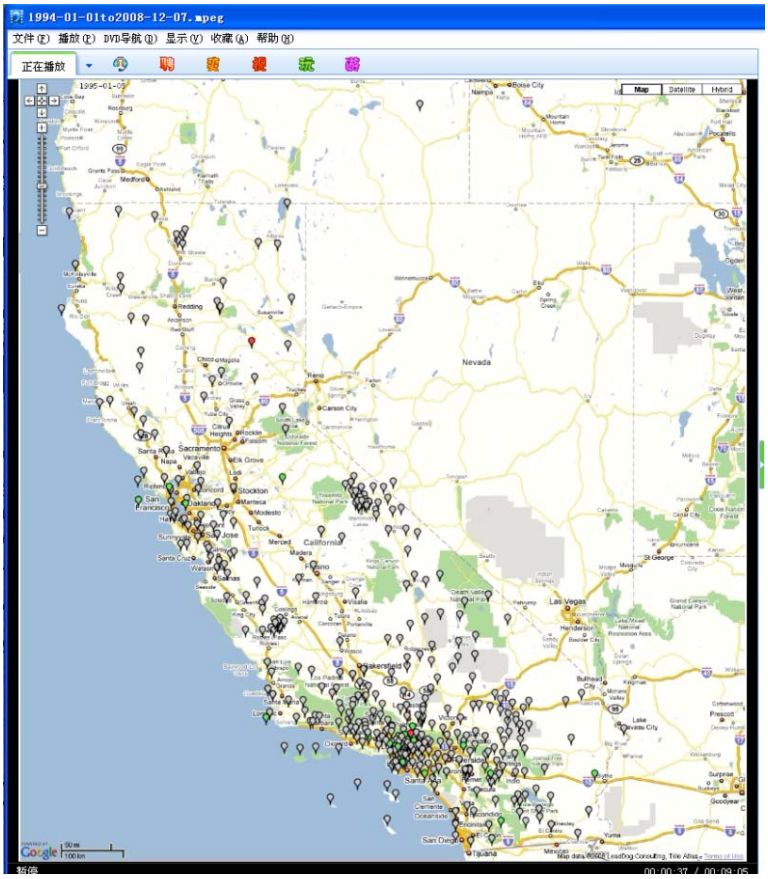


Figure 3 Screen capture of a generated movie of GPS stations' state changes between 1994-01-01 and the current date. Data is obtained from the REASoN project's Geophysical Resources Web Service.

2.3 Daily RDAHMM – Results Analysis Service Performance

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, and is developed to replace the "Managed Bean" solution described in [3]. Managed Beans are a feature of the Java Server Faces framework that we use to develop the user interface components. 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 tried. Table 2 gives a detailed explanation of these solution strategies. Obviously, the web service strategy produces both shorter

Comment [MP1]: You need to show an example of the XML results file and describe how it is generated.

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 of 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-load with JMeter [13] 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 1476ms, which is enough for the requirement of usage in the research community.

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

Table 1 Methods provided by the Daily RDAHMM Result Analysis Service

Method	Description
String getDataLatestDate ()	Calculates the latest date when any station has some input data.
String getLatLongForStation (String stationed)	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 store 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.
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	13s	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

Comment [MP2]: Why so big?

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. Previous work served only as a

demonstration 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 kind of 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 computation intensive because of the large amount of real-time data. We are looking forward to solving this problem with Grid Job Services and better scheduling of the remodeling process. These are described in the following section.

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

3 SWARM: Infrastructure for scheduling large-scale jobs

QuakeSim provides access to a range of application codes. Some of these (Simplex, Disloc, and RDAHMM for a single station) can be run interactively on a modest backend service. Other applications, large numbers of RDAHMM station analysis jobs or very detailed GeoFEST deformation analysis need larger and more powerful resources such as those provided by the NSF TeraGrid. The former is an example of embarrassingly parallel jobs, but the latter (GeoFEST) is a highly scalable parallel

application. We need thus to be able to submit both types of jobs. Our general approach has been to modify application services such as the GeoFEST Web Service so that they act as job management agents that manage remote jobs. That is, the portlet (or other client) contacts the same application service as before, but the application actually is executed on a more powerful remote host. We accomplish this by using the Globus GRAM service to execute codes on TeraGrid resources. Our agent service thus provides an application-specific WSDL interface while serving as a client to the general purpose Grid service. We have implemented our agent service using Apache Axis and chose to wrap Condor-G for handling the mechanics of job submission to Globus. We use Condor's BirdBath Web Service interface and supporting libraries.

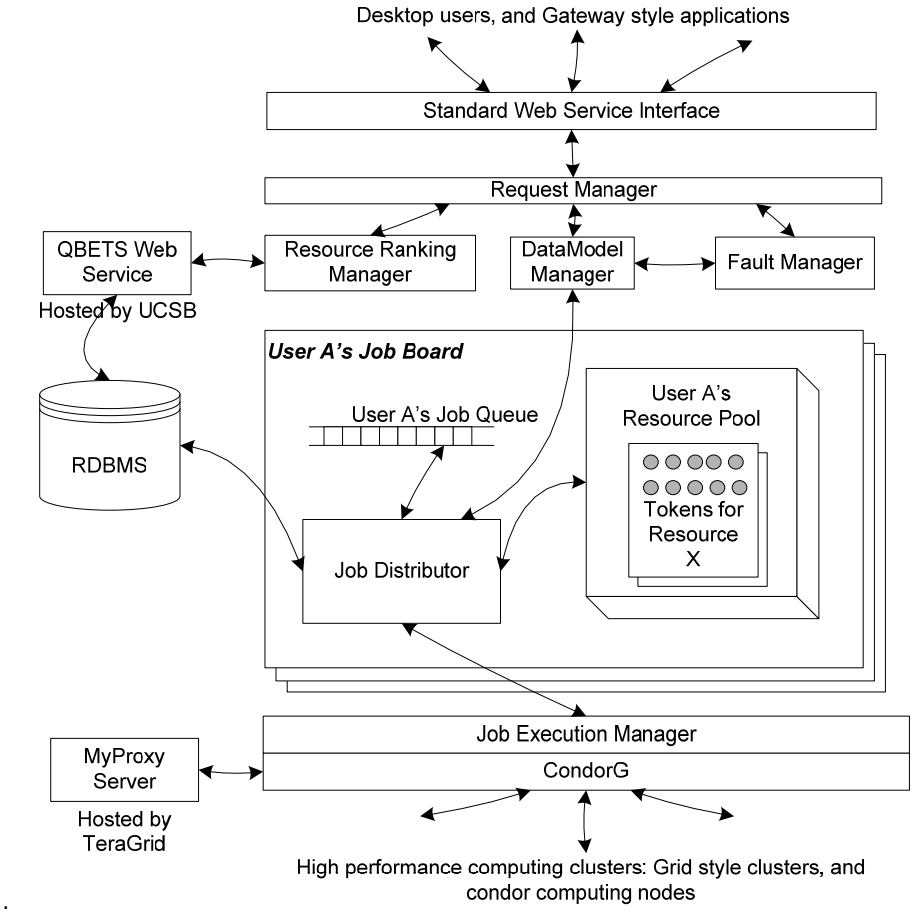
Following these initial considerations, we realized the potential for extending and generalizing this basic service to support additional features. These include finding and submitting to the best available resources and managing many thousands of jobs simultaneously. The latter is very useful for parameter sweeps and other embarrassingly parallel applications such as the GPS station analysis discussed above. The individual applications may be parallel (not just serial). We incorporated these ideas into our Swarm service. Apart from practical considerations for QuakeSim, Swarm gives us a platform for investigating long-standing problems in distributed job management on Grids and Clouds.

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 (or workflows) to highly distributed computing clusters. The jobs submitted by Swarm include serial and parallel jobs that are normally submitted to the batch job systems mostly 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 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 especially in the grid environment. GridWay [14], and PanDa [15] project 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 [16] and myCluster [17] enable users to access provisioned resources to submit large-scale scientific jobs. Instead of provisioning resources, Swarm provides a user-based resource pool, which limits the maximum number submissions to the batch queue system. This enables Swarm to incorporate policies from different batch queue systems more flexibly.

Pegasus [18] provides a workflow generation and mapping environment for grid computing environments. It generates a workflow plan based on the AI reasoning technology. The workflow plan is transformed into a directed acyclic graph which is then passed to the Condor [19] DAGMan system, and then executed on the grid environment. Swarm utilizes Globus [20] technology along with the CondorG and

Birdbath as the foundation job submission mechanism. Instead of using DAGMan, Swarm provides a simple built-in workflow manager that submits individual jobs through CondorG and maintains the status of the submitted jobs.



.

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.

Swarm provides highly extensible design for domain specific applications. Science gateways or Web portals, which are often aimed at specific scientific disciplines, can easily integrate Swarm with their required data model and fault handling scheme. In addition, the lightweight software provides a convenient hosting environment for the scientific applications.

3.1 Architecture

Swarm is a set of Web services and local servers. Figure 5 depicts the architecture of the Swarm framework. From top to bottom, Swarm provides a standard set of Web service interfaces. Desktop users and gateway style applications can easily access

Swarm via standard Web service interfaces. Each of the operations and parameters are defined in the WSDL associated with the services.

The requests from the users are delivered to the Request Manager. The Request Manager creates a 128-bit universally unique identifier ticket for the series of jobs. To provide the capability to track a large number of jobs, Swarm provides a simple structure to the submitted jobs. A job is identified by its ticket and internal ID. Here, internal ID is the identity of the job, which is unique within the job group. This structure is especially useful for the web application, which deals with multiple experiments launched by multiple users.

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 resources (computing clusters) to submit the job. To prioritize the resources listed in the user's job description, Swarm interacts with the QBETS batch queue prediction service [21]. The QBETS service provides queue delay predictions. The WallClockTime and number of nodes are key factors to get the predicted delay. Here the WallClockTime is duration of the execution of the job and the number-of-nodes is number of the computing nodes for the parallel jobs. The WallClockTime and the number of nodes are specified in the 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 determines 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 easily 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 categorizes faults into two catagories: *fatal fault* and *recoverable fault*. A fatal fault is defined as faults 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 path to the input data
- Hardware and software failures in the computing clusters
- Failures resulting from the policy of the computing cluster

Recoverable fault refers to faults that can possibly be recovered without contacting the user. It is mostly related to resource specifications such as expected execution time or memory requirements. When Swarm suspects 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 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.

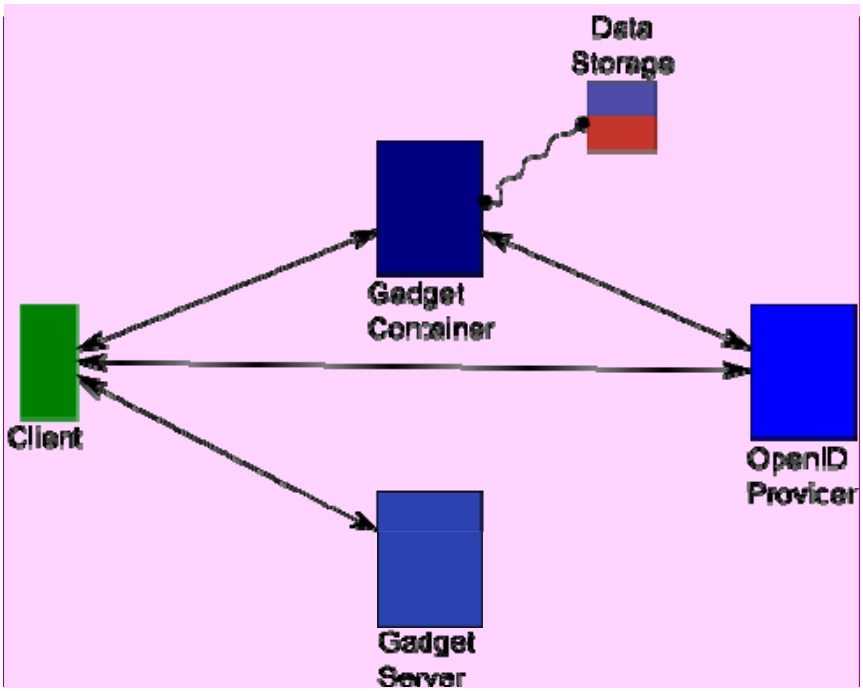
When the Job Distributor finds a match with an available remote resource, the Job Execution Manager will submit the job through CondorG's Web service APIs. The user's certificate (based on the X.509) is retrieved by means of interacting with the MyProxy service and used to access to the Globus GRAM job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

4 Developing a Gadget Container

Gadget container is a system by which users can build their own "iGoogle" system. It consists of user authentication system, user administration system and gadget management system. User authentication system allows our system to accept new user registration and login of existing users. User administration system provides a way for administrators to manage all users conveniently. And gadget management system allows users to manage gadgets in convenient way.

Shindig is an open source project in the Apache Software Foundation incubator. It implements gadget specification and OpenSocial specification. It includes server side implementation and does not have front-end layout manager.

4.1 Architecture



Comment [MP3]: This figure should be approved.

Figure 6 Gadget container architecture. Arrows represent HTTP-based communications. The curved line represents a JDBC connetction.

User registration and login: 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 user can bind account to OpenID. Binding is only needed to be done once and after that user can log in our system 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 user for login. After that, the user is asked to accept or deny request from 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.

Gadget container layout management: Gadgets are organized into tab panels. And in each table panel, there are three columns each of which contains variable number of gadgets. Gadgets can be reorganized using drag and drop. Users can add and remove tab panels. Of course users can add and remove gadgets. Content of a gadget is embedded in an *iframe* html element. So you can make an *iframe* point to address(URL) of a rendered gadget from iGoogle server.

4.2 Technical Details

Java Server Faces (JSF) technique was used at first as server side framework. However, there are several drawbacks using JSF. UI logic entirely runs on the server nevertheless some logic can be done at client side. For example, when a new user creates an account, input information can be verified at client side before being sent to server. In addition, performance cost is high because of complex processing cycle for

each JSF request. Also JSF couples client with server tightly so that adding a new type of components is not easy.

To make the system open, we use RESTful web services. So any client program that conform 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) technique is used to convert data between object-oriented programming languages and incompatible type systems of relational database. Hibernate is used in our system.

Attributes of user profile are compatible to OpenID Simple Registration Extension.
Format of messages transmitted between client and server is Java Script Object Notation (JSON). We chosen JSON over XML because of its simplicity.

At client side, Asynchronous JavaScript and XML technique is used to give user-friendly interface. After comparing some different javascript libraries, ExtJS was chosen. Currently, tab layout is supported. In the future, desktop-like layout and tree layout may be implemented.

5 Conclusion

This paper has reviewed progress of the QuakeSim portal and services. Challenges include adopting more interactive user interface building techniques demanded by users, managing and processing multiple data streams, integrating high-end supercomputing capabilities for mass job management, and adopting new techniques for managing user interface components.

Acknowledgements: 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.[1]

[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] 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)

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

[7] 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

[8] 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)

[9] 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)

[10] 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.

[11] "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.

[12] 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.

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

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

[15] Tadashi Maeno, "PanDA: distributed production and distributed analysis system

for ATLAS," Journal of Physics: Conference Series, 119 062036. 2008

[16] 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

[17] 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

[18] 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.

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

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

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

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

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

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