Computer Science • 16 (4) 2015 http://dx.doi.org/10.7494/csci.2015.16.4.373

John Walsh
Jonathan Dukes
Gabriele Pierantoni
Brian Coghlan

OVERVIEW AND EVALUATION OF CONCEPTUAL STRATEGIES FOR ACCESSING CPU-DEPENDENT EXECUTION RESOURCES IN GRID INFRASTRUCTURES

Abstract *The emergence of many-core and massively-parallel computational accelerators (e.g., GPGPUs) has led to user demand for such resources in grid infrastructures. A widely adopted approach for discovering and accessing such resources has, however, yet to emerge. GPGPUs are an example of a larger class of computational resources, characterized in part by dependence on an allocated CPU. This paper terms such resources "CPU-Dependent Execution Resources" (CDERs). Five conceptual strategies for discovering and accessing CDERs are described and evaluated against key criteria, and all five strategies are compliant with GLUE 1.3, GLUE 2.0, or both. From this evaluation, two of the presented strategies clearly emerge as providing the greatest flexibility for publishing both static and dynamic CDER information and identifying CDERs that satisfy specific job requirements. Furthermore, a two-phase approach to job-submission is proposed for those jobs requiring access to CDERs. The approach is compatible with existing grid services. Examples are provided to illustrate job submission under each strategy.*

Keywords generic resources, accelerators, EGI, Grid Integration, GLUE Schema

Citation Computer Science 16 (4) 2015: 373–393

1. Introduction

Since its conception, grid computing has focused on a "single program/single CPU" execution model. For the past decade, however, the exponential growth of CPU speed and processing power has plateaued [8] [17], and this has generated many questions about the future of computational-based scientific research using this single program/single CPU approach. Support for CPU-based parallel execution frameworks (such as OpenMP and MPI) has become commonplace in grid infrastructures [7]. The emergence of many-core and massively-parallel computational accelerators (e.g., GPGPUs) has, however, led to user demand for access to these resources in grid infrastructures [21]. At present, the integration of these resources into Grids (such as EGI) is 'ad-hoc' in nature, with no widely accepted mechanisms for discovery or access.

The inflexible nature of current grid discovery and access mechanisms presents a challenge to the integration of a diverse range of computational resources in existing grid infrastructures. In this paper, a flexible, dynamic approach to the integration of new (and yet-to-be-conceived) resources into existing grid infrastructures is proposed. The computational resources considered are characterized by specific properties: (a) access to the resource requires a CPU and is provisioned by a specific request to the Local Resource Management System (LRMS), (b) the resources are finite in number (limited job slots), (c) the resource is bound to a specific machine (node), and (d) the user perceives that they have exclusive access to the resource. A computational resource with these characteristics is considered to be a *CPU-Dependent Execution Resource* (CDER). This definition is intended to exclude applications or software (these are already facilitated) but include hardware resources such as GPGPUs, FPGAs, and hardware accelerators or software that may be "node-locked" due to licensing restrictions.

The proposed approach focuses on Grids that use the Open Grid Forum (OGF) Grid Laboratory for a Uniform Environment (GLUE) standards to publish information about the state of their resources and services. Although the GLUE standards (and, in particular, GLUE 2.0) facilitate the publication of "extended" information beyond the core specification, there is no standard practice for publishing information relating to CDERs using the GLUE schema.

Several conceptual strategies for discovering and accessing CDERs are considered. The strategies are evaluated against a set of criteria that capture, for example, whether the strategy allows key information about a CDER (e.g., resource type, capacity, or capability) to be gathered and published locally by the provider of the resource and then discovered and utilized globally by both users and grid services.

The structure of the paper is as follows: Section 2 presents an overview of the basics of grid computing. In particular, it focuses on Grids based on the Open Grid Forum (OGF) GLUE information model; why this model is important for large-scale computational-science; and why some CDER resources are likely to become increasingly important as computational-sciences become more dependent on massively-parallel computing. Section 3 describes five conceptual strategies for discovering and accessing CDERs. These strategies are evaluated against a number of criteria in Section 4. Section 5 reviews related and similar work. Finally, Section 6 summarizes the findings of this work and discusses future work addressing CDER integration into grid environments.

2. Background

A Grid is a distributed collection of computational and storage resources where (a) each resource is controlled and managed solely and independently by its owner or *resource-provider* (for example, a University, research center, company, or private individual) and (b) each resource-provider has some level of control over how the resource is accessed and used. This definition is sufficiently general to include both large-scale Grids, such as the European Grid Infrastructure (EGI[1]) and the Open Science Grid (OSG[2]), as well as "compute-cycle" volunteer (donation) systems such as BOINC [1]. However, the key differences that distinguish the EGI and OSG from other types of Grids is that these are specialist Grids that use common standards to provide interoperable security infrastructures (that help control access to the resources) and use a common information model, the Grid Laboratory for a Uniform Environment (GLUE), to describe the state of the resources on each Grid. Moreover, the role of a common information model is paramount in aiding the grid users to locate and select suitable resources according to their needs. An implementation of an information model is called a *Grid Information Service* (GIS). In the context of a Grid, these can simply be called the *Information System.* The ensemble of software and services that Grids are built upon is called the *Grid Middleware*, or in this paper, *middleware.* Different grid infrastructures can support different middlewares. For example, the EGI can use the gLite, UNICORE, and ARC middlewares. These three middlewares are collectively known as the *Unified Middleware Distribution* (UMD) [6]. The OSG uses the Globus[3] middleware.

2.1. The OGF GLUE Schemas: conceptual and concrete models

There are currently two major versions of the GLUE specification in common use by global grid infrastructures, such as the EGI, OSG, and LCG[4]. These are known as the GLUE 1.3 [12] and GLUE 2.0 [13] Schemas. It is important to note that these specifications define conceptual models, where the models show how entities (resources, services, security policies, etc.) in a Grid relate to one another, and which properties each resource should (mandatory) or may (non-mandatory) possess. Furthermore, the conceptual model is independent of the concrete data-format used

[1] `http://www.egi.eu`

[2] `http://www.opensciencegrid.org`

[3] `https://www.globus.org/`

[4] `http://wlcg.web.cern.ch/`

by specific technologies. Current concrete data-formats include the LDAP Schema, XML Schema, SQL, and JSON.

The first GLUE specification (GLUE 1.0) grew out of a need for many early grid infrastructures and middleware projects, such as DataTag, The European Data Grid (EDG), iVDGL, LCG, and Globus, to converge on a consistent description of globally distributed grid resources and services [4]. The Grids already conformed to a common security model – the Globus Security Infrastructure (GSI) – but some of the Grids published differing information to describe the available resources and services. Convergence on a common resource specification would greatly improve grid-interoperability – an objective that needed to be fulfilled in order to solve many grand-challenge problems, such as confirming the existence of the Higgs boson [2].

The initial specification was proposed in September 2002; however, further specifications followed in April 2003 (GLUE 1.1), February 2005 (GLUE 1.2), and October 2006 (GLUE 1.3) in order to solve numerous problems with the specification itself or to enhance the specification. Each of these incremental changes were required to be fully backward compatible. This restriction was deemed to be a major constraint, as it limited the evolution of both the schema and the grid-middleware that used it [4].

It should be noted that each new revision of the GLUE schema requires changes to each of the grid-middlewares. As these middlewares are intended to provide robust "production use" of the grid infrastructures, there must be an assurance that disruptions to services are kept to a minimum. Consequently, changes to both the schemas and middlewares need to be rigorously tested. This testing and deployment process is both costly in effort and time-consuming. Furthermore, changes may also affect grid users if they need to adapt existing applications. Despite the ratification of GLUE 2.0 in 2009, it has yet to fully replace GLUE 1.3 as of late-2014. A major motivation behind the evaluation of the conceptual strategies described in this work is to identify the provisions that exist in GLUE schemas that would facilitate a more flexible and dynamic approach to the integration of new resources, without requiring changes to the schemas.

The GLUE is used to describe the conceptual model of grid "entities," their key properties, and their relationships with other entities (if any exist). The relationship between entities often takes the form of a "child-parent" dependency. For instance, a Grid is composed of a set of Sites (resource-providers); each Site may provide a set of one or more services (computational, storage, security, grid job orchestration, etc.). There may be several instances of these grid services; for example, a Site may have several LRMS's, each managing their own collections of homogeneous nodes (clusters) that execute grid jobs. The nodes are further classified as belonging to one or more sub-collections (queues), and these queues will have their own time and memory limits and access polices. Furthermore, the LRMS may also implement a policy to ensure that collections of grid users (Virtual Organisations) have guaranteed access to these resources over a finite period of time (Fair Share Allocation).

Nodes in a cluster are considered to be homogeneous, in that properties such as the CPU model, speed, and memory allocated to each node should be same. However, the practice is that the values chosen for publication are often baseline or average representations – as nodes added into a cluster at a later stage may be more powerful.

The presentation of GLUE in a concrete format such as LDAP or XML is known as a *rendering*. The GIS is a service that generates, stores, updates, and allows querying of rendered GLUE data. Furthermore, the GIS is key to discovering resources and enabling other grid services, such as the Workload Management System (WMS), to orchestrate user jobs. One of the most-prevalent implementations of an information system is called the Berkeley Database Information Index (BDII). Both EGI and OSG use the BDII ([3], Sec. 3.3.5) Grid Information Service. This is an implementation of a hierarchical grid information system model with three BDII *types* and a set of *Information Providers* that generate information about the grid services, their resources, and their security-access policies. Figure 1 illustrates the hierarchical structure of the BDII. The Resource-BDII (lowest BDII level) aggregates the state of a service node by executing a set of Generic Information Provider (GIP) plugins; the Site-BDII accumulates data from all of the Resource-BDIIs belonging to the given Site; and finally, the Top-level BDII aggregates all data coming from the set of Site-BDIIs. Information is pulled from lower levels to higher levels. In this way, the state of any grid service can be determined by querying the Top-Level BDII. EGI requires that the BDII publishes both GLUE 1.3 and GLUE 2.0 formats using the LDAP Data Interchange Format (LDIF).

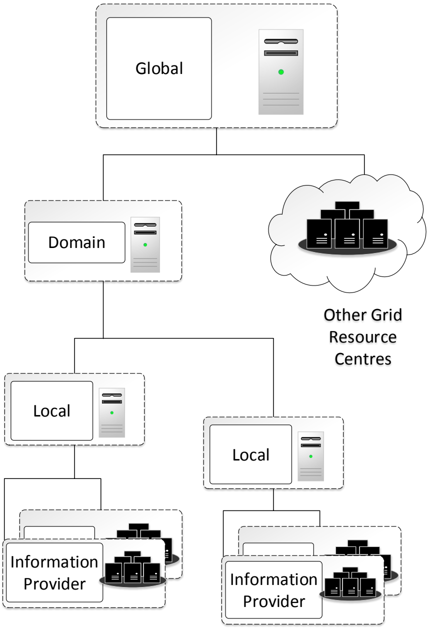

Figure 1. The Grid Information System. GLUE data is pulled from the lowest level (generated by Information Providers) up to the Global level.

The representation of computing resources in the GLUE Schemas allows Sites to publish information about the many high-level aspects of the Computing Service, such as the LRMS that manages the allocation of nodes to grid jobs, how many CPU-cores (or job slots) are available on each node (for multi-core applications), as well as capacity and utilization details.

The developers of GLUE 1.3 note [12] that the full set of features and policies for a given LRMS is much too complex to be represented in a reasonably compact schema. Furthermore, because LRMS implementations have features that vary qualitatively, the schema definition is intended to capture the most-common configurations among the supported LRMS's. The same concerns and considerations also apply to GLUE 2.0. An unintended consequence of this approach is that neither GLUE 1.3 nor GLUE 2.0 provide support for CDERs. This is because (i) CDERs are usually implemented as a *Generic Consumable Resource* in the LRMS, and this was not supported in LRMS scheduling systems such as MAUI; and, (ii) the use of CDERs in grid applications was not common.

2.2. Extending GLUE Entities

Both GLUE 1.3 and GLUE 2.0 schemas provide ways to associate additional data with existing GLUE entities and publish this additional data. These mechanisms correspond to two of the concrete realizations of the conceptual methods presented in Section 3; namely, the *Attribute-Extension* and *Class-Extension* strategies. The main differences are (i) GLUE 1.3 Attribute-Extensions are limited to a few GLUE classes through the *capacity* attribute, whereas under GLUE 2.0, all Classes (except Extensions) can be extended by adding one or more *OtherInfo* attributes. (ii) GLUE 1.3 allows for Services to be extended by using instances of a GlueServiceData Entity; under the LDAP rendering, the Key/Value can be associated with a Service Entity instance by using a *GlueChunkKey* – this extends Service instances only, so there are clear limitations in how this can be used. In contrast, the GLUE 2.0 LDAP rendering allows all object instances to be extended using one or more Extension instances.

3. Conceptual CDER Access Strategies

Several conceptual strategies have been identified that would allow grid jobs to discover and access CDERs on grid infrastructures based on the GLUE schema. Each of the strategies presented describes (i) an approach for publishing information describing a CDER and (ii) a method for using this published information to satisfy the specific CDER requirements of a grid job.

3.1. Strategies

A-Priori Strategy

This strategy does not publish any CDER GLUE data and requires that the grid user has prior knowledge of the specific CDER, its properties, the name of queue used to access it, and any additional access requirements. This is effectively a ‘null-strategy’ and is the only-available CDER access method in the absence any other strategy. It may, however, be a useful approach, for example, when testing the deployment of new CDERs.

When submitting a job using this strategy, the user must specify exactly the grid queue (A *ComputeElement Endpoint*) where the job is to be executed. For example, this method has been used to support the execution of GPGPU applications at selected Sites on the EGI [20].

Named-Queue Strategy

A community-adopted queue-naming convention may be used to indicate that a job requirement is satisfied or that a resource is attainable by using that queue. For example, the queue name suffix `sdj` (Short Deadline Job) has been used to advertise grid queues that support high-priority job execution [9]. Similarly, the suffix gpgpu could be used to imply that GPGPU CDERs are available through a specific queue. Listing 1 demonstrates the specification of a requirement that a job be submitted to a queue with the gpgpu suffix using the gLite/UMD middleware job description language (JDL).

Listing 1. An example Named-Queue grid job specification.

```
[
Type="Job";
JobType="Normal";
Executable = "myScript.sh"; # Script to invoke GPGPU application
StdOutput = "std.out";
StdError = "std.err";
InputSandbox =  {"myScript.sh"};
# Regular expression to match all queue names ending with 'gpgpu
    ↪ '
Requirements = ( RegExp(".*gpgpu$", other.GlueCEUniqueID) );
]
```

Tagged-Environment

More-versatile strategies are possible if the information schema allows arbitrary new information to be published alongside the information describing the existing resources. For example, it is common in grid information schemas that environment tags may be used to advertise that specific software is supported at a Resource Center. They may also be used to advertise hardware configurations (e.g., GLUE 1.3 SoftwareEnvironment tags have been used to advertise the availability of Infiniband networking [16]). This strategy is an implicit mechanism for adding new attribute values, and it may also be used to publish arbitrary information about the CDER.

The Tagged-Environment strategy is widely used on the EGI to support the execution of multi-core applications using MPI-START [7].

A concrete example showing how GPGPU CDERs are accessed is as follows: if a Resource Center publishes the tags shown in Listing 2, then a user requiring Nvidia Kepler GPGPUs can specifically target those Resource Centers by using the *Requirements* expression in Listing 3.

Listing 2. An example GLUE 1.X Tagged-Environment advertising both software (CUDA) and hardware (NVIDIA-KEPLER) capabilities.

```
GlueHostApplicationSoftwareRunTimeEnvironment: CUDA
GlueHostApplicationSoftwareRunTimeEnvironment: NVIDIA-KEPLER
```

Listing 3. Example Tagged-Environment grid job specification requiring the `NVIDIA-KEPPLER` hardware capability.

```
[
...
Requirements = (Member("NVIDIA-KEPLER",other.
    ↪ GlueHostApplicationSoftwareRunTimeEnvironment));}
]
```

Attribute-Extension

Using this strategy, CDERs are explicitly associated with existing grid resources (e.g., worker nodes), whose properties are already captured by the information schema (e.g., ExecutionEnvironment instances). The schema may allow the description of an existing resource to be *internally* extended with new attributes and their associated values. Under GLUE 2.0, Attribute-Extension can be implemented using OtherInfo attributes that can be applied to any GLUE 2.0 Entities. Listing 4 provides a concrete example of the use of this strategy to publish detailed information about a GPGPU CDER. The GPGPU is explicitly associated with an ExecutionEnvironment instance. The information published is both static – describing hardware characteristics – and dynamic – reflecting current capacity and utilization.

The Attribute-Extension strategy has been previously used in a prototype UMD-based testbed to publish GPGPU information in GLUE 2.0 ApplicationEnvironment instances [22]. Current implementations of job orchestration systems (e.g., the UMD WMS) will be unable to process CDER extensions to satisfy job requirements, so an alternative two-phase job submission mechanism was proposed. This will be described in Section 3.2.

Class-Extension

A by-reference alternative to the by-value Tagged-Environment and Attribute-Extension strategies may be used if the information schema allows the information describing existing resources to *externally* reference information describing CDERs.

Listing 4. An example of the publication of static and dynamic GPGPU information by extending GLUE 2.0 Execution Environment instance using OtherInfo attributes.

```
objectClass: GLUE2ExecutionEnvironment
...
GLUE2EntityOtherInfo: GPGPUTotalInstances=32
GLUE2EntityOtherInfo: GPGPUsedInstances=2
GLUE2EntityOtherInfo: GPGPUCUDAComputeCapability=2.1
GLUE2EntityOtherInfo: GPGPUMainMemorySize=1024
GLUE2EntityOtherInfo: GPGPUMP=4
GLUE2EntityOtherInfo: GPGPUCoresPerMP=48
GLUE2EntityOtherInfo: GPGPUCores=192
GLUE2EntityOtherInfo: GPGPUClockSpeed=1660
GLUE2EntityOtherInfo: GPGPUECCSupport=false
GLUE2EntityOtherInfo: GPGPUVendor=Nvidia
GLUE2EntityOtherInfo: GPGPUPerNode=2
```

For example, any GLUE 2.0 Entity may be associated with zero, one, or more instances of the Extension class, each of which contains a single Key/Value pair. These Key/Value pairs may be used to describe the properties of an associated CDER. This is illustrated in Listing 5. Like the Attribute-Extension strategy, this strategy requires the two-phase job-submission mechanism proposed in Section 3.2.

Listing 5. A single GLUE 2.0 Extension instance that is associated with a parent ComputingShare instance. The extension is used to publish the `GPGPUPerNode=2` Key/Value pair.

```
dn: GLUE2ExtensionLocalID=GPU_NVIDIA_P_1,GLUE2ShareID=
   ↪ gpgpu_gputestvo_ce.example.com_ComputingElement,
   ↪ GLUE2ServiceID=ce.example.com_ComputingElement,
   ↪ GLUE2GroupID=resource,o=glue
GLUE2ExtensionLocalID: GPU_NVIDIA_P_1
GLUE2ExtensionKey: GPGPUPerNode
objectClass: GLUE2Extension
GLUE2ExtensionValue: 2
GLUE2ExtensionEntityForeignKey: gpgpu_gputestvo_ce.example.
   ↪ com_ComputingElement
```

Listing 6. An LDAP query expression used to find GLUE 2.0 Extension instances with the `GPGPUPerNode` key.

```
GLUE2ExtensionKey=GPGPUPerNode
```

3.2. Two-Phase Job Submission

The strategies considered in this paper look at the conceptual approach for tackling the publication of CDER-related data. It is intended that the published data be used to help grid users or services identify where the CDERs are deployed, as well as their capabilities, usage, and state. By using the published data, grid jobs can be restricted to matching Sites. However, current implementations of job orchestration systems (such as the Workload Management System) cannot exploit the published CDER data. A *two-phase* approach was developed [22] in a prototype gLite/UMD-based grid infrastructure that used the Attribute-Extension of a GLUE 2.0 ApplicationEnvironment to publish information describing GPGPUs. The same two-phase approach is applied in Section 4.1 to both Attribute-Extension and Class-Extension strategies

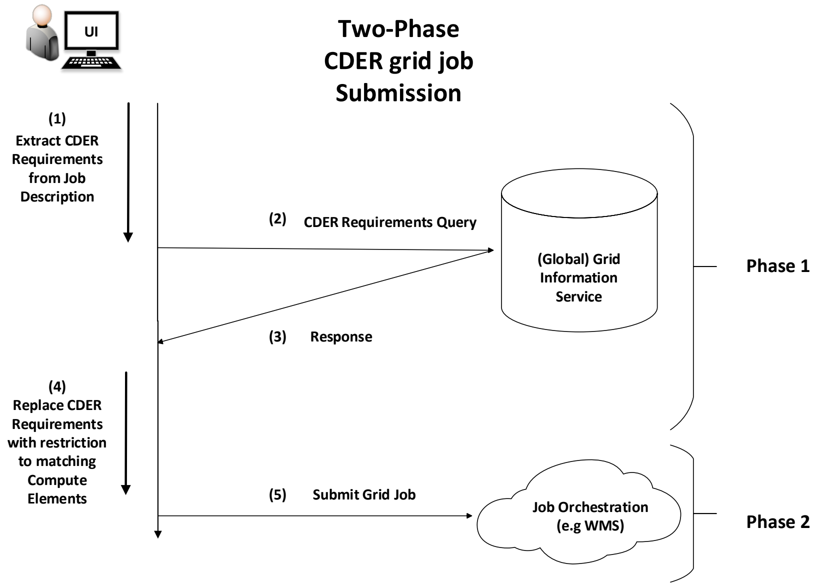
UI
Two-Phase CDER grid job Submission
(1) Extract CDER Requirements from Job Description
(2) CDER Requirements Query
(Global) Grid Information Service
Phase 1
(3) Response
(4) Replace CDER Requirements with restriction to matching Compute Elements
(5) Submit Grid Job
Job Orchestration (e.g WMS)
Phase 2

Figure 2. An abstract model of the two-phased submission of a grid job with CDER requirements.

A higher-level view of the two-phase model (Fig. 2) shows how the grid user can handle jobs requiring CDERs. In Phase-1, Step (1) is initiated on the *User Interface* – the machine from which grid jobs are submitted. This step extracts CDER requirements from the Job Description and is used to determine the type of CDERs required; Step (2) builds the GIS CDER query and then queries the GIS. This step is independent of any job-orchestration system, such as the WMS. In Step (3), results from the query are returned to the User Interface; in Step (4), the response data is used to determine the set of all site ComputeElements that support the specified CDER; but at this point, it has not yet been determined which of these ComputeElements satisfies the specific CDER requirements. The data returned for each of these ComputeElements is then reconstituted in ClassAd [14] format as a *Resource Offer.* A ClassAd *Resource Request* is created from the job CDER requirements. The Resource Request is matched against the Resource Offer to determine if the ComputeElement satisfies the original CDER requirements. In this way, a list of all ComputeElements satisfying CDER requirements can be calculated. This list of matching ComputeElements is used as the basis of a refined and targeted job requirement expression. All of this Step (4) is performed on the User Interface machine. Finally, the Phase-2 step (5) is to submit the modified grid job in the usual manner to a job-orchestration system.

To summarize, the first phase is to refine the job requirements to a targeted expression, and the second phase is to submit the targeted job to a job-orchestration system (such as the WMS). The first phase does not involve job orchestration.

4. Analysis

The conceptual strategies presented in Section 3 introduced five different approaches that may be taken when integrating CDERs into grid environments. Furthermore, some concrete examples have been presented to illustrate how these strategies may be used in practice. In Section 4.1 below, a methodology for comparing these strategies is proposed. The methodology sets out criteria for the evaluation of each strategy, while Section 4.2 provides an analysis of the strategies using these criteria. The analysis is summarized in Table 1.

Table 1
Summary of CDER Strategy Evaluation

	Discovery	Semantic Resource Detail	Semantic Structure	Dynamic Information	GLUE 1/2	Time Efficiency	Space Efficiency	2-Phase
A-Priori	No	None	None	No	Any	N/A	N/A	No
Named-Queue	Yes	Minimal	Minimal	Minimal	Any	N/A	N/A	No
Tagged-Environment	Yes	Coarse	Minimal	Deprecated	1.3	Low	N/A	No
Attribute-Extension	Yes	Fine	Yes	Yes	2.0	Med	High	Yes
Class-Extension	Yes	Fine	Yes	Yes	2.0	High	Low	Yes

To aid the comparison, each of the strategies is applied to a representative test case (a typical GPGPU resource) using an LDAP rendering. This test case is used to compare the cost of each strategy with respect to data size and query time.

4.1. Methodology

Each of the conceptual strategies proposed in Section 3 is evaluated below, with respect to the following criteria:

Discovery. It is only possible to discover previously unknown CDERs if the strategy used publishes at least some minimal information to identify the resource in the information system.

Semantic Resource Detail. The ability of a strategy to publish semantic detail beyond the mere existence of a particular type of CDER would enable more-powerful job requirement satisfaction. As an example, a CDER such as a GPGPU may have several intrinsic properties (GPGPU memory size, model, etc.) that are interesting for resource selection, so the strategy must be capable of publishing such details.

Semantic Structure. Similarly, the ability of a strategy to associate CDERs with other grid entities will again facilitate more-powerful resource selection.

Dynamic Information. While certain CDER characteristics (e.g., those describing physical hardware properties) will remain static, dynamically changing properties such as current availability, utilization, and fairshare measures will also be important considerations in resource selection. Strategies will vary in their ability to dynamically update such information.

GLUE version. Each strategy may be more or less appropriate to each version of the GLUE schema.

Information Time Efficiency. The effort required to (i) satisfy the CDER requirements for a specific grid job, and (ii) construct a complete representation of the current state of a CDER is considered for each strategy. If no data is published, then this is deemed *Not Applicable (N/A)*.

Information Space Efficiency. Conceptually, the properties of a CDER are published as Key/Value pairs. The overhead of publishing this data is considered under this heading.

Matchmaking/Job Submission Support. Some strategies will allow CDER requirements to be satisfied directly using existing job specification and submission mechanisms, while other strategies will require an extended mechanism, such as the two-phase job submission approach described in Section 3.2.

In order to illustrate and quantitatively compare the strategies, the evaluation references a contemporary example of a GPGPU CDER. A schema representing a set of typical GPGPU properties and (GPGPU-related) LRMS properties is tabulated in Table 2.

Experiment 1: Data Publication Cost

The first quantitative experiment used in Section 4.2 examines the total data size of the published CDER information when using each of the the strategies under an EGI BDII/LDAP-based realisation. The summary results are presented in Table 3.

Table 2

Schema representing typical properties of a GPGPU CDER

Sample GPGPU attributes	**Type**	**Source**	**Creation**	**Sample Values**
TotalGPGPU	Integer	LRMS	Dynamic	32
FreeGPGPU	Integer	LRMS	Dynamic	2
GPGPUPerNode	Integer	LRMS	Static	2
GPGPUCUDAComputeCapability	Float	GPGPU	Static	2.1
GPGPUMainMemorySize	Integer	GPGPU	Static	1024
GPGPUMP	Integer	GPGPU	Static	4
GPGPUCoresPerMP	Integer	GPGPU	Static	48
GPGPUCores	Integer	GPGPU	Static	192
GPGPUClockSpeed	Integer	GPGPU	Static	1660
GPGPUECCSupport	Boolean	GPGPU	Static	false
GPGPUVendor	String	GPGPU	Static	Nvidia
GPGPUModel	String	GPGPU	Static	GTS_450

Table 3

Overhead of publishing sample data under LDAP Data Interchange Format

Strategy	**Byte-count**
A Priori	N/A
Named-Queue	N/A
Tagged-Extension (GLUE 1.3)	800
Tagged-Extension (GLUE 2.0)	10500
Attribute-Extension	800
Class-Extension	4800

Experiment 2: Timed CDER Matchmaking

A second quantitative experiment looks at the time taken to determine the set of ComputeElements that satisfy a sample CDER property. Only GLUE 2.0 strategies that satisfy the *Fine Semantic Resource Detail*, *Semantic Structure*, and *Dynamic Information* criteria are considered. The experiment methodology is as follows: (i) Generate a snapshot of the GLUE 2.0 data from an EGI Top-Level BDII; (ii) the sample data from Table 2 is used to add GLUE 2.0 data representing CDERs (that don't presently exist) in the form required for that strategy. In the case of the Attribute-Extension strategy, OtherInfo attribute data is inserted into a sample set of objects. In the Class-Extension strategy case, a set of Extension objects are generated, consisting of Key/Value and Foreign Key; and (iii) the amended GLUE 2.0 is copied to a modified testbed Top-Level BDII as a snapshot. This modification to the BDII disables any further updates to the GLUE data.

The data presented in Table 4 is the average system-time cost (measured in seconds) of matching a sample ClassAd GPGPU Resource Request against 100 Resource

Offers generated from GLUE 2.0 BDII LDAP queries. The Schema in Table 2 was used to generate the GLUE 2.0 data for Attribute-Extension and Class-Extension Entities. This data was appended to 100 randomly selected GLUE2ComputingShare Entities (Attribute-Extension strategy). Similarly, 1200 Glue2Extension objects were generated to extend 100 GLUE2ComputingShares (Class-Extension strategy). The time values represent the cost of retrieval, Resource Offer generation, and match-making, and returning a list of matching resources.

Table 4
Average system-time cost, measured in seconds, of matching a sample ClassAd GPGPU Resource Request against 100 Resource Offers generated from GLUE 2.0 BDII LDAP queries.

Strategy	Time (ms)
Attribute-Extension	1.26
Class-Extension	1.31

The baseline EGI GLUE 2.0 data used to populate the Top-Level BDII is representative of the state of a very large grid infrastructure. Indeed, an EGI BDII currently contains more than 10^5 GLUE 2.0 Entities. The modified BDII's snapshot of this data is intended to simulate querying and discovering CDERs in Grids with a similar number of GLUE objects to EGI.

4.2. Strategy Evaluations

Each of the conceptual strategies is considered with reference to the criteria in Section 4.1, and a summary of the evaluation is shown in Table 1.

A-Priori Strategy

The A-Priori strategy is clearly contrary to the grid resource discovery principle – no data relating to the CDER or the LRMS is captured or published. This method must be deemed unsuitable for use for handling CDERs except for test purposes.

Named-Queue Strategy

The Named-Queue strategy requires that, at a minimum, the name of the queue is used to encode the nature of the CDER. This strategy is one of the easiest available methods to publish/discover particular CDER types. However, it does not work well as a method to encode the other properties listed in Table 2. Support for *Semantic Resource Details* and *Semantic Structure* is *Minimal*. Furthermore, *Dynamic Information* cannot be encoded into the queue name. Consequently, this method allows very limited resource discovery.

One method that may be used to publish the capacity and utilization of the CDER resources is to configure the LRMS with a 1 : 1 binding between a CPU core and a dependent CDER. Unbound CPUs must then be configured to be unavailable to the LRMS. The effect of applying this configuration is that the capacity and utilization of CDERs can be directly determined from the capacity and utilization of the CPUs. A negative consequence of this configuration is that it may result in CPU underutilization on many-core systems. This is due to the unavailability of the unbound and non-allocateable CPUs. This configuration also results in a condition whereby certain job requirements cannot be met despite the availability of sufficient resources (again, due to the unbound CPUs).

Tagged-Environment Strategy

Neither of the first two strategies support the publication of arbitrary information describing CDERs. In contrast, the remaining Tagged-Environment, Attribute-Extension, and Class-Extension strategies enable the publication of information with greater semantic detail. The first of these – Tagged-Environment – allows Sites to publish arbitrary data enabling basic CDER discovery. The limitations of this strategy are: (a) there are no implicit relationships between published tags, and any relationship between tags must be reconstructed by other means; (b) dynamic data is possible but impractical – tags are normally treated as static values. Encoding frequently changing CDER capacity, usage, or fairshare measures into a tag implies removing old tags and adding new ones. Hence, this method cannot be recommended.

If the schema in Table 2 were to be published as GLUE 1.3 SoftwareRuntimeEnvironment attributes, then the production of the sample GLUE data generates approximately 800 bytes – a relatively small amount of data.

Listing 7. Sample Data for Tagged-Environment Strategy.

```
GlueHostApplicationSoftwareRunTimeEnvironment: TotalGPGPU=32
GlueHostApplicationSoftwareRunTimeEnvironment: FreeGPGPU=30
GlueHostApplicationSoftwareRunTimeEnvironment: GPUPerNode=2
GlueHostApplicationSoftwareRunTimeEnvironment:
    ↪ GPUCUDAComputeCapability=2.1
GlueHostApplicationSoftwareRunTimeEnvironment: GPUMainMemorySize
    ↪ =1024
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUMP=4
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUCoresPerMP
    ↪ =48
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUCores=192
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUClockSpeed
    ↪ =1160
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUECCSupport=
    ↪ false
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUVendor=
    ↪ Nvidia
GlueHostApplicationSoftwareRunTimeEnvironment: GPGPUModel=
    ↪ GTS_450
```

There are, however, some negative effects that have been observed when evaluating this approach on the EGI. Namely, GLUE 1.3 SoftwareRuntimeEnvironment attributes are automatically converted by the UMD grid-middleware into individual GLUE 2.0 ApplicationEnvironment object instances. Using the the sample values from Listing 7, the GLUE2 information provider produced over 10.5 KB for the 12 ApplicationEnvironment objects – over 800 bytes per object. Note that, as the SoftwareRuntimeEnvironment tags do not exist in native GLUE 2.0, this strategy is only applicable to GLUE 1.3.

Attribute-Extension Strategy

This strategy uses GLUE 2.0 OtherInfo attributes to insert an arbitrary number of Key/Value pairs into selected Entity instances. In this way, objects can be rich in *Fine Semantic Resource Detail*. Furthermore, these Key/Value pairs can be applied to different entities as appropriate – CDER hardware properties can be applied to ExecutionEnvironment instances, and capacity and utilization data can be applied to VOShare instances, so *Semantic Structure* is supported. The individual attribute values in object instances can be *Dynamically* updated, but only by converting the Key/Value pair to a string and replacing the old string with the new one. This has an impact on the *Time Efficiency*. All GLUE 2.0 Entities can be extended in this strategy by using OtherInfo, but support under GLUE 1.3 is limited to a few Entities that support *capability* attributes. *Space Efficiency* is *High* as data publishing costs are similar to the GLUE 1.3 Tagged-Environment strategy (approximately 800 bytes), but would increase linearly (by less than 160 bytes per VOShare). To use the CDER data to submit grid jobs, the two-phase method must be used.

Class-Extension Strategy

Arbitrary Extension instances, consisting of a key, a value, and a *Foreign Key* reference to the object instance that it extends, can be created in the GLUE 2.0 LDAP rendering. As a result, the basic *Discovery*, *Fine-grained Semantic Resource Details*, and *Semantic Structure* criteria are satisfied. *Dynamic Information* is supported by updating the Extension value. Furthermore, discovering Extension attributes Keys or Values are low-cost operations, so the *Time Efficiency* is *High*. There is an overhead in the creation of each Extension instance, as each Extension (and its Key/Value/Foreign Key) is encapsulated with extra LDIF data. This encapsulation data is generally several orders of magnitude greater than both the key and value. For the sample data, the average number of bytes for both the key and values was approximately 17 bytes, but encapsulating a single Key/Value pair in an object costs in the order of 400 bytes (including the Foreign Key). So, *Space Efficiency* is *Low* in comparison to Attribute-Extension, which is counter-intuitive for a by-reference mechanism. To use the CDER data to submit grid jobs, the two-phase method must be used.

5. Related Work

GLUE Rendering and Information Models

The LDAP rendering is one of several methods that may be used to publish GLUE information. Other renderings that have been used include XML (the Nordugrid ARC middleware [15]) and SQL (R-GMA [5]). Furthermore, the XML approach to handling Extensions differs significantly to the LDAP approach. In the XML rendering, all Extensions and their attributes are inserted as a collection of XML subtrees to the object that they extend. This approach falls under the *Attribute-Extension* strategy.

Handling Software Licenses

The question of how to handle limited software licenses in grid environments is quite similar to the CDER problem, particularly in the case where access to software is *node-locked*; i.e., the software may only run on certain nodes due to license restrictions. A typical way to handle this in the LRMS is to assign a nominal "property" to the node – when the user needs a license, then a specific LRMS directive ensures the allocation of only those nodes satisfying this property. Furthermore, by treating the software as a Generic Consumable Resource (which is now supported by most LRMS's), the usage of the software can also be managed by the LRMS. The key differences between the treatment of CDERs and the Software Applications is that the GLUE 2.0 ApplicationEnvironment can already be used to publish the availability of particular software as well as to indicate capacity and usage of licenses. Other GLUE 2.0 Entities such as the ApplicationEnviromentHandle can be used to guide the user application on how to configure or bootstrap the application's environment.

Other Information Systems

The Relational Grid Monitoring Architecture (R-GMA) [5] was developed as a generic grid information system based on a Producer/Consumer model. Inserting, updating, deleting, and querying of data was available through command-line tools and an API that supported a subset of the SQL92 query language standard. In addition to publishing GLUE 1.3, R-GMA allowed users and services to publish, consume, update, and query other non-GLUE data. This system was used to publish LRMS Usage Records to monitor the usage of Compute Resources across the EGEE, a grid infrastructure that evolved into the current EGI. User-based applications included grid-wide intrusion detection [10]. The R-GMA may have been a suitable facility for Sites to publish up-to-date information about CDERs without depending on the GLUE and the GIS; however, this is no longer in use anywhere.

Hierarchical Brokering

Toor et al [18] describe a prototype extension to the ARC middleware's job-submission system on a User Interface. Additional static and dynamic information about resources are published through a GLUE 2.0 XML-based information system. ARC performs brokering on the User Interface, and this is known as *Distributed Brokering*.

Under the gLite/UMD middleware, job submission can either be directed at a user-specified resource without any brokering or left to a job-orchestration system (such as the Workload Management System). The WMS is an example of *Centralised Brokering*. The two-phase method presented in this paper implements *Hierarchical Brokering* with the pre-filter stage acting as a *Distributed Broker* and the final job-orchestration through the WMS acting as a *Centralised Broker*.

The available brokering criteria (Random, FastestQueue, Benchmark, Data) [19] used by the ARC middleware to select resources is not as comprehensive as those available under the WMS. Indeed, the WMS match-making system simultaneously supports multiple criteria (Queue Utilization, Benchmark, etc.) for ranking matching resources.

6. Conclusions and Future Work

Any strategy for integrating CPU-dependent execution resources into grid infrastructures must support the fine-grained publication of the resource's properties and capabilities. In addition, the dynamic publication of the number of such resources installed (capacity), their utilization, and any fairshare measures are also required. Complementary mechanisms to discover and select resources based on these factors (properties, capabilities, capacity, utilization, and fairshares) allow grid users to refine the selection of Sites to only those that match their needs.

Of the five conceptual strategies investigated in this paper, some strategies that have previously been applied for use on the grid – A Priori, Named-Queue, and Tagged-Extension – cannot be effectively used with CDERs. The final two strategies (Attribute-Extension and Class-Extension) have been shown to be the most-flexible strategies for adding arbitrary (CDER) attributes and their values. However, they have their own weaknesses and strengths. The Attribute-Extension strategy is more data efficient, and the extension data is added internally to the object instance. The Class-Extension strategy creates new Extension instances, and this incurs a data volume penalty. However, updating individual attribute values will be more time efficient.

Although the match-making time measurements (Table 4) show that the Class-Extension strategy is slightly slower than the Attribute-Extension strategy, there is scope for reducing the time taken for Class-Extension even further. The measured time includes the cost of retrieving *all* Key/Value pairs for the GPGPU CDER. This need not be the case for the Class-Extension strategy. Only the Keys/Value pairs corresponding to the keys specified in the job-requirements expression are needed when generating the Resource Offer. Under Attribute-Extension, all OtherInfo values are retrieved – this is due to limitations in constructing more-specific LDAP queries.

The question of whether a hybrid approach (i.e., using both Attribute and Class-Extension strategies to describe the CDER state) may offer a more-optimal solution has yet to be investigated. In particular, a hybrid approach that uses Attribute-Extension for describing static properties and uses Class-Extensions for dynamically changing data may be a good space and time solution, and should be explored in future work.

Further areas of improvement that could be addressed in future work may be to investigate strategies that help avoid CDER namespace collisions. These collisions may occur if different Sites use the same name to advertise different CDER types, or in the case of the same CDER type, different attributes have been advertised. A potential solution is to ensure that each CDER definition has a globally identifiable unique identifier that is published as part of the CDER schema.

Acknowledgements

This work carried out on behalf of the Telecommunications Graduate Initiative (TGI) project. TGI is funded by the Higher Education Authority (HEA) of Ireland under the Programme for Research in Third-Level Institutions (PRTLI) Cycle 5 and co-funded under the European Regional Development Fund (ERDF). The authors also acknowledge the use of additional support and assistance provided by the European Grid Infrastructure. For more information, please reference the EGI-InSPIRE paper[11].

References

[1] Anderson D.P.: BOINC: A System for Public-Resource Computing and Storage. In: *5th International Workshop on Grid Computing (GRID 2004), 8 November 2004, Pittsburgh, PA, USA, Proceedings*, R. Buyya, ed., pp. 4–10. IEEE Computer Society, 2004. `http://doi.ieeecomputersociety.org/10.1109/GRID.2004.14`.

[2] Bird I.: Computing for the Large Hadron Collider. *Annual Review of Nuclear and Particle Science*, vol. 61(1), pp. 99–118. `http://dx.doi.org/10.1146/annurev-nucl-102010-130059`.

[3] Burke S., Campana S., Lanciotti E., Litmaath M., Lorenzo P.M., Miccio V., Nater C., Santinelli R., Sciaba A.: The gLite 3.2 User Guide. `https://edms.cern.ch/file/722398/1.4/gLite-3-UserGuide.pdf`, 2012.

[4] Burke S., Field L., Horat D.: Migration to the GLUE 2.0 information schema in the LCG/EGEE/EGI production Grid. *Journal of Physics: Conference Series*, vol. 331(6), p. 062004, 2011. `http://stacks.iop.org/1742-6596/331/i=6/a=062004`.

[5] Cooke A.W., Gray A.J.G., Ma L., Nutt W., Magowan J., Oevers M., Taylor P., Byrom R., Field L., Hicks S., Leake J., Soni M., Wilson A.J., Cordenonsi R., Cornwall L., Djaoui A., Fisher S., Podhorszki N., Coghlan B.A., Kenny S., O'Callaghan D.: R-GMA: An Information Integration System for Grid Monitoring. In: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE – OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3–7, 2003*, *Lecture Notes in*

Computer Science, R. Meersman, Z. Tari, D.C. Schmidt, eds, vol. 2888, pp. 462–481. Springer, 2003. http://dx.doi.org/10.1007/978-3-540-39964-3_29.

[6] David M., Borges G., Gomes J., Pina J.M., Plasencia I.C., Fernández-del-Castillo E., López Á., Orviz P., Cacheiro J.L., Fernández C., Simón Á.: Software Provision Process for EGI. *Computing and Informatics*, vol. 31(1), pp. 135–148, 2012. http://www.cai.sk/ojs/index.php/cai/article/view/892.

[7] Fernández-del-Castillo E., Walsh J., Simon A.: Parallel Computing Workshop. In: *Proceedings of the EGI Community Forum 2012/EMI Second Technical Conference*. Proceedings of Science, Munich, Germany, 2012. http://pos.sissa.it/archive/conferences/162/057/EGICF12-EMITC2_057.pdf.

[8] Fuller S., Millet L., eds.: *The Future of Computing Performance: Game Over or Next Level?* The National Academies Press, Washington, DC, 2011. http://www.nap.edu/catalog/12980/the-future-of-computing-performance-game-over-or-next-level.

[9] Germain-Renaud C., Loomis C., Moscicki J.T., Texier R.: Scheduling for Responsive Grids. *Journal of Grid Computing*, vol. 6(1), pp. 15–27, 2008. http://dx.doi.org/10.1007/s10723-007-9086-4.

[10] Kenny S., Coghlan B.A.: Towards a Grid-wide Intrusion Detection System. In: *Advances in Grid Computing - EGC 2005, European Grid Conference, Amsterdam, The Netherlands, February 14-16, 2005, Revised Selected Papers*, P.M.A. Sloot, A.G. Hoekstra, T. Priol, A. Reinefeld, M. Bubak, eds, *Lecture Notes in Computer Science*, vol. 3470, pp. 275–284. Springer, 2005. http://dx.doi.org/10.1007/11508380_29.

[11] Newhouse S.: EGI-InSPIRE paper. http://go.egi.eu/pdnon, 2010.

[12] OGF GLUE 1.3 Specification. https://redmine.ogf.org/dmsf_files/61?download=.

[13] OGF GLUE 2.0 Specification. http://redmine.ogf.org/dmsf/glue-wg?folder_id=18.

[14] Raman R., Livny M., Solomon M.H.: Matchmaking: Distributed Resource Management for High Throughput Computing. In: *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, HPDC '98, Chicago, Illinois, USA, July 28–31, 1998*, pp. 140–146. IEEE Computer Society, 1998. http://dx.doi.org/10.1109/HPDC.1998.709966.

[15] Smirnova O., Ellert M., Johansson D.: ARIS and EGIIS: Installation, Configuration and Usage Manual. http://www.nordugrid.org/documents/aris-egiis.pdf.

[16] The EGI MPI Users Guide. https://wiki.egi.eu/wiki/MPI_User_Guide.

[17] Top 500 Supercomputers Highlights – November 2014. http://www.top500.org/lists/2014/11/highlights/.

[18] Toor S., Mohn B., Cameron D., Holmgren S.: Case-Study for Different Models of Resource Brokering in Grid Systems. http://www.it.uu.se/research/reports/2010-009/2010-009-nc.pdf, 2010.

[19] User Manual for ARC 11.05 (client version 1.0.0) and above. `http://www.nordugrid.org/documents/arc-ui.pdf`, 2014.

[20] Vella F., Cefalá R.M., Costantini A., Gervasi O., Tanci C.: GPU Computing in EGI Environment Using a Cloud Approach. In: *International Conference on Computational Science and Its Applications, ICCSA 2011, Santander, Spain, June 20–23, 2011*, A. Iglesias, B.O. Apduhan, O. Gervasi, D. Taniar, M.L. Gavrilova, eds, pp. 150–155. IEEE Computer Society, 2011. `http://doi.ieeecomputersociety.org/10.1109/ICCSA.2011.61`.

[21] Walsh J., Coghlan B., Eigelis K., Sipos G.: Results from the EGI GPGPU Virtual Team's User and Resource Centre Administrators Surveys. 2012. Crakow Grid Workshop 2012.

[22] Walsh J., Dukes J.: Supporting job-level secure access to GPGPU resources on existing grid infrastructures. In: *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, September 7–10, 2014*, M. Ganzha, L.A. Maciaszek, M. Paprzycki, eds, pp. 781–790, 2014. `http://dx.doi.org/10.15439/2014F337`.

Affiliations

John Walsh
School of Computer Science and Statistics, Trinity College Dublin, `John.Walsh@scss.tcd.ie`

Jonathan Dukes
School of Computer Science and Statistics, Trinity College Dublin, `jdukes@scss.tcd.ie`

Gabriele Pierantoni
School of Computer Science and Statistics, Trinity College Dublin, `pierantg@scss.tcd.ie`

Brian Coghlan
School of Computer Science and Statistics, Trinity College Dublin, `coghlan@scss.tcd.ie`

Received: 2.12.2014
Revised: 15.05.2015
Accepted: 19.05.2015