A Capability-Based Matchmaking Mechanism Supporting Resource Aggregation within Large-Scale Distributed Computing Infrastructures

Feng Liang[1(✉)], Hai Liu[1], Yunzhen Liu[1], Shilong Ma[1], Siyao Zheng[2], and Pan Deng[3]

[1] State Key Lab of Software Development Environment, Beihang University, Beijing 100191, China
{liangfeng,liuhai,liuyunzhen,slma}@nlsde.buaa.edu.cn
[2] The Laboratory of Embedded Systems, Beihang University, Beijing 100191, China
zhengsiyao@les.buaa.edu.cn
[3] Lab of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
dengpan@iscas.ac.cn

Abstract. Facing the large-scale, heterogeneous dynamic resource and the complex constraints of computation-intensive parallel scientific applications, collaborating large-scale computation resource for these applications within Large-Scale Distributed Computing Infrastructures like grids and clouds are challenging. This paper addresses this issue by proposing a Resource Capability Model and implementing a corresponding language GSO and an information service Application Information Service to ensure not only the resource description, aggregation, matchmaking but also life cycle management of the reservations and tasks. Experiment result shows that our mechanism is scalable in both Grid and Cloud environment.

Keywords: Grid Service Object · Large-scale Distributed Computing Infrastructures · Resource Capability model · Application Information Service

1 Introduction

As Lage-scale Distributed Computing Infrastructures (LDCI) like grids and clouds become robust, more and more computation-intensive parallel scientific applications want to utilize them. But because of the following reasons, it is not easy to matchmake the suitable resource for these applications. (1) The large-scale clusters increases the time to search for the suitable resource. (2) The diverse heterogeneity of the nodes makes it hard to compare the resource capability. (3) The dynamic feature of the nodes requires more concern on the availability of the nodes. (4) The computation-intensive scientific applications

V.C.M. Leung and M. Chen (Eds.): CloudComp 2013, LNICST 133, pp. 145–154, 2014.
DOI: 10.1007/978-3-319-05506-0_14,

demands large-scale computation resource. (5) The resource collaboration constraints lead to the life cycle management of the reservations and tasks.

With large-scale, diverse heterogeneity and dynamic resource and complex resource requirement of the applications, it is required to establish a resource matchmaking mechanism for fine-grained resource description, proper resource aggregation, and optimized resource matchmaking. To address these issues, this paper proposes a Resource Capability Model and its XML implementation Grid Service Object language to support the resource description, aggregation and matchmaking and life cycle management.

The contributions of this paper include two aspects: firstly, we propose the Resource Capability model to measure the capability of a single resource and aggregated resources. Secondly, we propose a Grid Service Object language based on the Resource Capability model and implement it in an Application Information Service, so as to support the matchmaking for aggregated resources. Experiments show that our methods are acceptable for production-level LDCI.

This paper is structured as follows: we begin with a discussion of related work in Sect. 2. The Resource Capability Model is given in Sect. 3. Section 4 presents the GSO and AIS in details. Section 6 presents experimental results that evaluate the scaling and effectiveness of our mechanism. In Sect. 7 we summarize the contributions of this paper and present some areas of future work.

2 Related Work

To proper describe the resource and the request, a proper and complete resource and request description model is required for resource discovery and matchmaking. Up to now the existing models can be categorized into three classes: the request description models, the resource description models and the symmetrical models.

2.1 Job Description Models

The request description models are used mainly for task requirement description. The typical models are RSL (Resource Specification Language) [4] and JSDL (Job Submssion Description Language) [2]. RSL is used mainly in Globus Toolkit, it uses the $\{attribute, value\}$ key-value pair for requirement description and utilizes operations &, | and + for single task description, the logical relationships between tasks and the resource set respectively, but its format is not friendly for collaborate with other LDCI. Proposed by the Open Grid Forum and implemented in XML format, JSDL is used widely in Grid, but it can support only task descriptions. JSDL can be considered as an extended RSL, as it allows the user and the grid machines to add new keys for extra attributes.

2.2 Requirement Description Models

The resource description models are for the resource description. The most widely accepted models are GLUE (Grid Laboratory for a Uniform Environment) [1]

and DRMAA (Distributed Resource Management Application API) [10]. GLUE is the proposed by the Open Grid Forum using the accumulated experience from piratical Grid Projects. It encapsulate the computation resource into a Computing Element and uses the $\{attribute, value\}$ for description. The Monitoring and Discovery Service from Globus Toolkit implements GLUE in XML and manages the Computing Elements in hierarchy. But this model does not support the resource aggregation. DRMAA is a Open Grid Forum proposed generalized API for distributed resource management systems in order to facilitate the development of portable application programs and high-level libraries. DRMAA includes detailed definitions about resource descriptions in aspects such as os version, cpu architecture, etc.

2.3 Symmetrical Models

The symmetrical models include ClassAds (Condor classified Advertisement) [9] and GODsL (Grid Object Description Language) [6]. ClassAds is used in Condor, every ClassAds is an entity to describe both the resource attributes, the job attributes and the preference attributes. ClassAds uses the $\{attribute, value\}$ for description and allow the users to define constraints in job type, access control, time, resource requirement. But it does not support resource aggregation. GODsL is an object-oriented extensible description model, it defines Grid Object (GO) to define the job and its resources. Each GO includes 5 containers, namely the Resource Container, the Server Container, the Machine Container and the Backup Container, each container can contain one or more profiles about resource, used machine, file path, backup location. GODsL is mainly used for job migration and recovery and is written in C, which is not proper for collaboration.

From all the listed resource and request description models, we can see although the current existing models do support detailed description of single resource and request, but there is no model supporting resource aggregation and collaboration, which is essential for accurate matchmaking for those computation-intensive scientific parallel computing applications.

3 Resource Capability Model

As the crucial factor of LDCI systems, the computation node evaluates its capability in different metric such as task execution time, success rate and its availability, etc. These metric can be measured using the attributes of nodes such as the CPU speed, memory volume, network bandwidth, etc. According to the effects of the nodes in different aspects, the node capability can be split into 5 aspects: Computation Capability, Communication Capability, Memory Capability, Availability Capability and the Software Support Capability.

3.1 Capability-Based Matchmaking for a Single Node

The capability of a single resource $node_i$ is described as in $C_i = \{Ssup_i, Comp_i, Comm_i, Memo_i, Avai_i\}$. The 5 aspects and its meanings are as follow. $Ssup_i$, the

Software Support Capability of $node_i$, means the Software Environment of the node for computation, including the Operating System, the runtime software, the third party library, etc. $Comp_i$, the Computation Capability of $node_i$, measures the processing ability of the processor by the CPU speed. A multi-core processor is considered as several equal CPU processors. $Comm_i$, the Communication Capability of $node_i$, measures the data transfer of the the node's network by both the latency and the bandwidth. $Memo_i$, the Memory Capability of $node_i$, measures the memory of the nodes by the memory volume. $Avai_i$, the Availability Capability of $node_i$ measure the availability of the node by the online time percentage of the node.

During the matchmaking process for a single computation node, the capability requirement should be satisfied by the node candidate. Assuming the capability of the request and the node are described as $C_r = \{Ssup_r, Comp_r, Comm_r, Memo_r, Avai_r\}$ and $C_a = \{Ssup_a, Comp_a, Comm_a, Memo_a, Avai_a\}$ respectively, then its matchmaking for 5 aspects are defined below.

$Ssup$ is a qualitative attribute, usually the requirement and the node should be equal in this attribute to ensure an execution. $Comp_a, Comm_a, Memo_a, Avai_a$ are quantitative attributes. As this paper aims mainly for the execution of computation-intensive parallel applications, so the node should has bigger capability than the requirement for $Comp_a, Comm_a, Memo_a, Avai_a$ respectively.

3.2 Matchmaking for Aggregated Resource

As the parallel computation program normally need multiple nodes, so it is essential to aggregate their capability and match with the requirement. Assume the capability requirement of a parallel application is defined in $CReq = \{C_R, N\}$ and $C_R = \{Ssup_R, Comp_R, Comm_R, Memo_R, Avai_R\}$. Within it, C_R means the requirement of each node, N means the required node number. Assuming there exists N nodes $\{C_i | 1 \leq i \leq N\}$, These nodes need to be aggregated first to match with the requirement. Assume the aggregated capability of these nodes is $CAgg$, as shown in $CAgg = \{C_A, N\}$ and $C_A = \{Ssup_A, Comp_A, Comm_A, Memo_A, Avai_A\}$. When $CAgg$ satisfies $CReq$, there should be $Ssup_A = Ssup_R, Comp_A \geq Comp_R, Comm_A \geq Comm_R, Avai_A \geq Avai_R, Memo_A \geq Avai_R$. As the N nodes might be heterogeneous, so the aggregation process for the $Ssup$ and $comp$ capabilities are listed as $Ssup_A = Ssup_i, 1 \leq i \leq N$ and $Comp_A = min_{i=1 \cdots N}(Comp_i)$. The others are similar.

As a qualitative attribute, the required capability of $Ssup_A$ should be equal to each of the N nodes. As a quantitative attribute, the required capabilities $Comp_A, Comm_A, Memo_A, Avai_A$ should be equal to the lowest value among the nodes as the execution time of a parallel application mostly depends on the lowest node.

3.3 Matchmaking for Co-reservation

For a single reservation, it can be described as q, and then the request for co-reservation is $Q = \{q_1, q_2, \cdots, q_l\}$ $(l \in N)$, i.e. each co-reservation request Q has

multiple sub-request q. The resource set is $R = \{r_1, r_2 \cdots, r_k\}$ $(k \in N)$, a R includes multiple r. $R(q)$ means all the r that satisfies q. t means the start time, d means the duration, then the quadruple $A = \langle q, r, t, d \rangle$ means to allocate a resource r from time t with the duration of d to satisfy the requirement q. Then the total allocation result would be in $TA(Q, R) = \{\langle q, r, t, d \rangle\} = \{A \mid \bigcup q_A = Q, \forall A \neq B, r_A \neq r_B\}$, it means an executable solution for resource allocation.

For a $TA(Q, R)$, it has to satisfy the temporal and spatial constraints. Suppose a single constraints is $sc(A)$. Then the $TA(Q, R)$ has to satisfy all single constraints, as is shown in $SC = \bigcup_{q_l \in Q, r_k \in R(q_l)} sc(q_l, r_k, t, d) = \bigcup_{A \in TA(Q,R)} sc(A)$. Assume two arbitrary node allocations A, B and their network allocation C, then the temporal constraint among these three is $tc(A, B, C)$, and the spatial constraint is shown in $pc(A, B, C) = pc_{net}(A, B, C) \cup pc_{nnt}(A, B)$ including both the network constraint and the non-network constraint. If the constraints among multiple sub tasks is $mc(ASet)$ and $ASet$ is shown in $mc(ASet), ASet = \{A_1, A_2, \cdots, A_l\}$.

So the final constraints for $TR(Q, R)$ can be described as in $(\bigcup_{A \in TA(Q,R)} sc(A)) \bigcup (\bigcup_{A,B,C \in TA(Q,R)} pc(A, B, C) \bigcup tc(A, B, C)) \bigcup (\bigcup_{ASet \subseteq TA(Q,R)} mc(ASet))$.

4 Grid Service Object Language

To ensure the execution of the Grid application with co-reservation, a information model is required to support the resource description, resource aggregation and the life cycle management of the task and reservations. As an earlier Grid resource information model, GODsL defined an extensible information model to describe the resource, applications, service and data within Grid. But this language did not support resource capability description, resource aggregation or the life cycle management of the task and reservations, besides, it used the C language syntax, therefore not suitable to exchange information with the modern Grid systems using XML. So Based on the Grid Object Description Language (GODsL), the Grid Service Object model (GSO) is proposed to address these issues. This model refines the GODsL according our Resource Capability Model and add the support about resource aggregation and the life cycle management of the tasks and reservations. XML Scheme is used to implement this language to ensure its compatibility with the other Grid Systems.

GSO model can be seen as in Fig. 1. Every GSO object owns a global unique id and a version number, to identify it. Besides, it also contains three sub-object, so called container, namely *RequirementContainer*, *ExecutionContainer* and *ResourceContainer*. These three containers can include 1 or more sub members. The contents of each container is listed as below:

RequirementContainer. It is used to describe the requirement of the resource capability, including the specifications about resource and the reservations. The *RequirementContainer* contains one or more $ResourceProfile$. It describes the Capability of nodes, shown in Table 1. In order be compatible with more distributed resource systems, the *RequirementContainer* adapt the DRMAA API standards for resource requirement description. Besides these $ResourceProfile$,

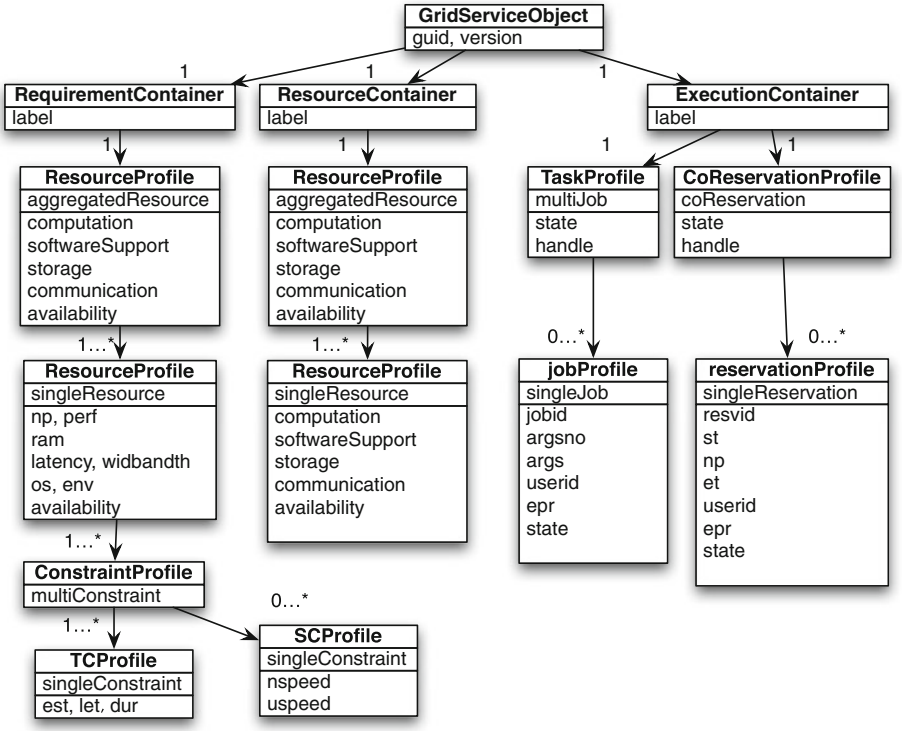

Fig. 1. GSO model

Table 1. The ResourceProfile within the RequirementContainer and the ResourceContainer

Capability	Name	Meaning	Format
Computation	np	number of processors	positive integer
	perf	processor performance	positive integer (GHz)
Memory	ram	memory	positive integer (GB)
Communication	band width	the bandwidth	positive number (MB/s)
	latency		positive number (ms)
	os	OS name and version	String_version number
SoftwareSupport	env	runtime name and version	multiple String_version, split using ";"
Availability	online ratio	the online time length within the last 100 days	percentage (%)

the *RequirementContainer* also contains zero or more $ConstraintProfile$ (to describe the constraints), which include $SCProfile$ (for spatial constraints) and $TCProfile$ (for temporal constraints).

ExecutionContainer. It is used to describe to related attributes and status of the tasks and reservations during the execution process, so mainly the life cycle management the tasks and the reservations.

ResourceContainer. It is used to describe the resource capability and status, including the computation capability, the storage capability, the communication capability, the software support capability and the availability. It is mainly for the resource management. The contents are the same as the *RequirementContainer*. Besides, it also includes the unique url of the resource.

5 Application Information Service in Migol

Migol [5,7,8] is a grid middleware, it aims at providing fault tolerance functionality to ensure the robust execution of the computation-intensive parallel applications. Composed of a set of loosely-coupled service, Migol follows the Open Grid Service Architecture (OGSA) [3]. Migol makes use of some basic components form Globus Toolkit, such as GRAM and MDS. This modularized service-oriented design allows Migol to collaborate with other Grid Middleware or to utilize the Cloud resource. As shown in Fig. 2, Migol uses GSO as the basic metadata model to store all the related information of the applications to ensure their execution.

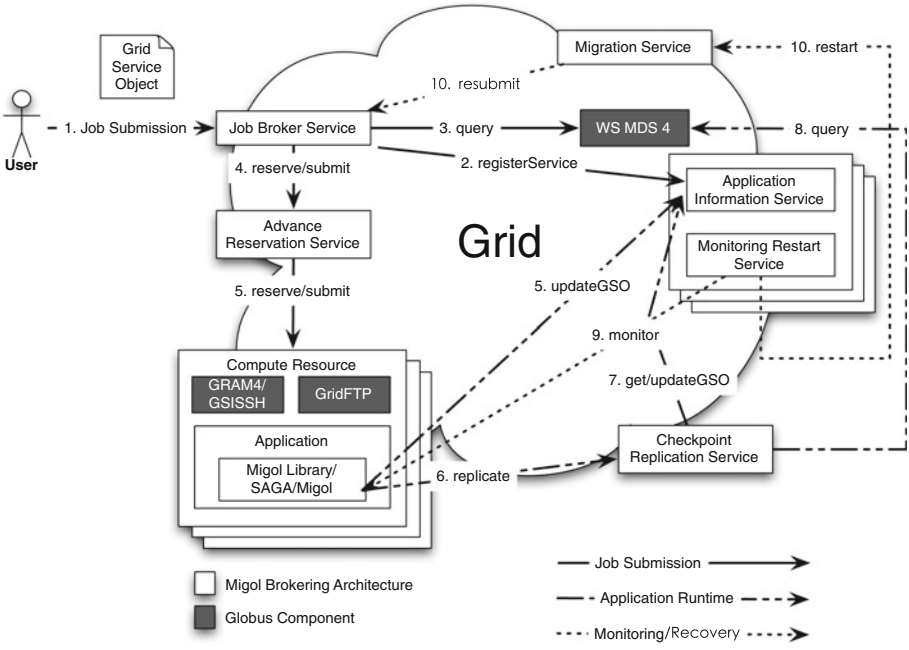

Fig. 2. Migol architecture

Application Information Service (AIS) is a registry service for LDCI applications within Migol. It stores all the GSO. Before the application execution, the user should registry the application's GSO to AIS. After the registration, the GSO contains information such as resource requirement, resource location, application task states, reservation states, etc. Meanwhile, the updated information of the used resource, tasks and reservations are collected from MDS, JBS, ARS and CRS respectively.

6 Experimental Results

6.1 Testbed Configuration

In order to evaluate the efficiency of the AIS service, an experiment is conducted to test for its currency. The experiment include 2 server nodes for services such as JBS and AIS and 3 different LDCI resources, a local cluster, a SeisGrid resource site and an Amazon EC2 virtual server. The two server nodes each has 1 GB memory, AMD Opteron 2.2 GHz processor with Globus Toolkit 4.0.5 and JRE 1.5.0_14 installed. The 3 resource are shown in Table 2 and the testbed is deployed as shown in Fig. 3.

Table 2. Site network configurations

Resource name	Roundtrip time	Band width
Local cluster	0.1 ms	95.0 Mbit/s
SeisGrid	5.8 ms	68.5 Mbits/s
EC2 server	180 ms	1.98 Mbit/s

6.2 AIS Concurrency Evaluation

As the AIS needs to exchange information with JBS, ARS, MDS and CRS, so it faces great pressure when multiple users query it concurrently. To evaluate the robustness of AIS, a pressure test is conducted. Considering the network delay of resources can have effects to AIS, so three different LDCI resources are used.

During the experiment process, multiple users registers a 3.1 kb GSO file concurrently using AIS. This file will lead AIS to access each of the three different resource and execute the matchmaking. The response time of AIS varies with different resource and different number of concurrent user numbers, as shown in Fig. 4.

From Fig. 4, it can be seen that the response time of AIS grows when the number of concurrent users increase for all three resources. But for local cluster and SeisGrid cluster, the growth increase linearly, while for EC2 virtual server, it increases faster. This is reasonable as it has bigger delay, so when more users try to access the resource concurrently, the delay gets bigger easily.

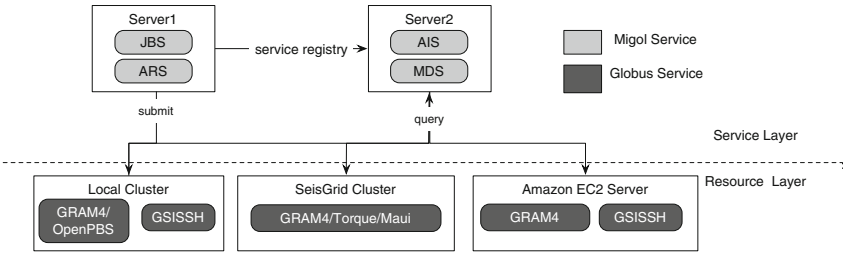

Fig. 3. Testbed deployment

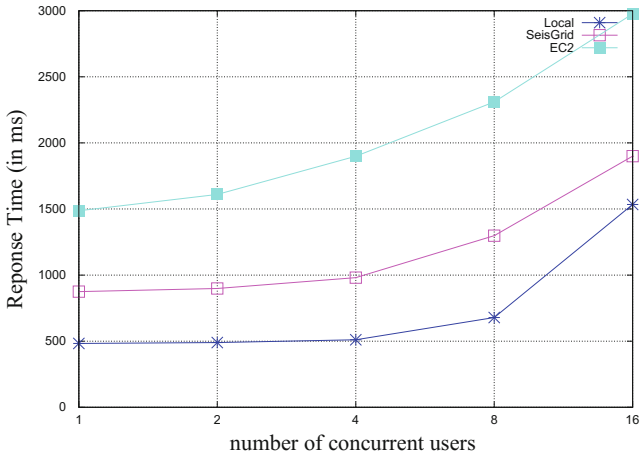

Fig. 4. The comparison of AIS response time with different resource delays and different number of concurrent user numbers

7 Conclusions

The middleware for utilizing LDCI should support optimized resource matchmaking so as to ensure the automatic resource collaboration. As the resource in LDCI are large-scale, dynamic, and heterogeneous, and the resource requirement of the computation-intensive parallel scientific applications are complex and include multiple constraints, so a mechanism is required for fine-grained resource description, proper resource aggregation, optimized resource matchmaking. Different from the current resource description models, this paper proposes a Resource Capability Model to enable resource description, aggregation and optimized matchmaking. Based on the Resource Capability model, a XML-based GSO language and the Application Information Service is developed to implement the matchmaking process. The experiment result shows that our mechanism is scalable in both Grid and Cloud environment.

Acknowledgement. This research work was supported by the Migol project from the Potsdam University (http://www.migol.de), the self-conducted exploratory research program "Green Lighting in Internet of Things" from State Key Laboratory for Software Development Environment in China (NO.SKLSDE-2010ZX-06), the Special Program for Seism-Scientific Research in Public Interest "Research in Online Processing Technologies for Seismological Precursory Network Dynamic Monitoring and Products" (NO. 201008002) and the National Natural Science Foundation of China (No. 61100066)

References

1. Andreozzi, S., Ehm, F., Field, L., Kónya, B.: GLUE specification. http://ogf.org/documents/GFD.147.pdf. March 2009
2. Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., Savva, A.: Job submission description language (jsdl) specification, version 1.0. http://www.gridforum.org/documents/GFD.136.pdf. July 2008
3. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: an open grid services architecture for distributed systems integration. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/physiology.pdf (2002)
4. Foster, I.: Globus toolkit version 4: software for service-oriented systems. In: Jin, H., Reed, D., Jiang, W. (eds.) NPC 2005. LNCS, vol. 3779, pp. 2–13. Springer, Heidelberg (2005)
5. Jeske, J., Luckow, A., Schnor, B.: Reservation-based resource-brokering for grid computing. In: Proceedings of German E-Science Conference 2007, Baden-Baden, Germany, pp. 1–10 (2007)
6. Lanfermann, G., Schnor, B., Seidel, E.: Grid object description: characterizing grids. In: Eighth IFIP/IEEE International Symposium on Integrated Network Management, Colorado Springs, Colorado, USA, March 2003
7. Luckow, A., Schnor, B.: Migol: a fault-tolerant service framework for mpi applications in the grid. Future Gener. Comput. Syst. - Int. J. Grid Comput. **24**(2), 142–152 (2008)
8. Migol Research Group: Migol, Migration in the Grid OGSA Lite. http://migol.de/. (2010)
9. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. Concurrency - Pract. Experience **17**(2–4), 323–356 (2005)
10. Troger, P., Brobst, R., Gruber, D., Mamonski, M., Templeton, D.: Distributed resource management application API version 2 (DRMAA). http://www.ogf.org/documents/GFD.194.pdf. January 2012