Publishing and Consuming GLUE v2.0 Resource Information in XSEDE

Warren Smith
Texas Advanced Computing Center
University of Texas at Austin
10100 Burnet Road (R8700)
Austin, TX 78758-4497
wsmith@tacc.utexas.edu

Sudhakar Pamidighantam
Research Technologies, UITS
Indiana University
2709 East 10th St.
Bloomington, IN 47408
pamidigs@iu.edu

John-Paul Navarro
Mathematics and Computer Science Division
Argonne National Laboratory
9700 S. Cass Avenue
Argonne, IL 60439
navarro@mcs.anl.gov

ABSTRACT

XSEDE users, science gateways, and services need a variety of accurate information about XSEDE resources so that they can use those resources effectively. They need information to decide which resources to use, to track their usage of resources, and to provide services to their users. To support this, XSEDE is deploying a new system to gather and publish static and dynamic resource information. This paper gives an overview of the resource information available with this new system, describes the design and performance of the software and services that make up this system, and finally provides examples of how to use this new resource information.

1. INTRODUCTION

XSEDE [35] is an infrastructure funded by the National Science Foundation that integrates a set of large resources for scientific simulation and analysis. These resources support a variety of users and usage models. Large-scale compute resources are available for tightly coupled parallel simulations, loosely coupled analyses, or workflows that consist of a number of different types of computation. Visualization resources allow users to analyze their results, and data storage resources allow short- to long-term storage of user data. In addition, new resource types, such as cloud computing and data analysis resources, are being added to XSEDE.

The mission of the XSEDE project is to help users make use of the diverse types of resources that are part of XSEDE. We accomplish this in a variety of technical and nontechnical ways, such as by providing training and advanced user support and by deploying common software and services across XSEDE resources. As part of this effort, XSEDE has deployed services that provide a variety of information about XSEDE. Such information is important so that users, science gateways, and tools have accurate information about XSEDE and can therefore make good decisions about how to use XSEDE.

This paper describes new software and services that provide static and dynamic resource information to XSEDE users. XSEDE initially used legacy TeraGrid information services that provided less information and that information that was typically out of date by the time it was received by consumers. Our new design addresses these problems in several ways. First, we created a new piece of software called the Information Publishing Framework to efficiently gather information from XSEDE resources and publish it. This software gathers more information than was gathered during TeraGrid. Second, we adopt GLUE v2.0, a standard information model and JSON schema that can represent a wide variety of grid resources, services, and software. Third, we deliver information very quickly from producers to consumers via a new XSEDE publish/subscribe messaging system.

This paper also provides examples of how to use this new information system. We describe a general approach to receiving and processing information from the XSEDE messaging system. In addition, this paper includes two specific examples of services which already use this information system. The first example is the SEAGrid science gateway, which uses GLUE v2.0 information to maintain a representation of the current state of XSEDE. The second example is the Karnak queue prediction service that uses historical resource information to train predictors and uses current resource information when making predictions. The final section of this paper describes the current status of this work and our future plans.

2. RESOURCE INFORMATION

A number of different types of resource information are valuable to users and tools. In this work, we focus on providing the following:

1. Summaries of the static and dynamic state of resources. A tool such as the XSEDE user portal can use this information to provide an at-a-glance overview of XSEDE.

2. Descriptions of the software installed on a resource. A science gateway can use this information to determine which resources have the software needed to perform an analysis.

XSEDE '15, July 26 - 30, 2015, St. Louis, MO, USA
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-3720-5/15/07...$15.00
DOI: http://dx.doi.org/10.1145/2792745.2792770

3. Description of queued jobs. A service can use this information to suggest where to submit new jobs so that they will execute quickest.

4. Individual job status. A workflow tool can use the status of individual jobs to trigger succeeding tasks.

In addition to making this information available, its timeliness is also important. For example, if a scheduling service has queue information that is several minutes old, it would not know about recent large job submissions from other sources and could submit jobs to a system that is now busy. Another example is that a complex workflow can execute much more quickly if it is notified of job state changes within seconds, rather than within minutes.

3. DESIGN

The architecture we use to provide the resource information described in the previous section is shown in Figure 1. The lowest level of the architecture consists of information gatherers running on XSEDE resources. These gatherers are implemented by using the Information Publishing Framework [14, 13] and gather information from batch schedulers and module files. When the information gatherers have new information, they publish it to the messaging system.

The middle layer of our architecture is a fault tolerant publish/subscribe messaging system. The messaging system allows subscribers to register interest in message types and accepts messages from publishers. When messages arrive, the messaging system delivers them to the subscribers that wish to receive them. This approach decouples information producers from consumers but can still deliver high volumes of information with low latency.

The top layer of the architecture is consumers of resource information. This layer consists of tools such as the SEAGrid science gateway [6] and the Karnak queue prediction service [31]. These tools subscribe for the types of resource information they wish to receive and then process and act on the information they receive.

3.1 Information Publishing Framework

XSEDE is deploying the Information Publishing Framework (IPF) software to gather information about XSEDE resources. IPF is implemented in Python and executes information gathering and publishing workflows. These workflows are defined as JSON [15] documents and they specify a set of steps to execute. Each step can require input data, can produce output data, and can publish representations of data. Many steps have been implemented to discover information about resources and to monitor resource state. A typical workflow consists of a number of information gathering steps along with a few steps that publish representations. Steps are currently available that can publish to local files, REST services, and messaging services.

Since steps specify what data they require and produce, IPF can construct workflows based on partial information. A common case is that a JSON workflow specification simply lists the steps that should be executed and IPF connects these steps into a workflow based on the input and output types of the steps. Another example is that if a workflow does not specify steps to generate some inputs that are needed by specified steps, IPF examines its catalog of steps and adds the needed steps to the workflow.

Workflows can run to completion relatively quickly or they can run continuously. The first type of workflow can be run out of cron and can be used to run a few commands or look at status files and publish that information. The second type of workflow can be run as init processes and continuously monitor log files or periodically run commands to gather and publish new information.

IPF can be used to gather and publish a wide variety of information. In this work, we focus on using IPF to gather static and dynamic information about resources by interacting with batch scheduling systems and module files. IPF has steps to retrieve job, queue, and node information from scheduling systems such as HTCondor, OpenStack, LSF, Moab, SGE, SLURM, and Torque. IPF can also retrieve module information from the modules package and from lmod. In the next section, we describe how IPF publishes this information using the JSON rendering of the GLUE v2.0 standard.

3.2 GLUE v2.0

The Grid Laboratory for a Uniform Environment (GLUE) schema was originally developed by several multi-institution physics project to describe the resources and services available to them [9, 4]. This data was stored in Lightweight Directory Access Protocol servers and accessed by users and other services to help decide how best to process physics data generated by instruments and how best to perform physics simulations. The original GLUE effort was successful and widely adopted in that community.

This success, the applicability of GLUE to other distributed infrastructures, and the identification of improvements to GLUE resulted in the GLUE v2.0 effort in the Open Grid Forum [20]. This effort first defined a generic GLUE v2.0 information model [3] and then defined different representations of this model, called renderings.

A number of entities are defined in GLUE v2.0. In addition to generic entities, there are entities to describe compute infrastructure and ones to describe storage infrastructure. In this work, we use the following entities to describe compute infrastructure:

- ComputingManager and ComputingService, which provide a summary of a batch scheduler.

- ComputingShare, which describes a queue including a summary of the jobs associated with the queue.

- ComputingActivity, which provides detailed information about a batch job.

- ExecutionEnvironment, which describes a set of identical
  nodes including the number of nodes that are available, down, or in use.

- ApplicationEnvironment and ApplicationHandle, which provide informabion about an installed software package and its associated module.

XSEDE organizes these entities into several different documents for publication, as described in Section 3.3.

XSEDE uses the JSON rendering because JSON is the representation preferred by many consumers of information in XSEDE, such as the XSEDE user portal and science gateways. JSON is popular because it is a simple way to repre-

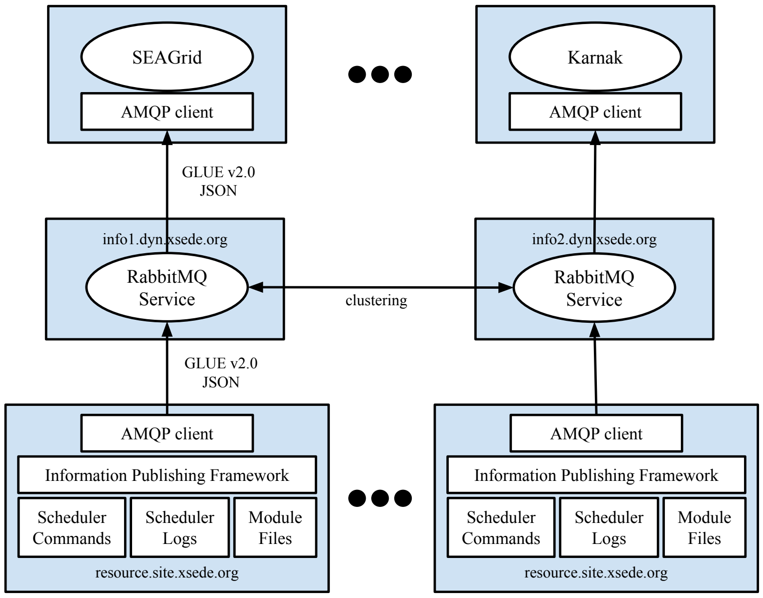


Figure 1: XSEDE publish/subscribe messaging architecture with GLUE v2.0 information.

sent machine-parsable information and is also easy for humans to read.

3.3 Publish/Subscribe Messaging

XSEDE has deployed a publish/subscribe messaging service to distribute information gathered from XSEDE resources to consumers that wish to receive this information. The publish/subscribe messaging model deployed by XSEDE consists of the following:

- Producers of information, which publish **messages** to the
messaging service as they have new information to publish. A message consists of data and a routing key that provides a small amount of metadata about the message. Messages are delivered to **exchanges** in **virtual hosts** in the messaging service. An exchange is a logical location to publish messages to, and a virtual host is a logical container of other entities.
- Consumers, which create a **queue** in the messaging service to hold messages until the consumer is ready to receive them. The consumer then binds a queue to an exchange with a **filter** that describes what messages should be forwarded from the exchange to the queue.
- The messaging service, which receives messages, routes messages to queues, and then sends messages to consumers. If the consumer is connected when a message arrives in its queue, the message is sent to it immediately. If the consumer is not connected, messages will be sent to the consumer when it does connect.

An important characteristic of this approach is that information is delivered very quickly from where it is produced to the tools that use the information.

We selected a standard messaging protocol called the Advanced Message Queuing Protocol (AMQP) [1, 2]. Several production-quality messaging services implement this standard, as well as a wide variety of client libraries. Selecting a standard protocol allows us to more easily switch to a different client library or messaging service if we encounter problems with specific software.

We selected RabbitMQ [25] as our AMQP service. RabbitMQ is a production-quality messaging service that provides mechanisms for scalability and fault tolerance and has been shown to have high performance [32, 26].

While some of the published information is not sensitive and can be made available to any consumer, other information (detailed job descriptions) is sensitive and can be made available only to XSEDE participants. A consumer that wants sensitive information must authenticate to the messaging service, typically via an X.509 certificate, and be authorized. All publishers of information must authenticate, again typically via an X.509 certificate.

XSEDE has configured the four exchanges shown in Table 1 in the `xsede` virtual host to distribute GLUE v2.0 information. The `glue2.applications` exchange receives messages that describe the modules available on an XSEDE resource. The messages use the XSEDE resource name as their routing key, and the message contains a JSON document that uses ApplicationEnvironment and ApplicationHandle GLUE v2.0 entities to describe modules. These documents are published relatively infrequently (hourly) and can be consumed without having to authenticate to RabbitMQ.

The `glue2.compute` exchange receives messages that provide high-level information about a resource. The messages use the XSEDE resource name as their routing key and contain a JSON document with ComputingService, ComputingManager, ComputingShare, and ExecutionEnvironment GLUE v2.0 entities. These documents are published frequently (every minute or two) so that consumers have accurate information about the dynamic state of XSEDE. These messages can also be consumed without authentication.

The `glue2.computing_activities` exchange receives messages that contain the detailed queue state of a resource. The routing key is again the XSEDE resource name. The

Table 1: RabbitMQ exchanges.

Exchange	Information Type	Authenticate
glue2.applications	Module definitions	No
glue2.compute	Compute resource description	No
glue2.computing_activities	Queue state	Yes
glue2.computing_activity	Job information	Yes

content of each message is a JSON document containing a list of GLUE v2.0 ComputingActivity entities. Each of these entities describes a job being managed by the batch scheduler for that resource. These documents are also published frequently (every minute or two) so that consumers have accurate information about the dynamic state of XSEDE. A consumer must authenticate and be authorized to consume these messages.

The `glue2.computing_activity` exchange receives messages that contain updates for individual jobs being managed by XSEDE resources. The routing keys for these messages are more complex and include the batch scheduler job identifier, the local user who owns the job, and the name of the XSEDE resource. The extra information in these routing keys allows consumers to more effectively filter the messages (for example, only job information for a specific user). Each message contains a single ComputingActivity with updated information about that job. These messages are published as jobs state change entries appear in the batch scheduler logs of XSEDE resources.

4. PERFORMANCE

In previous work [32], we evaluated the performance of RabbitMQ under loads as large as and larger than the loads that XSEDE places on it. We found that RabbitMQ easily handles this volume of messaging traffic in terms of throughput (messages per second) and bandwidth (MB per second). In addition we found that a significant amount of excess capacity is available.

To verify these results, we observed the age of the GLUE v2.0 messages received from the XSEDE RabbitMQ services. These messages include a creation time attribute that we compared against the current time on the receiving server. These results are shown in Table 2. As the table shows, information is available to the client at most 10 seconds after the producer created the GLUE v2.0 document. On average, information is available in less than 2 seconds. When these observations were made, information about the TACC Stampede system was consistently the oldest. The reason is that Stampede was publishing more than 5 times the amount of information as the other systems, primarily because of the number of jobs it was managing at this time. It therefore tooks longer to gather, organize, and publish these larger documents.

We note that system clocks are generally well synchronized across XSEDE, but this is not guaranteed. Our observations did not indicate that any of the clocks were significantly unsynchronized: The time the client reported receiving each message was after the time the publisher reported creating it, and while it took several seconds longer to receive information from Stampede, this is explained by the amount of information being organized and sent from Stampede.

Table 2: Age of received GLUE v2.0 information

	Age (seconds)	
Exchange	Maximum	Mean
glue2.compute	5.8	2.1
glue2.computing_activities	10.1	3.5
glue2.computing_activity	1.1	0.4

5. USING XSEDE GLUE V2.0

Using the GLUE v2.0 information published by XSEDE is relatively easy. Many programming languages have AMQP client libraries so consumers can connect to RabbitMQ to receive GLUE v2.0 messages. These libraries support client authentication via X.509 certificates or username/password, as XSEDE requires for a consumer to receive messages containing job information.

Typically, the programming language used by the consumer has several JSON parsing libraries. For example, Python has the json module and Java has the Jackson library. JSON libraries often parse JSON documents into generic programming structures such as lists and maps that are relatively easy to use. A more advanced approach is to use a JSON library that parses JSON documents into specific GLUE v2.0 objects. The Open Grid Forum GLUE working group provides such Plain Old Java Objects (POJO) for use with the Jackson data processing toolkit [16]. These objects are generated from the GLUE v2.0 JSON schema definition using jsonschema2pojo [17] and some post processing.

Figure 5 provides an example of how to use the RabbitMQ Java client library, Jackson, and the POJO classes from the Glue working group to handle GLUE v2.0 documents published by XSEDE. This example consumes from the `glue2.compute` exchange, which does not require authentication. To connect to one of the exchanges that requires authentication, see the examples on the RabbitMQ SSL [27] web page for how to configure the ConnectionFactory.

The example connects anonymously to the `xsede` virtual host of the RabbitMQ service running on `info1.dyn.xsede.org`. When creating the connection factory, no authentication information is provided so the default user `guest` and password `guest` is used. An exchange `glue2.compute` already exists to receive messages from IPF, so the code only creates a queue and binds the queue to the exchange using a filter of `#`. This filter indicates that the consumer wants to receive messages with any routing key. The example then begins to consume messages, and `handleDelivery()` is invoked whenever a message arrives. For each message, the Jackson library is used to parse it into the GLUE v2.0 POJO objects and the consumer then handles these objects as it likes.

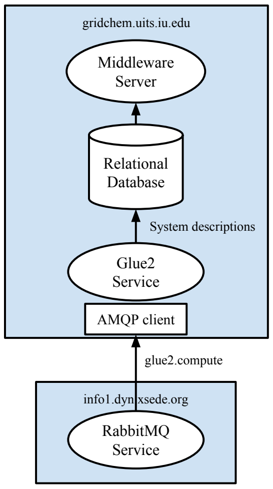
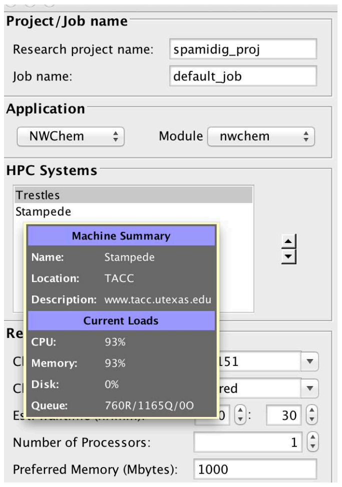


Figure 2: Integration of SEAGrid gateway and XSEDE GLUE v2.0 publishing.



Project/Job name

Research project name: spamidig_proj

Job name: default_job

Application

NWChem   Module nwchem

HPC Systems

Trestles
Stampede

Machine Summary

Name:	Stampede
Location:	TACC
Description:	www.tacc.utexas.edu

Current Loads

CPU:	93%
Memory:	93%
Disk:	0%
Queue:	760R/1165Q/0O

Number of Processors: 1

Preferred Memory (Mbytes): 1000

Figure 3: Resource-level load information for a given HPC system available in GridChem client.

5.1 SEAGrid

Our first example of using GLUE v2.0 information in XSEDE is the Science and Engineering Applications Grid (SEAGrid) science gateway [30]. SEAGrid, previously known as the Computational Chemistry Grid [6], is a virtually organized community cyberinfrastructure [34]. The cyber component of the infrastructure is made up of several high-performance XSEDE resources, a middleware server, and mass storage facilities to archive user data. The organization has provided over 6.4 million XSEDE service units of allocated CPU time to execute 15,280 jobs from 12 applications of in the last year. The infrastructure provides a Java-based desktop client application named GridChem/DESSERT. Users typically select an application that is supported by the organization from a menu and subsequently choose an HPC resource to execute the application. Inputs and other job parameters such as memory, number of processors, wall time, and a queue are also specified by the user. The estimation of these job parameters is currently based on user intuition and experience, and estimating across the resources available in XSEDE is difficult. The GridChem/DESSERT client should ideally provide the dynamic conditions such as the how loaded the resource is in terms of the number of jobs queued ahead and the queue waittime and runtime estimates based on the user's initial choice and perhaps also advise the user of other configurations that are estimated to have comparable or better completion times for the run. Between the estimated start and runtimes and the dynamic information regarding resources users would have all the information needed to select the best resource and queue for their job. The queue waittime estimation for XSEDE resources has been previously implemented [8]. Runtime predictions are application dependent, and work is ongoing to provide reasonable predictions for some SEAGrid applications. Here we discuss the inclusion of dynamic resource information into the GridChem/DESSERT client.

SEAGrid currently receives XSEDE GLUE v2.0 information as shown in Figure 2. A Glue2 Service, which is an independent process from the rest of the SEAGrid services, subscribes to the glue2.compute RabbitMQ exchange and receives updates on the state of XSEDE resources. This modular architecture allows components of SEAGrid to be updated or replaced individually without affecting the entire system.

As this service receives state updates, it writes to the SEAGrid relational database to set the current number of jobs in the running, waiting, or other states for each queue and for each resource as a whole. It also writes the current CPU and memory load of each resource (fraction of total CPU/memory in use). The SEAGrid middleware server then provides this information to GridChem/DESSERT graphical clients, and it is presented during the job submission process as depicted in Figure 3. The availability of this dynamic information for various queues and resources helps users make the best choice for their run.

5.2 Karnak

Our second example of using XSEDE GLUE v2.0 information is the Karnak service [33, 31], which provides predictions about how long batch jobs will wait in scheduling queues on XSEDE systems before they begin to execute. Users can request predictions for jobs that they want to submit or that have already been submitted. For a hypothetical job, the user provides the number of cores, the requested execution time, and a list of queue/system combinations. For an already submitted job, the user simply provides a job identifier and a system name. In either case, the service replies with a predicted wait time (represented either as a duration to wait or as a date/time the job will start) and a prediction interval that indicates how confident the service is of the prediction. For example, the service might reply to a query with 4 hours ±1 hour for queue Q1 on system S1 and 6 hours ±3 hours for queue Q2 on system S2.

The service creates predictions using machine learning where a predictor is trained by using past information and is then used to make predictions about what will occur in the future. To train predictors, the Karnak service needs historical information about job wait times on XSEDE resources. For each job, this information includes a description of the job (for example, the number of cores and the requested wall time) and a description of the queue state at the time a prediction would have been made (for example, right before the

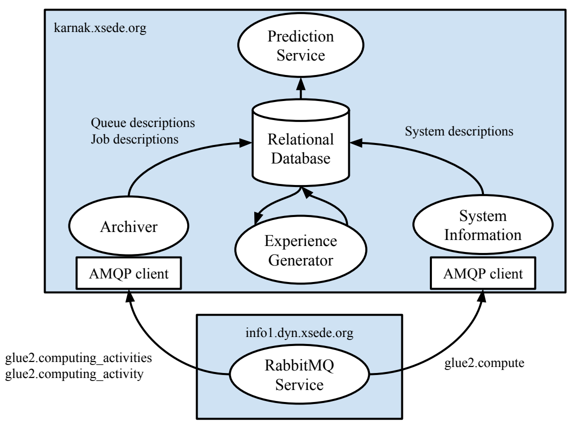


Figure 4: Integration of the Karnak service and XSEDE GLUE v2.0 publishing.

job is submitted). The description of the queue state consists of characteristics such as the number of jobs waiting to run and the amount of cores and node hours requested by waiting jobs. To make a prediction, the Karnak service needs information about the job being predicted and about the current state of XSEDE queues.

Figure 4 shows how the Karnak service obtains the information it needs and processes that information. An Archiver daemon, written in Python, is subscribed to the `glue2.computing_activities` and the `glue2.computing_activity` RabbitMQ exchanges. The daemon receives descriptions of queue states over the first exchange and updates on individual job states over the second exchange. The Archiver daemon writes the queue and job information that it receives to the relational database. A second System Information daemon is subscribed to the `glue2.compute` exchange and is receiving descriptions of system hardware and the queues used for managing jobs on a system. This daemon also updates the relational database with this information.

As this information is added to the database, a daemon is generating experiences from that information, and the prediction service is using these experiences to train predictors. The predictors in the prediction service then use information about the current state of XSEDE systems to provide predictions to users.

6. RELATED WORK

XSEDE is a continuation of TeraGrid [5] and is still partly using the TeraGrid information services. There are a variety of TeraGrid/XSEDE information subsystems [18], and these subsystems are only partially integrated. The Integrated Information Service (IIS) is implemented using the Globus WS-MDS [29] and contains information about resource configuration, resource load, and the software and services deployed on systems.

The Open Science Grid (OSG) is a consortium of eighty sites that advances science through open distributed computing [22]. OSG uses an older version of GLUE to publish resource and software information using a Condor-based Resource Selection Service (ReSS) service [10]. The GLUE data is collected centrally into an LDAP-based server called the Berkeley Database Information Index (BDII) [21]. For monitoring, OSG utilizes the Resource and Service Validation (RSV) software [28] consisting of a client that executes a number of tests and publishes it to a centralized accounting service called Gratia [11].

The Partnership for Advanced Computing in Europe (PRACE) spans twenty-four countries to provide a supercomputing infrastructure for Europe [23]. Like XSEDE, PRACE utilizes Inca to verify its software infrastructure, the PRACE Common Production Environment [24], and uses perfSONAR for network monitoring [24].

The European Grid Infrastructure (EGI) is a federation of approximately forty resource providers to deliver a sustainable, integrated and secure computing services to European researchers and their international partners [7]. For monitoring, EGI uses Gstat [12], a monitoring solution built on top of Nagios [19]. EGI has deployed several instances of the ActiveMQ message broker and is experimenting with using messaging in their infrastructure. One example is publishing status information gathered by tools such as Nagios to these brokers.

7. STATUS AND FUTURE WORK

A pilot version of the information system described here has been deployed on XSEDE since 2014. This deployment includes two RabbitMQ servers, publishing of GLUE v2.0 JSON documents, and information gathering from many XSEDE resources. This pilot has been used to validate our approach as well as refine and fix bugs in the IPF software. At the time of publication, this information system is being tested by XSEDE Operations. Once the software passes this testing, it will be deployed in production on XSEDE.

The GLUE v2.0 information provided via messaging has been integrated into the SEAGrid science gateway and the Karnak queue prediction service. SEAGrid is using this information to keep an up-to-date representation of the dynamic state of XSEDE resources and provides this information to its users so that they can better select where to perform their analyses. The Karnak service uses GLUE v2.0 information to know the current state of XSEDE resources and to maintain an archive of jobs and the past state of resources. Karnak uses this information to train queue wait-time predictors and to make predictions given the current state of XSEDE.

In the future, XSEDE will publish additional GLUE v2.0 information such as Services and Endpoints so that users and tools can learn what services are available in XSEDE and how to contact these services. We also expect to publish GLUE v2.0 storage entities to describe the storage systems attached to XSEDE resources. Additionally, XSEDE will publish additional types of information via RabbitMQ, such as Inca and Nagios test results, and will explore publishing other types of information, such as accounting data.

While many consumers are best served by a publish/subscribe messaging interface, other consumers prefer a query/response interface. To support this, XSEDE plans to deploy a REST interface that will be layered atop an information warehouse that receives its information via the messaging service. This warehouse will contain both current information and historical information.

The RabbitMQ messaging system that we have deployed has a significant amount of unused capacity. We will investigate using this unused capacity to provide Messaging as a Service so that XSEDE users and gateways can publish custom information for their own use.

Acknowledgments

This material was based upon work was supported in part by the National Science Foundation under grant 1053575 and the U.S. Department of Energy Office of Science under Contract DE-AC02-06CH11357.

8. REFERENCES

[1] S. Aiyagari et al. AMQP: Advanced Message Queuing Protocol Specification Version 0.9.1. Technical Report 0.9.1, AMQP Working Group, November 2008.

[2] AMQP: Advanced Message Queuing Protocol. http://www.amqp.org.

[3] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, and J. Navarro. GLUE Specification v. 2.0. Technical Report GFD-R-P.147, The Open Grid Forum, March 2009.

[4] S. Burke, S. Andreozzi, and L. Field. Experiences with the GLUE Information Schema in the LCG/EGEE Production Grid. In *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP 2007)*, 2007.

[5] Charlie Catlett. The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility. In *Proceedings of the 2nd International Symposium on Cluster Computing and the Grid*, 2002.

[6] R. Dooley, K. Milfeld, C. Guiang, S. Pamidighantam, and G. Allen. From Proposal to Production Lessons Learned Developing the Computational Chemistry Grid Cyberinfrastructure. *Journal of Grid Computing*, 4:195–208, 2006.

[7] European Grid Infrastructure – towards a sustainable infrastructure. http://www.egi.eu.

[8] Y. Fan, S. Pamidighantam, and W. Smith. Incorporating Job Predictions into the SEAGrid Science Gatewa. In *Proceedings of the XSEDE'14 Conference*, July 2014.

[9] L. Field and M. W. Schulz. Grid Deployment Experiences: The path to a production quality LDAP based grid information system. In *Proceedings of the Conference for Computing in High-Energy and Nuclear Physics*, pages 723–726, 2004.

[10] G. Garzoglio, T. Levshina, P. Mhashilkar, and S. Timm. ReSS: A Resource Selection Service for the Open Science Grid. Technical report, Fermilab, 2008.

[11] Gratia. https://www.opensciencegrid.org/bin/view/Accounting/WebHome.

[12] Gstat 2.0. http://gstat2.grid.sinica.edu.tw.

[13] M. Hanlon, W. Smith, and S. Mock. Providing Resource Information to Users of a National Computing Center. In *Proceedings of the XSEDE13 conference*, July 2013.

[14] The Information Publishing Framework. https://bitbucket.org/wwsmith/ipf.

[15] Introducing JSON. http://www.json.org.

[16] JSON Java examples from the Open Grid Forum Glue working group. https://github.com/OGF-GLUE/JSON/tree/master/examples/java.

[17] Generate Plain Old Java Objects from JSON or JSON-Schema. http://www.jsonschema2pojo.org.

[18] L. Liming et al. TeraGrid's integrated information service. In *Proceedings of the 5th Grid Computing Environments Workshop*, GCE '09, pages 8:1–8:10, New York, NY, USA, 2009. ACM.

[19] Nagios - The Industry Standard In IT Infrastructure Monitoring. http://www.nagios.org.

[20] Open Grid Forum. http://www.ogf.org.

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

[22] R. Pordes et al. The Open Science Grid. *Journal of Physics: Conference Series*, 78, 2007.

[23] PRACE Research Infrastructure - The top level of the European HPC ecosystem. http://www.prace-project.eu.

[24] First Annual Operations Report of the Tier-1 Service. http://www.prace-ri.eu/IMG/pdf/D6-1_2ip.pdf.

[25] RabbitMQ: Messaging that just works. http://www.rabbitmq.com.

[26] RabbitMQ Performance Measurements, part 2. http://www.rabbitmq.com/blog/2012/04/25/rabbitmq-performance-measurements-part-2/.

[27] RabbitMQ SSl Support. https://www.rabbitmq.com/ssl.html.

[28] The Resource and Service Validation (RSV) Service. http//www.opensciencegrid.org/bin/view/Documentation/Release3/RsvOverview.

[29] J. M. Schopf, L. Pearlman, N. Miller, C. Kesselman, and A. Chervenak. Monitoring the grid with the Globus Toolkit MDS4. *Journal of Physics: Conference Series*, 46, 2006.

[30] N. Shen, Y. Fan, and S. Pamidighantam. E-science infrastructures for molecular modeling and parametrization. *Journal of Computational Science*, 5(4):576 – 589, 2014.

[31] W. Smith. A Service for Queue Prediction and Job Statistics. In *Proceedings of the 6th Gateway Computing Environments Workshop (GCE '10)*, November 2010.

[32] W. Smith and S. Smallen. Building an Information System for a Distributed Testbed. In *Proceedings of the XSEDE'14 conference*, July 2014.

[33] The Karnak Prediction Service. http://www.karnak.xsede.org.

[34] N. Wilkins-Diehr, D. Gannon, G. Klimeck, S. Oster, and S. Pamidighantam. TeraGrid Science Gateways and Their Impact on Science. *IEEE Computer*, 41(11):32–41, 2008.

[35] XSEDE: eXtreme Science and Engineering Discovery Environment. http://www.xsede.org.

```
import com.fasterxml.jackson.databind.ObjectMapper;
import com.rabbitmq.client.*;

public class SubscribeGlue2 {
    public static void main(String[] argv) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(``info1.dyn.xsede.org'');
        factory.setPort(5672);
        factory.setVirtualHost(``xsede'');
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        String queueName = channel.queueDeclare().getQueue();
        channel.queueBind(queueName,''glue2.compute'',''\#'');
        channel.basicConsume(queueName,true,new Glue2Consumer());

        System.out.println(``**** To exit press CTRL+C ****'');
        while(true) {
            try {
                Thread.sleep(1*1000);
            } catch (InterruptedException e) { }
        }
    }
}

class Glue2Consumer implements Consumer {
    public Glue2Consumer() { }
    public void handleConsumeOk(String consumerTag) { }
    public void handleCancel(String consumerTag) { }
    public void handleCancelOk(String consumerTag) { }
    public void handleShutdownSignal(String consumerTag, ShutdownSignalException sig) { }
    public void handleRecoverOk(String consumerTag) { }

    public void handleDelivery(String consumerTag,
                               Envelope envelope,
                               AMQP.BasicProperties properties,
                               byte[] body) throws java.io.IOException {
        System.out.println(envelope.getRoutingKey());
        ObjectMapper mapper = new ObjectMapper();
        try {
            org.ogf.glue2.Glue2 glue2 = mapper.readValue(body,org.ogf.glue2.Glue2.class);
            if (glue2.getComputingService().size() > 0) {
                // handle description of scheduler
            }
            if (glue2.getComputingShare().size() > 0) {
                // handle descriptions of queues
            }
            if (glue2.getExecutionEnvironment().size() > 0) {
                // handle descriptions of node types
            }
        } catch (com.fasterxml.jackson.core.JsonParseException e) {
            // handle parse exception
        }
    }
}
```

Figure 5: Example code for receiving GLUE v2.0 documents via RabbitMQ.