4 DCI Bridge: Executing WS-PGRADE Workflows in Distributed Computing Infrastructures

Miklos Kozlovszky, Krisztián Karóczkai, István Márton, Péter Kacsuk, and Tibor Gottdank

Abstract. Solving distributed computing infrastructure (DCI) incompatibility issues in a generic way is a challenging and complex task. Gateways and workflow management systems are often tightly bound to some limited number of supported DCIs. To enable gateways access to many different DCIs and to solve DCI compatibility among the very different workflow management systems, we have developed a generic solution, the DCI Bridge. In this chapter we describe its internal architecture, provide usage scenario and show how the DCI Bridge resolves interoperability issues between various middleware-based DCIs. We also provide insight about the capabilities of the realized system. The generic DCI Bridge service seamlessly enables the execution of workflows (and jobs) on major DCI platforms such as ARC, Globus, gLite, UNICORE, SGE, PBS, as well as web services or clouds.

4.1 Introduction

In most cases, gateways and workflow management systems (and therefore their workflows) are tightly bound to some small number of specific distributed computing infrastructures (DCIs), and effort is required to enable additional DCI support. As a result, solving workflow management systems' DCI incompatibility, or their workflow interoperability [Krefting/2011] issues are very challenging and complex tasks. In this chapter we show a concept of how to enable generic DCI compatibility, which is feasible for many major gateways and grid workflow management systems (such as ASKALON [Duan/2005], MOTEUR [Glatard/2008], WS-PGRADE/gUSE [Kacsuk/2012], etc.) on workflow level (and also on the job level). To enable DCI compatibility among the different gateways and workflow management systems, we have developed the DCI Bridge, which is also one of the main components of the so-called fine-grained interoperability approach (FGI) developed by the SHIWA (SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs) project [Plankensteiner/2013]. In this chapter we target the generic DCI Bridge service component and describe its internal architecture, provide usage scenarios, and show how the DCI Bridge can resolve

the DCI interoperability issues between various middleware types (e.g., between gLite, ARC, and UNICORE).

4.2 The Generic Concept of DCI Bridge

In the WS-PGRADE/gUSE portal framework, the DCI Bridge provides flexible and versatile access to all the important applied DCIs within Europe. In the previous versions of gUSE, as many submitters had to be developed as there were different DCIs to be supported. The DCI Bridge component originally was developed to support only SHIWA's FGI solution; however, later on it turned out that it is useful for any OGSA Basic Execution Service 1.0 (BES) enabled workflow management system to solve their DCI interoperability issues. The DCI Bridge is a web service based application, which provides standard access to various distributed computing infrastructures such as: service/desktop grids, clusters, clouds and web service-based computational resources (it connects through its DCI plugins to the external DCI resources). The main advantage of using the DCI Bridge as a web application component of workflow management systems is that it enables the workflow management systems access to various DCIs using the same well-defined communication interface (Fig. 4.1). When a user submits a workflow, its job components can be submitted transparently into the various DCI systems using the OGSA Basic Execution Service 1.0 (BES) interface. As a result, the access protocol and all the technical details of the various DCI systems are completely hidden behind the BES interface. The standardized job description language of BES is JSDL [Anjomshoaa/2006]. Additionally, DCI Bridge grants access to a MetaBroker service called Generic Metabroker Service (GMBS) [Kertesz/2010]. This service acts as a broker among different types of DCIs: upon user request it selects an adequate DCI (and, depending on the DCI, an execution resource as well) for executing the user's job. Just like the DCI Bridge, GMBS accepts JSDL job descriptions and makes use of the DCI Bridge service to actually run the job on the selected DCI.

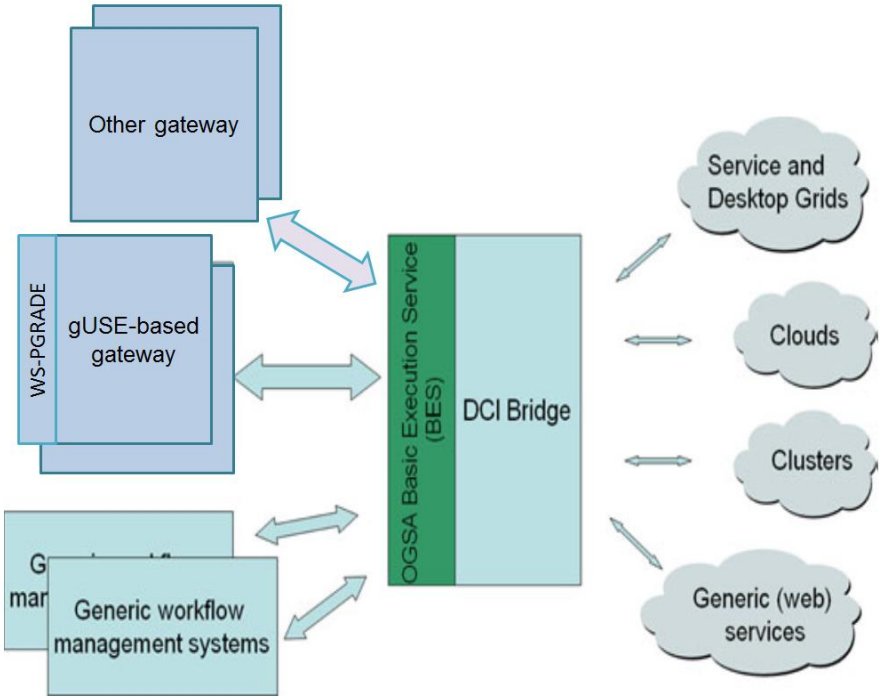

Fig. 4.1 DCI Bridge architecture overview

The DCI Bridge can be used as a stand-alone service [DCIBRIDGE] and also as part of the WS-PGRADE/gUSE gateway framework service set. In the current chapter we focus on the features that are used inside WS-PGRADE/gUSE. In this context WS-PGRADE/gUSE and the DCI Bridge have the following three main important roles and properties:

1. WS-PGRADE provides the user interface where the references to the DCI resources can be defined. Consequently, only those DCI resources are visible during workflow/job configurations that have been defined by the System Administrator of WS-PGRADE.
2. WS-PGRADE contains the default base parameters for remote resources. Consequently, the actual job submissions can be controlled and observed via the DCI Bridge.
3. DCI Bridge can provide an interface for external brokering services. As a unique and single interface the DCI Bridge is an ideal insertion point for "meta-brokering". Meta-brokering means to call a special service which performs the eventual late binding of jobs to resources, upon a smart algorithm trying to match “free” resources and jobs.

4.2.1 DCI Compatibility

A wide range of different middleware types are supported by the DCI Bridge (Table 4.1). The number of supported DCI is growing constantly. So far the following DCI types are supported: service grids (gLite, GT2, GT4, GT5, ARC, UNICORE), clusters (PBS, LSF, SGE, MOAB), BOINC desktop grids, clouds (EC2, CloudBroker), web services, Google App Engine, GEMLCA, and local resources. As future work the connection to XSEDE grid and DIRAC based DCIs is planned.

Table 4.1 DCIs supported by the DCI Bridge

No.	Middleware	Plugin name	Name	Type	Links
1	Arc	Arc	Advance Resource Connector	Service Grid middleware	http://www.nordugrid.org/arc/
2	Boinc	Boinc	Berkeley Open Infrastructure for Network Computing	Desktop Grid Middlware	http://boinc.berkeley.edu/
3	Cloudbroker	Cloudbroker	CloudBroker Platform	Cloud gateway	http://cloudbroker.com/solutions/
4	EDGI	EDGI	European Desktop Grid Initiative	Desktop Grid Gateway	http://edgi-project.eu/
5	GAE	GAE	Google App Engine	Gateway	https://developers.google.com/appengine/?csw=1
6	GAE - HTTP	HTTP	Google App Engine	Gateway	https://developers.google.com/appengine/?csw=1
7	GAE - REST	REST	Google App Engine	Gateway	https://developers.google.com/appengine/?csw=1
8	GBAC	GBAC	Generic BOINC Application Client	Desktop Grid Gateway	http://gbac.sourceforge.net/
9	Gemlca	Gemlca	Grid Execution Management for Legacy Code Archite	Service Grid Gateway	http://dev.globus.org/wiki/Incubator/GEMLCA
10	Glite	Glite	Lightweight Middleware for Grid Computing	Service Grid Middleware	http://en.wikipedia.org/wiki/GLite
11	GT2	GT2	Globus Toolkit 2	Service Grid Middleware	http://toolkit.globus.org/toolkit/
12	GT4	GT4	Globus Toolkit 4	Service Grid Middleware	http://toolkit.globus.org/toolkit/
13	GT5	GT5	Globus Toolkit 5	Service Grid Middleware	http://toolkit.globus.org/toolkit/
15	EC2	EC2 DirectCloud	Amazon Elastic Compute Cloud	Cloud middleware	http://aws.amazon.com/ec2/
16	LSF	LSF	Platform Load Sharing Facility	Workload management p	http://en.wikipedia.org/wiki/Platform_LSF
17	Moab	Moab	Moab	Cluster workload manage	http://en.wikipedia.org/wiki/Moab_Cluster_Suite
18	PBS	PBS	Portable Batch System	Job scheduler for cluster	http://en.wikipedia.org/wiki/Portable_Batch_System
20	SGE	SGE	Sun Grid Engine	Service Grid Middleware	http://en.wikipedia.org/wiki/Oracle_Grid_Engine
21	Unicore	Unicore	Uniform Interface to Computing Resources	Service Grid Middleware	http://www.unicore.eu/
14	Local	Local	-	-	-
19	WebService	WebService	-	-	-

4.2.2 Cascading DCI Bridges

Several DCI Bridge objects may coexist within the same gUSE environment. These DCI Bridges can be connected together in a treelike way (Fig. 4.2). Notice that this cascading of DCI Bridge services is possible because they all use the JSDL job submission description which contains information on the resource and target storages required for input and output file staging. This feature of the DCI Bridge is exploited in creating a solution for the direct connection of gUSE with clouds (Sect. 4.5). The first connected DCI Bridge (object) is distinguished, and we call it the Master DCI Bridge (this is the root of the DCI Bridge tree). It can be configured via the Information portlet (Resource tab) within the WS-PGRADE portal. All other eventual DCI Bridges must be configured by using their explicit URLs. All visible resources should be presented by their names within the Master DCI Bridge. This rather restrictive condition has a very simple cause: the front end of the WS-PGRADE/gUSE system is not able (for the time being) to walk over the internal chains of the connected DCI Bridge objects, i.e., it “sees” only the Master DCI Bridge. However, the real access properties of the mentioned resources may be described in different DCI Bridges.

The use of several DCI Bridges can provide seamless load-balancing between multiple DCIs. The load-balancing is implemented through the Master DCI Bridge, which can distribute the incoming jobs in a treelike arrangement to other DCI Bridges. The outline of the load-balancing scenario is shown in Fig. 4.2. The core gUSE services are aware of only one DCI Bridge installation. Although this DCI Bridge service is not connected to any DCI, it may forward the jobs it receives to other DCI Bridge deployments as they are using the same submission interface and job description language. In this way the central DCI Bridge service may distribute the incoming jobs among the other services of which it is aware. After the jobs are distributed, they have the possibility to report job status back to the central gUSE services using the callback JSDL extension we described at the beginning of this section.

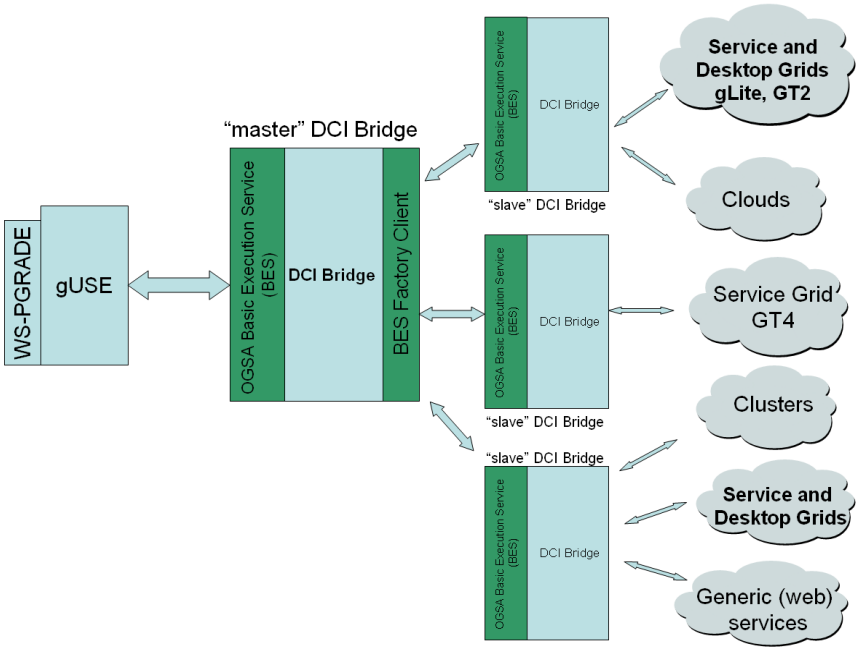

Fig. 4.1 Treelike connection of DCI Bridges to provide load-balancing between DCIs

4.3 Internal Architecture and Main Components of the DCI Bridge

The DCI Bridge contains a set of components as shown in Fig. 4.3. All components of the DCI Bridge must run within a generic web container (such as Tomcat or Glassfish).

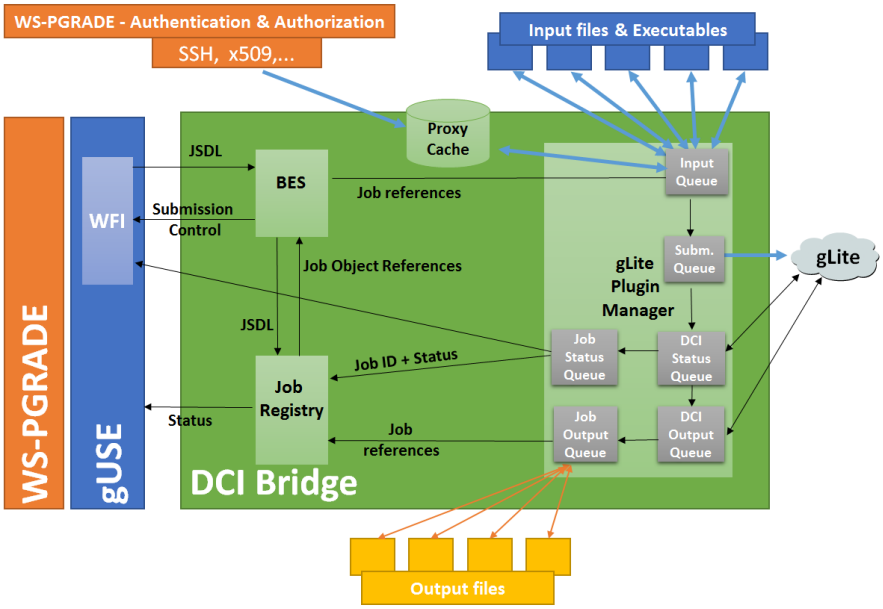

Fig. 4.2 Internal architecture of the DCI Bridge

The DCI Bridge can be called by the BES Web Services Description Language (WSDL) and it executes the operations defined by the Open Grid Services Architecture (OGSA) Basic Execution Service (BES) 1.0 specification on different grid/cloud/service-based middleware. The separate DCIs can be handled by plug-ins and their numbers can be increased without any restriction. Main components of the DCI Bridge are:

- BES Service
- Job Registry
- Proxy Cache
- DCI Plugin Manager (Fig 4.2 shows the gLite plugin manager as an example)
- Job Temporary Directory (not shown in Fig. 4.2 to simplify the figure)

The plugin manager is the main service of the DCI Bridge; it contains four types of queues: Input Queue, Submit Queue, Status Queue (DCI and Job Status queues) and Output Queue (DCI and Job Output queues). All of them are processed by several threads that are created dynamically according to the load of the DCI Bridge. The DCI Bridge administrator can define the maximum number of queue-handling threads. There are as many Plugin Manager instances as there are DCIs connected to the given gateway. For example, if a gateway is connected to a gLite and a BOINC grid, then it has two plugin manager instances.

The BES service of DCI Bridge accepts standardized JSDL job description documents, which are based on well-defined XML schemes, and which contain in-

formation about the job inputs, binaries, runtime settings, and output locations. The core JSDL itself is not powerful enough to fit all needs, but various extensions are used to overcome this issue. The DCI Bridge makes use of three legacy extensions: one for defining execution resources, one for proxy service, and one for callback service access. The execution resource extension is needed both for the core DCI Bridge in order to define specific execution resource needs and for the metabroker service. The proxy service extension is needed for security management and credential handling (see below). The callback service extension is needed if status change callback functionality is needed: the DCI Bridge will initiate a call to the service specified in the extension upon every job status change.

Incoming BES jobs are placed in the job registry and their references are passed to the other DCI Bridge services. The job's reference is immediately placed into the input queue of the corresponding Plugin Manager. The processing of a job consists of the following steps:

- Security management
- Input file management
- Submission
- Status inquiry
- Transfer status information to gUSE workflow interpreter (WFI)
- Output inquiry
- Result files upload to gUSE storage

4.3.1 Security Management – Credential Handling

User credentials (proxies or SAML assertions) are handled by the proxy cache using the JSDL extension that identifies the virtual organization (VO) where the job should be executed. Based on the user and VO identifiers, the proxy cache service of DCI Bridge downloads the authentication file from the certificate service of WS-PGRADE (see Chap. 6). Depending on the type of the authentication file, the following actions will be executed:

- User login/password: no action is required.
- SAML: Expiration time is checked.
- X509: Expiration time is checked, and if it is for gLite then the Virtual Organization Membership service (VOMS) extension needed for the VO is added based on the information in JSDL file.
- Secure Shell (SSH) key: no action is required.

If any problem occurs an error message is placed in the error log. If the authentication file check is successful, then the processed authentication file is placed into the proxy cache and the job receives this information. Storing the processed authentication file in the proxy cache reduces the time of further authentication checks of new jobs having the same user–VO pair in their JSDL.

4.3.2 Input File Management

Input file management can be done at two places. The generic solution is to do it inside the DCI Bridge, but for optimization purposes this activity can be postponed and executed by special wrappers in the target DCI resources. In the latter case, the DCI Bridge administrator should set this optimization parameter provided that the required firewall conditions are met. In this case, there is no need for file management activities in the DCI Bridge. This is also the case when the user specified remote file access for the job.

In the generic case, JSDL contains the description of the required input files, and the input queue service inside the plugin manager downloads the input files to the job temporary directory, either from the storage service of gUSE or from an HTTP server. At this point the job is prepared to be submitted to the target DCI, and hence the job is placed into the submit queue of the plugin manager.

4.3.3 Job Submission and Execution

Jobs in the submit queue are already prepared for the target DCI with all the information needed to submit there, and hence the submission is a straightforward activity by the DCI-specific plugin thread. Meanwhile, as the job is executed in the target DCI, the thread handling the DCI status queue periodically polls the status of the job execution and places the status information into the queue. Whenever the status is changed in the DCI status queue it is written back to the job status queue and then passed back to the gUSE workflow interpreter. Once the status is success or failed, the output file management step is initiated, where the output files of the job execution are collected into the DCI output queue.

4.3.4 Output File Management

Output file management can also be done at two places. The generic solution is to do it inside the DCI Bridge, but for optimization purposes this activity can be executed by special wrappers in the target resources. In the latter case, the DCI Bridge administrator should set the same optimization parameter that was mentioned in the case of the input file management. In this case there is no need for file management activities in the DCI Bridge. This is also the case when the user specifies remote file access for the output of the job.

In the generic case JSDL describes the place of the output files, and the job output queue service inside the plugin manager uploads the output files to the storage service of gUSE or to an HTTP server.

4.4 Configuration of the DCI Bridge

In order to control the operation of the DCI Bridge and to configure the connected resources the *DCI Bridge administration interface* has been introduced. The administration interface has two main parts (Fig. 4.4):

1. The base part, where the configuration affects the generic settings of DCI Bridge, and
2. The middleware-specific part, where the settings are unique for each individual middleware.

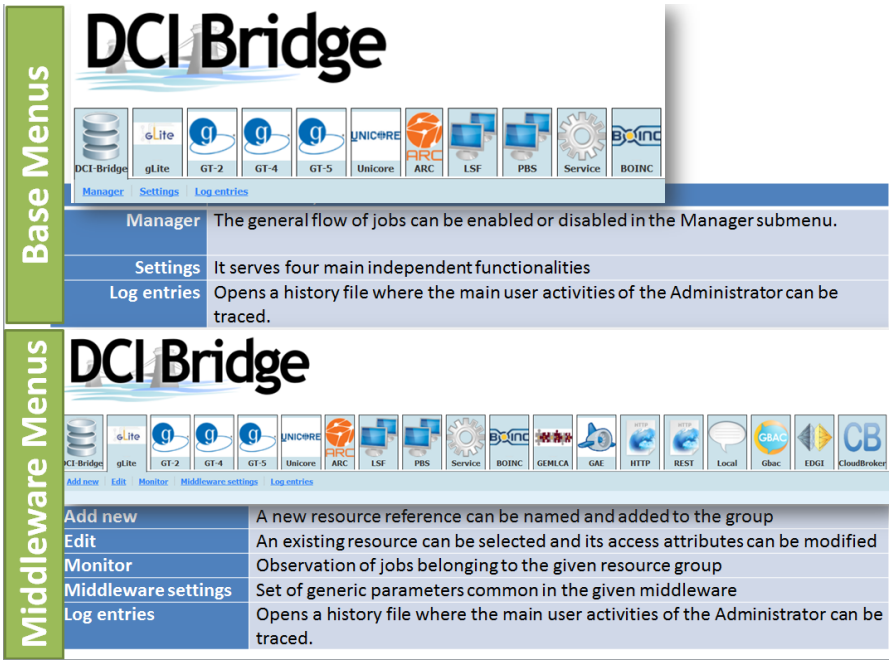

Fig. 4.3 The two main menus of the DCI Bridge administrator interface

1. The Base menus (*Manager*, *Settings*, *Log entries*) – from the *DCI Bridge* tab – are responsible for
    a. Managing DCI Bridge generic operation by enabling or disabling the flow of jobs between the workflow interpreter and DCI Bridge (in *Manager* menu). In case of troubleshooting, it is useful to disable the DCI Bridge to accept jobs from the workflow interpreter. In this case the DCI Bridge is still able to accept the status and outputs of the jobs already sent to the connected DCIs. As a result, the gateway administrator can check how the previously allocated jobs are exe-

cuted in their target DCIs. In fact, these status information and result files are transferred back to the gUSE storage.

b. Setting the work directory of DCI Bridge where all the temporary files of DCI Bridge are stored. Its default value is the CATALINA_HOME of the web container where the DCI Bridge as a web application can be found (in *Settings* menu).
c. Providing the opportunity to give an external URL of the DCI Bridge for middleware plugins to speed up the status sending process. This is particularly useful in the case of gLite resources, where it often happens that the job is finished but it takes a long time to get the status and result via the gLite middleware. In this case a special job wrapper script can immediately send back the status and result files via the provided external URL of the DCI Bridge when the job is finished on the gLite resource.
d. Debugging and error handling. When the *Debug mode* is set to *Enabled*, then the temporary job directory will not be deleted after job execution, and hence the DCI Bridge administrator can search for a given error. Since this solution significantly loads the local storage it is not recommended to be enabled in production systems (in the *Settings* menu).
e. Access a log file where the main user activities can be traced by the DCI Bridge administrator (in the *Log entries* menu).

2. The middleware menus are responsible for separating settings of all supported and connected middleware. Every middleware setting belongs to a single tab, and there are five submenus within each middleware tab (Fig. 4.4):
    a. The *Add new* menu is responsible for adding new resource references for the selected DCI type.
    b. The *Edit* menu is responsible for the selection or modification of an existing resource.
    c. The *Monitor*, *Middleware settings* and *Log entries* menus are generic. The *Monitor* menu opens the Job History Database, where only those jobs are visible that belong to the selected type of middleware. The *Middleware settings* function defines the middleware plugin details (see Figure 5). Very importantly, this is the place where the required authentication proxy types should be defined. From the point of view of performance, the number of threads used inside a certain plugin to handle the internal queues can be defined here. The *Log entries* menu shows only those events of the general log that are related to the given middleware.

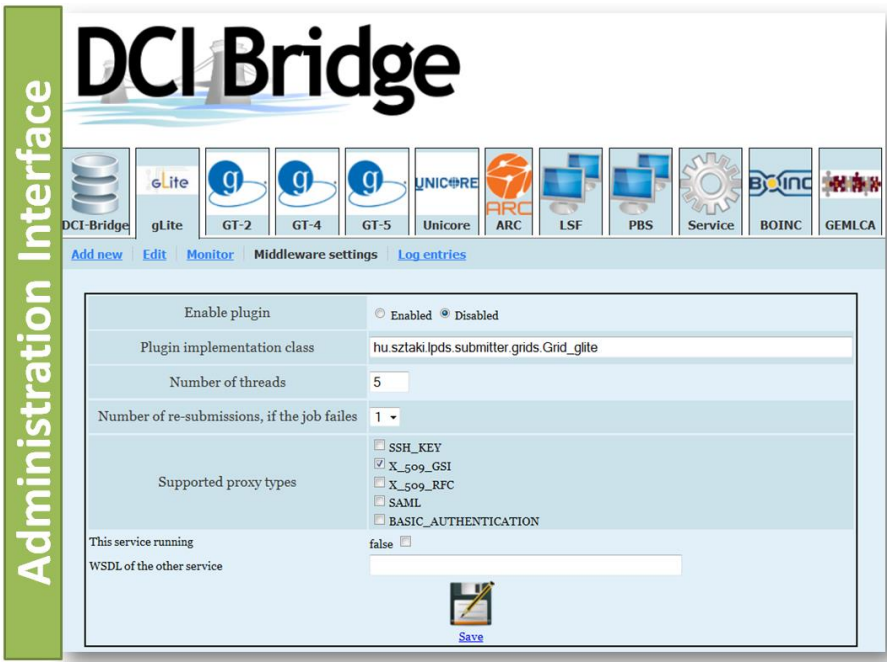

Fig. 4.4 A sample view of the DCI Bridge administration interface

4.5 Direct Cloud Access via the DCI Bridge

4.5.1 Concept of Direct Cloud Access

The main goal of the direct cloud solution is to access directly cloud infrastructure (without using CloudBroker Platform) and launch in the cloud any type of services/jobs defined previously in WS-PGRADE workflows. The direct cloud solution works according to a Master/Slave arrangement of DCI Bridges (Sect. 4.2.2) and does not use the CloudBroker Platform service to access cloud infrastructure. The direct cloud access is based on the capability of creating distributed DCI Bridge deployment and on job forwarding from master to slave DCI Bridge(s). The master DCI Bridge that directly connects to gUSE, forwards jobs through Amazon Elastic Compute Cloud (EC2)-based front-end cloud service to the slave DCI Bridge located in the cloud. Technically, the master DCI Bridge starts the virtual machine (VM) via EC2-based service. The started VM contains a Slave DCI Bridge that is configured to run jobs via the local submission facility, i.e., in the VM where the slave DCI Bridge runs. The VM containing the slave DCI Bridge should previously be created and saved as an image in the cloud repository (Fig. 4.6). This solution can be applied on all Amazon EC2-compatible cloud services.

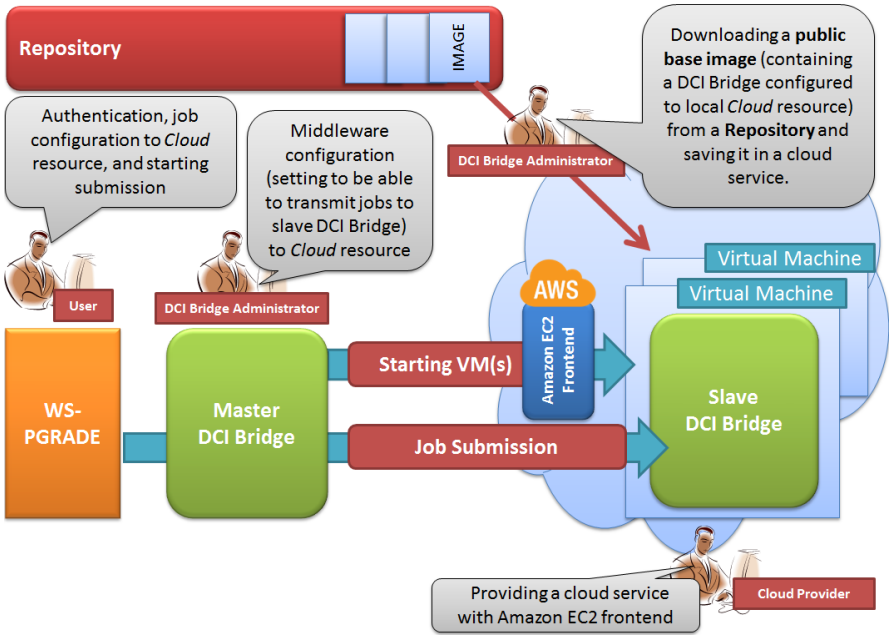

Fig. 4.5 The roles and processes of the direct cloud access solution

The direct cloud access process contains the following tasks and roles:

- Task 1: The DCI Bridge administrator downloads a public base image containing a properly configured DCI Bridge (this will be the slave DCI Bridge) from a corresponding repository. This image will be saved in the target cloud environment. (The cloud service provided by the cloud provider must contain an Amazon EC2 front-end).
- Task 2: The DCI Bridge administrator properly configures the master (root) DCI Bridge (which connects to gUSE).
- Task 3: The user obtains an account from the Cloud Provider to the cloud where the image was imported from the Repository (the Cloud Provider can provide information about the exact way to get a cloud account). From this point, the user can use the WS-PGRADE portal for job submission to the target cloud.

4.5.2 Scalability

The master DCI Bridge can start more VMs in the connected cloud service. There is a parameter by which the gateway system administrator can define the maximum number of slave DCI Bridge VMs that can be started in parallel in the cloud. Notice that in this way, parameter sweep applications can run in parallel on many cloud resources. Moreover, the slave DCI Bridge images can be started in

parallel in several connected clouds. As a result, this solution gives the same multi-cloud functionality that is provided by CloudBroker as discussed in Chap. 7.

4.5.3 Current Restrictions

Direct cloud access from the network point of view does not need any contextualization. It operates on publicly accessible (not private) IP addresses. The main reason for this is that the portal calls the virtual machine (VM) by an IP address that comes from the cloud service, and the VM doesn't call back to the portal. According to this, the currently available gateway–cloud communication directions are: private gateway --> private cloud, private gateway --> public cloud, public gateway --> public cloud.

However, the situation when the gateway and its master DCI Bridges are on public networks, while the slave DCI Bridges are on private network clouds require a special workaround (e.g., dynamic DNS usage).

The current direct cloud solution can be used in clouds that

- implement the EC2 protocol
- contain Kernel-based Virtual Machine (KVM) hypervisor.

The direct cloud access supports all OpenNebula-based and OpenStack-based clouds that fit to these requirements above. However, the current image is not usable for VMware-based or Xen-based hypervisors.

Robot certification is currently not available for direct cloud job submissions. In the near future this restriction will be eliminated by a new release of WS-PGRADE/gUSE.

4.6 Performance

We have done several performance tests to systematically check how well the DCI Bridge works under heavy loads. Our test environment consists of a test application written in Java, script languages, and the DCI Bridge itself. Our test applications are able to automatically send thousands of jobs (with various job parameters (e.g., job owner, job type) in high frequency to measure the performance metrics and the job handling capacity of the DCI Bridge.

From the performance point of view one of the most important DCI Bridge components is the BES Factory. During our first example performance test we sent 9 samples of 1000 configured single jobs with predefined amounts of inputs into the DCI Bridge. According to our assumptions, the amount of input influences the performance parameter because each input should be retrieved by the input queue before submitting the job into the targeted DCI. We have launched our tests with various input sizes (zero, 10, 20, 30, 40, 50, 100, 500, and 1000 inputs). All the other job and input parameters were similarly configured. The starting phase of the DCI Bridge services (due to the network topology and services) requires longer processing time, so we manually eliminated this transient period

from our performance results. As can be seen from Fig. 4.7, (please note that the horizontal axis is not linear), BESFactory service scales smoothly, thus the increased number of inputs causes only about a linear processing time increase.

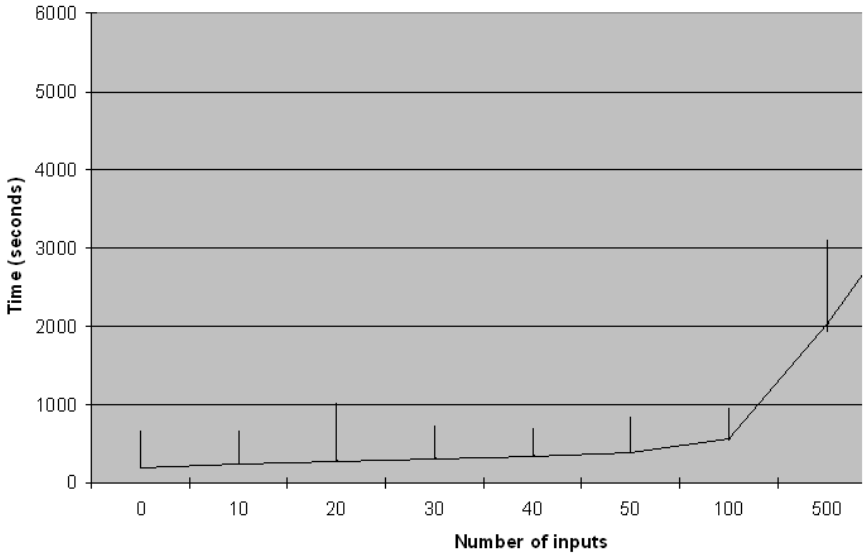

Fig. 4.6 BESFactory service performance test results

Our second example performance test focused on how effectively the load balancing algorithm used the pool of slave DCI Bridges that can be started in cloud infrastructure.

We used a complex synthetic workflow to set up this testing environment: the workflow was built up by consecutively connected jobs to create the testing environment (10x0 sec job, 1x10 min job, 100x0 sec jobs, 1x10 min jobs, 2x0 sec jobs, 1x1 min job, 100x0 sec jobs, 1000x1 min jobs, 1x0 sec job). We have measured how many jobs ran on the DCI Bridges and what their wall clock times were (execution time + submission time). As can be seen from Fig. 4.8, the load was equally distributed among the slave DCI Bridges. On the horizontal axis we show the elapsed time, and on the vertical axis the number of the allocated jobs for the various slave DCI Bridges. All VMs are depicted with a different line.

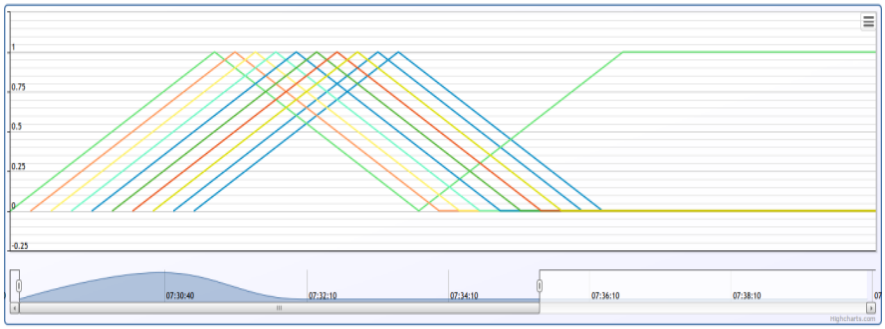

Figure 4.7 DCI Bridge load balancing test

4.7 Conclusions and Future Works

In this chapter we have given an overview of the DCI Bridge approach, which enables seamless DCI access in a generic way. The DCI Bridge is already a proven concept. It is based on the standard OGSA BES interface, and as such can be used with many major grid workflow management systems (such as ASKALON, MOTEUR, WS-PGRADE/gUSE, and others) as was demonstrated in the EU FP7 SHIWA project. We have described its internal architecture and provided information on how its components work together. According to our tests, the DCI Bridge implementation is able to successfully resolve DCI compatibility issues in an effective way. The implemented DCI Bridge solution was used successfully in numerous projects (SHIWA, SCI-BUS, HP-SEE, etc.) as an internal service at the back-end to resolve the DCI interoperability issues between various middleware types. The modular, plugin-like internal architecture enables the DCI Bridge service to be extended easily with new plugins, and hence to provide access for further DCIs. As a result, various communities have already developed new plugins for the DCI Bridge:

- University of Tubingen has developed the MOAB plugin
- Ruđer Bošković Institute has developed the SGE plugin

These plugins are now part of the recent WS-PGRADE/gUSE releases. As future work we are planning to:

- Extend the capabilities of the DCI Bridge with additional middleware support (for example, XSEDE and DIRAC grid plugins);
- Enable seamless data transport between various storage architectures (a collaboration between the DCI Bridge and the Data Avenue services);
- Support advanced and effective brokering and scheduling services;
- Enable the usage of the public gateway --> private cloud connection of WS-PGRADE gUSE and clouds;
- Enable robot certificates in the direct cloud access mechanism of DCI Bridge.