J Grid Computing
DOI 10.1007/s10723-006-9060-6

The Design, Usage, and Performance of GRUBER: A *G*rid *U*sage Service Level Agreement based *B*rok*ER*ing Infrastructure

Catalin L. Dumitrescu · Ioan Raicu · Ian Foster

Received: 9 March 2006 / Accepted: 4 November 2006

Abstract We present GRUBER, a *G*rid *R*esource *U*sage service level agreement (uSLA) based *B*rok*ER*ing infrastructure, aimed at addressing the challenging issues that can arise within virtual organizations (VOs) that integrate participants and resources spanning multiple physical administrative domains. In such environments, participants delegate to one or more VOs the right to use certain resources subject to local policy and service level agreements; each VO then uses those resources subject to VO policy. GRUBER supports the explicit representation, enforcement, and management of service level agreements (SLAs) concerning resource usage (uSLAs) that can serve as an objective organizing principle for controlled resource sharing in distributed systems. uSLAs express how resources must be used over various time intervals and represent a novelty for the Grid domain. This paper provides a detailed overview of the GRUBER infrastructure, the evolution of its design to improve scalability, specifically the distribution of the resource brokering service, and the extended support for dynamic environments. We also present various results achieved over time that demonstrate both the utility and performance of GRUBER under various application workloads and scenarios.

Key words controlled resource sharing · Grid computing · resource management resource and service brokering · usage service level agreements (uSLAs)

Abbreviations

uSLA	usage Service Level Agreement
VO	Virtual Organization
RM	Resource Manager

1 Introduction

GRUBER is an infrastructure for usage service level agreements (uSLAs [1]) specification, management and enforcement in any distributed environment in general; our implementation has been successfully

This work was carried out for CoreGrid IST project n°004265, funded by the European Commission.

C. L. Dumitrescu (✉)
Mathematics and Computer Science,
Delft University of Technology,
Mekelweg 4, Delft,
2628 CD Delft, The Netherlands
e-mail: c.dumitrescu@ewi.tudelft.nl

I. Raicu
Computer Science Department,
The University of Chicago,
5801 S. Ellis Ave.,
Chicago, IL 60637, USA
e-mail: iraicu@cs.uchicago.edu

I. Foster
Mathematics and Computer Science Division,
Argonne National Laboratory,
9700 S. Cass Ave., MCS/221,
Argonne, IL 60439, USA
e-mail: foster@mcs.anl.gov

deployed and used in Grid environments in particular. In these environments, each resource owner assigns a percentage of the owned resources to several consumers. The aggregated virtual resources assigned to a consumer can be used for the consumer immediate benefits, may be shared among various consumer's entities or provided further to other third parties. In the third scenario, the first consumer acts as a middleman for resource aggregation and provisioning. The total resource amount each consumer can use depends on the specified uSLAs at each level in the allocation chain.

The novelty of GRUBER consists in its capability to provide a means for enforcement of uSLAs and support for automated agents to select available resources in a Grid-like environment. It focuses on computing resources such as computers, storage, and networks and higher-level services as well as any Grid services. A VO is a group of participants who seek to share resources for some common purpose. From the perspective of a single site in an environment such as Grid3 [2, 69], a VO corresponds to either one or several users, depending on local uSLAs. However, the problem is more complex than a cluster fair-share allocation problem, because each VO has different allocations under different scheduling policies at different sites and, in parallel, each VO might have different task assignment policies. This heterogeneity makes the analogy untenable when there are many sites and VOs.

The rest of this paper is organized as follows. Section 2 introduces the controlled resource sharing problem, presents the common scenarios for uSLA-based sharing and related work. Section 3 presents GRUBER's design. Next, Sections 5 and 6 cover GRUBER's performance in terms of request scheduling performance and, respectively, its infrastructure performance for various scenarios. The paper ends with acquired lessons during this work and our conclusions.

2 Background Information and Related Work

The thread shared by most Grid environments is *cooperative computing* [3, 49]. The goal of these systems is to provide large-scale, flexible and secure resource sharing among dynamic collections of individuals, institutions, and resources, also referred as virtual organizations (VOs) [4]. In such settings, users from multiple administrative domains pool available resources to harness their aggregate power, to benefit from the increased computing power and the diversity of these resources, especially when their applications are customized for a specific computing configuration.

2.1 The Controlled Resource Sharing Problem

Resource sharing within large distributed systems (i.e., P2P and Grid environments [13, 68]) that integrate participants and resources spanning multiple physical institutions raises challenging issues [2]. Physical institutions may wish to delegate to one or more participants the right to use certain resources subject to local preferences and various agreements; each participant then wishes to enable those resources subject to their own policy.

2.1.1 Usage Service Level Agreements (uSLAs)

We distinguish in our work between "resource *access* policies" and "resource *usage* service level agreements" (uSLAs). Resource access policies typically enforce authorization rules. They specify the privileges of a specific user to *access* a specific resource or resource class, such as submitting a job to a specific site, running a particular application, or accessing a specific file. Resource access policies are typically binary: they either grant or deny access. In contrast, resource uSLAs govern the *sharing* of specific resources among multiple groups of users. Once a user is permitted to access a resource via an access policy, then the resource uSLA steps in to govern *how much* of the resource the user is permitted to consume.

uSLAs [1, 5–8, 41, 47, 50] represent a novelty for Grids. The term uSLA was introduced to denote this new type of specific resource sharing for multi-type resources. In the networking domain, usage policies, the equivalent of uSLAs, are used to address the problem of bandwidth allocation based on specific rules. Such policies are specified by network administrators and contain the rules for handling different types of traffic. In this domain, a simple usage policy example is "Email traffic is only allowed from outside the company's servers only from a special mail gateway" [9–12, 48, 56].

2.1.2 Motivating Scenarios

In the following sub-sections we describe two scenarios that motivate controlled resource sharing. These scenarios are based on the real OSG/Grid3 environment and an external provider example that outsources service. In each scenario, common uSLA examples are provided.

2.1.2.1 OSG/Grid3 Scenario OSG/Grid3 comprises tens of institutions and hundreds to thousands of individual investigators that collectively control thousands of computers and associated storage systems [2, 14]. Each individual investigator and institution participates in, and contributes resources to multiple collaborative projects that vary in scale and formality. Figure 1 depicts a graphical representation of the OSG/Grid3 sites.

In this environment, several VOs exist that are composed of users with various common interests and applications. The most common ones are the USATLAS [15], Sloan Digital Sky Survey (SDSS) [16] and iVDGL [17] VOs. USATLAS VO users simulate the collisions of protons on protons at 14 TeV for the LHC experiment; applications are composed of hundreds of embarrassingly parallel programs with large input/output files. The SDSS VO users measure the distance to, and the masses of, clusters of galaxies in the SDSS data set; applications are composed again of many components, but in this case they have input/output dependencies [18] that can be represented using direct acyclic graphs (DAGs). The iVDGL VO performs protein sequence comparisons at increasingly larger scales (various size workflows in which a single BLAST job has an execution time of about an hour – the exact duration depends on the CPU, reads about 10–33 KB of input, and generates about 0.7–1.5 MB of output). OSG/Grid3 sites are sponsored either by different VOs or directly by the hosting institutions. Each site has uSLAs (expressed sometimes as "provide 30% of resources to USATLAS") that are enforced by means of a local resource manager (RM).

For the OSG/Grid3 scenario, some of the requirements include the provisioning of uSLAs capable of expressing situations both with and without contention. Usually, resource providers (universities and laboratories) and resource consumers (scientists from different domains) want access to these resources pooled together according to various needs. For example, before important conferences we have observed that Grid utilization increases and job higher contention occurs, while during holidays most of the time resources are free for long time intervals [19]. These observations presented by Iosup et al. [19] motivate our introduction of a uSLA that ensures "whenever there is no contention users can use a certain amount of resources, while when contention occurs, the resources are allocated according to predefined rules that provide the incentives for Grid participation". The following sharing example is widely accepted by each individual site (or with different variations in terms of the amount of resources provided) [20]: "there are three types of incoming jobs to balance: one from SDSS, one from ATLAS, and one from iVDGL. We call them SDSS-Prod, USATLAS-Prod, and IVDGL. We want SDSS-Prod and USATLAS-Prod to get an equal share of available CPUs, but IVDGL should get a small fraction, of the resources, if there is contention (a 4th of what the others get)". Figure 2 shows this controlled resource sharing scenario for the University of Chicago resources.

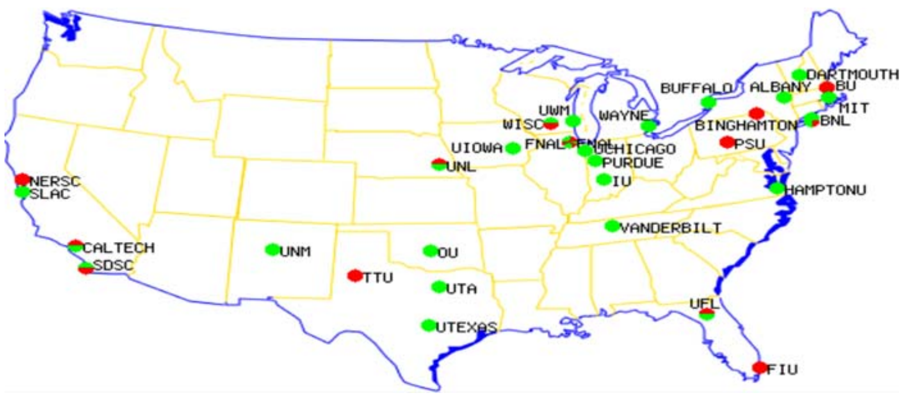


Fig. 1 Grid3 site and instantaneous utilization – the Grid Catalog Monitoring System (GridCat) snapshot

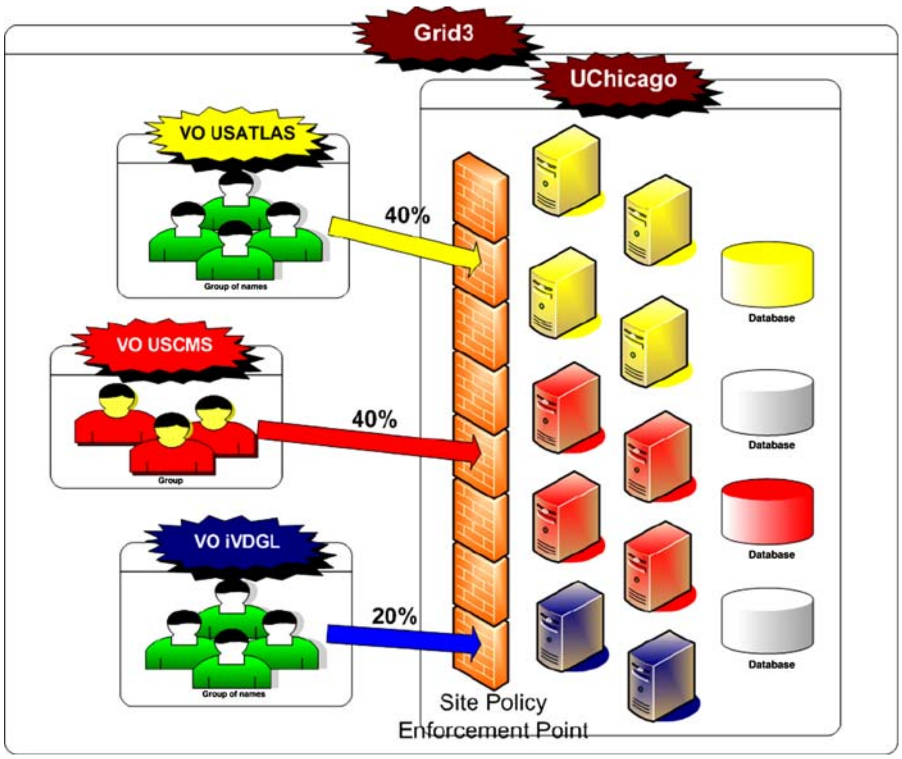


Fig. 2 Graphical view of the UChicago resourse sharing

2.1.2.2 Outsourcing Scenario In the second scenario, we envisage that a community outsources one or more services to reduce deployment and operational costs. Community members acquire resources and services from independent utility providers that specialize in providing those services (see Fig. 3). Service examples include scheduling prediction services, monitoring services (MonALISA [22]), or community authorization services (e.g., DOE certificate authority [23]).

In this scenario, the service provider requires that uSLAs express the amount of resources or services a consumer is entitled to: "I provide 1000 requests for service A from 7:00 to 16:00 for 1 month for any remote user from Grid3" or "I accept 1,000 requests for service A from 7:00 to 16:00 on date X for any remote user from Grid3" [24].

2.2 Related Work

Current solutions for controlling resource access in large scale distributed systems focus mainly on access control [25, 26, 40], but other groups have started pursuing various paths for controlled resource sharing [5, 6, 27–32]. Finer access control mechanisms focus on enabling resource providers in expressing additional conditions about access and in delegating partial control to other entities.

For example, a Community Authorization Service (CAS) for access control policy management allows resource providers to maintain ultimate authority over their resources, but spars them from day-to-day policy administration tasks (e.g. adding and deleting users, modifying user privileges) [25]. However, access control dictates the operations an entity is entitled to perform on certain resource without any further restrictions once access is granted. A policy based scheduling framework for Grid-enabled resource allocations is under development at the University of Florida [32]. This framework provides scheduling strategies that (a) control the request assignment to Grid resources by adjusting resource usage accounts or request priorities; (b) manage efficiently resources assigning usage quotas to intended users; and (c) supports reservation based Grid resource allocation.

Other methods focus on economic models or match-making for controlled resource provisioning. In such cases, mini-markets are built for resource brokering

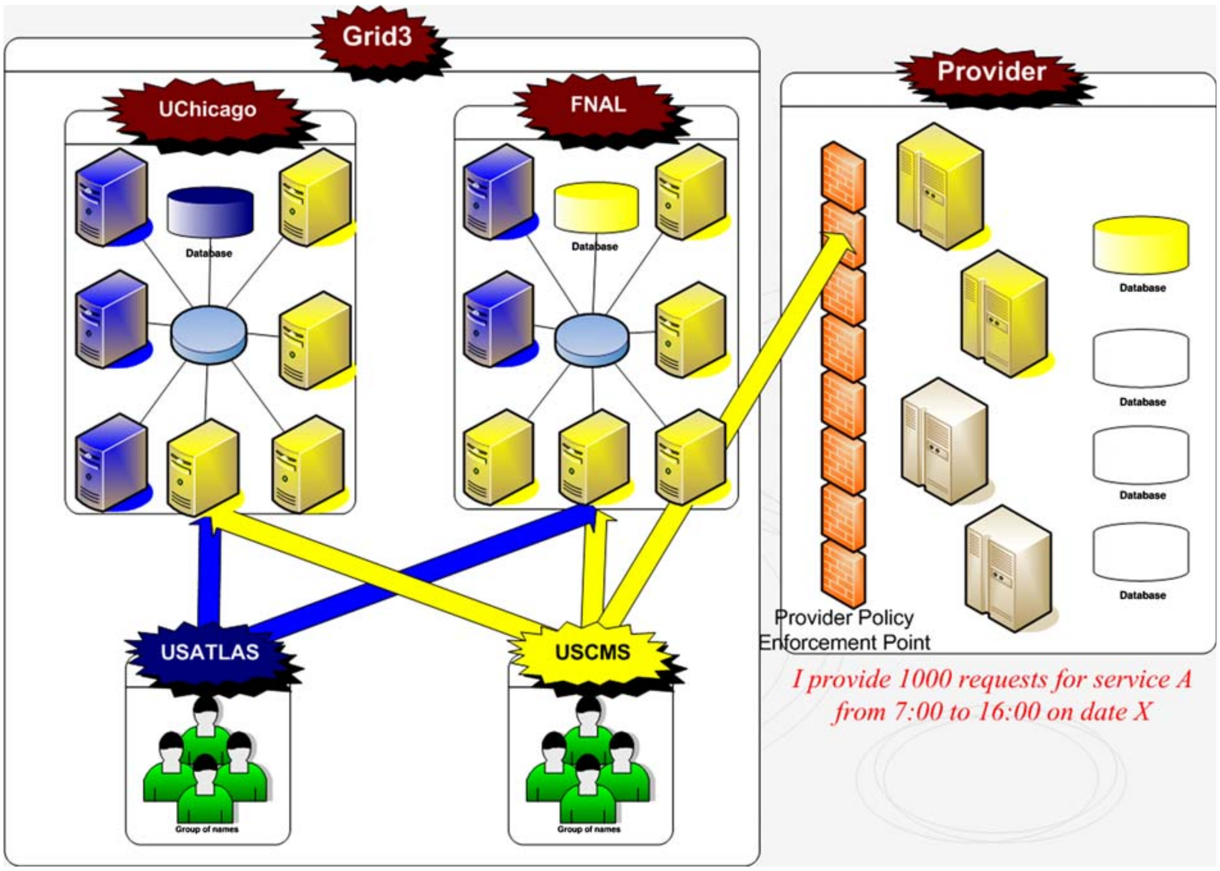


Fig. 3 Service outsourcing scenario

and provisioning [27]. The Grid Service Broker, a part of the GridBus Project, mediates access to distributed resources by (a) discovering suitable data sources for a given analysis scenario, (b) suitable computational resources, (c) optimally mapping analysis jobs to resources, (d) deploying and monitoring job execution on selected resources, (e) accessing data from local or remote data source during job execution, and (f) collating and presenting results. The broker supports a declarative and dynamic parametric programming model for creating Grid applications [33].

Service level agreements [5, 21] focus on establishing consumer-provider relationships concerning how resources must be consumed. Such relationships can be designed by bi-lateral rules that are driven by internal policies that govern any institution. Cremona is a such project developed at IBM as a part of the ETTK framework [31]. It is an implementation of the WS-Agreement specification and its architecture separates multiple layers of agreement management, orthogonal to the agreement management functions: the Agreement Protocol Role Management, the Agreement Service Role Management, and the Strategic Agreement Management. Cremona focuses on advance reservations, automated SLA negotiation and verification, as well as advanced agreement management.

GARA [29] represents modular and extensible system architecture for resource reservations to support end-to-end applications QoS. It offers a single interface for reserving different types of resources (network, CPU, disk), and focuses on provisioning generic solutions and algorithms for different types RMs (the heart of GARA). Reservations (and QoS) are specified through a specialized C-API, composed of client and Globus gatekeeper modules (admission control). GARA has a modular design, based on three levels: low-level QoS RMs, a QoS component for interfacing with the low-level RMs and provisioning the common interface, and high-level libraries (at the user level) for leveraging reservation synchronizations for user-level applications. The prototype supports only finite reservations, with three main classes of elements: reservations, resources, and QoS elements. All communications client-agreement provider are done through a specific API, and the underlying language for messages is RSL, with only one QoS quantitative parameter per reservation request.

SNAP [6] tries to overcome previous resource managements by providing a generic framework instead of considering specialized classes of resources. The generalized framework maps resource interactions onto a well defined set of platform independent service level agreements. SNAP represents the instantiation of this generalized framework, which provides a management infrastructure for such SLAs. However, the entire approach is generic enough and can be used beyond the Grid domain and Globus Toolkit in particular. The SLAs are categorized as: task service level agreements, resource level agreements, and binding service level agreements. A minimal number of scenarios are also introduced, to provide a basic understanding how SNAP should be used in practice: file transfer service, job staging with transfer service, resource virtualization. Also, for agreements realization, a supporting protocol is introduced that supports the SLA management.

2.3 The Motivation and Challenges for Our Work

None of the above systems fully address the problem of controlled resource management in distributed environments where many independent resource providers share their resources according to some local preferences and uSLAs. Mechanisms for supporting controlled resource sharing that allow cooperative systems to provision resources based on local preferences become a necessity in such cooperative environments.

Important challenges for uSLA-based resource management can arise in practice from: the lack of automated mechanisms for uSLA discovery, publication, or interpretation [34] to the complexity of the uSLA operations to be performed (to satisfy the requirements [35, 36]) of many resources and users.

Additionally, the increased scale of such distributed systems calls for minimizing the need for human supervision and for automating as many management tasks as possible. In a system with over 1,000 providers and consumers, new uSLAs will occur thousands of times more often than on a single system. In the same time, the complexity of necessary services for uSLA discovery, management and enforcement will increase with the scale of the system.

Increasing scale in large cooperative computing also makes performance and reliability challenging and centralized systems are unlikely to be suitable for these challenges. A single unified management approach over hundreds to thousands of consumers and providers can become a bottleneck in terms of both reliability and performance. Additionally, in a wide area network, where short and transient failures often occur, the centralized approach can become inaccessible for varying time periods.

3 The GRUBER Infrastructure

GRUBER was specifically developed to address the problem of uSLA discovery, management and enforcement in large Grid environments. GRUBER addresses issues regarding how uSLAs can be stored, retrieved, and disseminated efficiently in these types of distributed environments and has been implemented by means of both the Grid Services (OGSI [37]) and Web Services (WS [38]) versions of the Globus Toolkit (GT3, respectively, GT4). The main elements of GRUBER are:

(a) The *GRUBER specification language* represents the medium for uSLA specification. It supports well-defined semantics and syntax for allocation specification, as described next.
(b) The *GRUBER engine* implements various algorithms for detecting available resources and maintains a generic view of resource utilization in the Grid. Our implementation is a Grid service capable of serving multiple requests and based on all the features provided by the GT3 or GT4 container (authentication, authorization, state or state-less interaction, etc).
(c) The *GRUBER site monitor* is one of the data providers for the GRUBER engine. It is composed of a set of UNIX and Globus tools for collecting Grid status elements.
(d) The *GRUBER site selectors* are tools that communicate with the GRUBER engine and provide answers to the question: "Which is the best site at which I can run this job?" Site

selectors can implement various task assignment policies, such as round robin, least used, or last recently used task assignment policies.

(e) The *GRUBER queue manager* is a complex GRUBER client that must reside on a submitting host. It monitors VO policies and decides how many jobs to start and when. The GRUBER architecture is presented in Fig. 4.

Planners, work-runners, or application infrastructures invoke GRUBER site selectors to get site recommendation, while the GRUBER queue manager is respon-

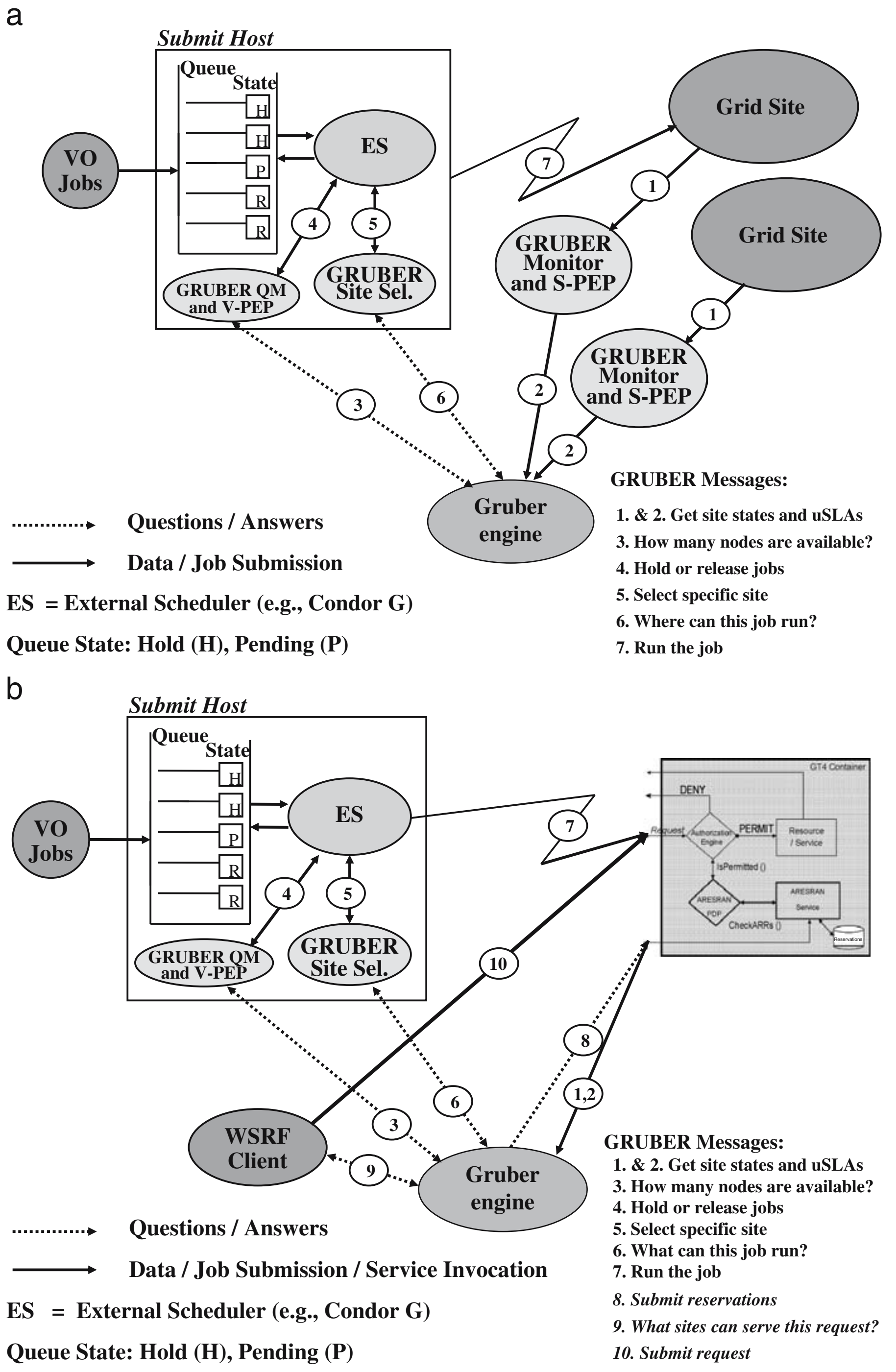


Fig. 4 GRUBER architecture: low-level resource (*left – A*) vs. service (*right – B*) brokering

sible for controlling job starting time. If the queue manager is not enabled, GRUBER becomes only a site recommender, without the capacity to enforce any usage SLA expressed at the VO level. The site level usage SLA is still enforced by limiting the choices a job can have and by means of removing a site for an already over-quota VO user from the list of available sites.

3.1 GRUBER Specification Language: Semantics and Syntax

Without a well-defined specification language for uSLAs, their interpretation may become misleading. We focus in the rest of this section on the GRUBER's language semantic and syntax and describe the four semantics for specifying resource usage constraints and give examples for the proposed schemas.

3.1.1 GRUBER uSLA Semantics

The uSLA semantics capture how controlled resource sharing is performed in the scenarios described in Section 2.1. They are generic enough to be also applied for any distributed environment. The proposed semantics are named after their goals: *no-limit*, *fixed-limit*, *extensible-limit*, and *commitment-limit* [43–46].

- The ***no-limit*** uSLA is a statement that specifies no limit in acquiring resources. Resources are acquired on a first come first executed basis.
- The ***fixed-limit*** uSLA specifies a hard upper limit on the fraction of resources $\mathbf{R_i}$ available to a $\mathbf{VO_i}$. A request to run a job is granted if this limit is not exceeded, and rejected otherwise. More precisely, a job requiring **J** resources is admitted if and only if $\mathbf{C_i+J \leq R_i}$, where $\mathbf{C_i}$ denotes the resources currently consumed by $\mathbf{VO_i}$ at the site. Note that an admitted job will always be able to run immediately, unless the resource owner oversubscribes resources, i.e., $\mathbf{\Sigma_i R_i > 1}$.
- The ***extensible-limit*** uSLA also specifies an upper limit, but this limit is enforced only under contention. Thus, under this SLA a job requiring **J** resources is admitted if $\mathbf{C_i+J \leq R_i}$ or $\leq \mathbf{C_{free}}$, where $\mathbf{C_i}$ and $\mathbf{R_i}$ have the same meaning as before, and $\mathbf{C_{free}}$ denotes the site's current unused resources. Note that because this policy allows VOs to consume more than their allocated resources, whether or not an admitted job can run immediately may depend on the site's preemption policy.

While the fixed or extensible-limit uSLA are sufficiently expressive for the OSG/Grid3 scenarios, their limitations become obvious when moving to the second scenario.

- The uSLA, the ***commitment-limit*** SLA, supports these more complex queries. It specifies two upper limits, an epoch limit $\mathbf{R_{epoch}}$ and a burst limit $\mathbf{R_{burst}}$, and specifies for each an associated interval, $\mathbf{T_{epoch}}$ and $\mathbf{T_{burst}}$, respectively. A job is admitted if and only if (a) the average resource utilization for its VO is less than the corresponding $\mathbf{R_{epoch}}$ over the preceding $\mathbf{T_{epoch}}$, or (b) there are idle nodes and the average resource utilization for the VO is less than $\mathbf{R_{burst}}$ over the preceding $\mathbf{T_{burst}}$. Both periods are modeled here as recurring within fixed time slots. A provider may grant requests above the epochal allocation if sufficient resources are available, but these resources can be preempted if other parties with appropriate allocations request those resources at a later stage. More precisely, any job accepted by the following algorithm is admitted, with the following definitions:

s	site
J	Job
R_{epoch}	Epoch Usage Policy for VO_i at site s
R_{burst}	Burst Usage Policy for VO_i at site s
BA_i	Burst Resource Usage for VO_i at site s
EA_i	Epoch Resource Usage for VO_i at site s
TOTAL	upper limit allocation on the site s

```
algorithm commitment-uSLA inputs
  (J, VO_i, s) returns accept/reject
# Case 1: site over-used by VO_i
1. if $EA_i > R_{epoch}$ then
2.     return reject
# Case 2: sub-allocated site
3. else if $\Sigma_k(BA_k) + J < TOTAL$ and
   $BA_i + J < R_{burst}$ then
4.     return accept
# Case 3: over-allocated site
```

```
5. else if $\Sigma_k(BA_k)$ = TOTAL and
   $BA_i + J < R_{epoch}$ then
6.     return accept
7. else
8.     return reject
```

This uSLA can be extended further by introducing an unlimited number of sharing intervals, which makes it generic enough to express any requirements in practice. I note that the fixed limit and extensible-limit uSLA can be expressed as particular cases of the commitment-limit uSLA.

Based on the UNIX quota system, the same four uSLAs can be implemented with success in the case of disk space. However, in this case once a file is saved at a site and the allocation is higher than the uSLA limit allows (extensible-limit and commitment limit cases), the space cannot be preempted without evicting the violating files to other sites. Our approach builds on the UNIX quota system, which prevents one user on a static basis from using more than his hard limit (but it still considers soft and hard limits similarly to the commitment limit). More precisely, for scheduling decisions a list of site candidates that are available for use by a VO_i for a job with disk requirements J is built by executing the following logic, with the following definitions:

s	Site Set
k	index for any VO != VO_i at site s
IP_i	Epoch uSLA for VO_i at site s
ISP_i	Instantaneous uSLA for VO_i at site s
IA_i	Instantaneous Resource Usage for VO_i at site s
TOTAL	Upper limit allocation on the site

```
algorithm commitment-uSLA_disk inputs
   (F, $VO_i$, s) returns accept / reject
# Case 1: over hard-limit site by $VO_i$
1. if $IA_i$ $IP_i$ for $VO_i$ at site s
2.     return reject
# Case 2: over soft-limit site by $VO_i$
3. if $IA_i > ISP_i$ and time < grace period
   for $VO_i$ at site s
4. if $\Sigma_k(IA_k)$ < s.TOTAL-J && $IA_i + J < IP_i$
   then
6.     return reject
# Case 3: un-allocated site
7. if $\Sigma_k(IA_k)$ < s.TOTAL-J && $IA_i + J < IP_i$
   then
8.     return accept
```

The final type of resources we consider is a Grid service (a high-level resource). Because Grid services are difficult to quantify in term of their utilization (a weather service and a matrix multiplication service are difficult to compare in terms of resource consumption without a through service performance model analysis), we maintain the CPU semantics unchanged for the uSLAs, but with different utilization metrics: instead of CPU utilization, the number of requests a client can perform on a certain service is considered for the uSLA algorithms. While this approach may seem an over-simplification, the end result is similar: a service provider states by means of the uSLAs how many requests a certain consumer can perform on its resources.

Based on this approach, the algorithm to supports controlled Grid service sharing is identical with the one for controlled CPU sharing. We present a variation of the previous algorithm that allows advance service reservations (a request can be sent in well in advance of its starting time). It applies a pre-specified uSLA on each "future" sub-intervals resulted from a variation in terms of either service requests or allocations. The algorithm for accepting new advance reservations is introduced next, where:

R	resource request
A_j	allocated resources
R_j	requested resources
Allocations	set of accepted allocations
Requests	set of already accepted requests

```
algorithm fixed-uSLA_service (R)
   returns accept/reject
 1. response = accept
# Stores availability on the considered
   interval
 2. S = empty
# Identify all requests overlapping
   current request
# (save their start/end times and
   requested quantities)
 3. foreach $R_j$ in Requests do
 4.     if $R_j$ time overlaps R time then
 5.      save S, R_j.start, + R_j.attributes
 6.      save S, R_j.stop, - R_i.attributes
 7.    fi
 8. done
# Identify all allocations overlapping
  current request
# (update accordingly previous values
  with these allocations)
 9. foreach A_j in Allocations do
10. foreach T_j in S do
11.     if A_j overlaps T_j then
12.         save S, T_j, + A_j.attributes
13.     fi
14.   done
15. done
# Check constraints (available resources)
16. compute Request = R.attributes
17. foreach Availability in S do
# Apply the according uSLA algorithm
  (hard limit example here)
18.   if (Request > Availability)
      response = reject
19. done
20. if response == accept then
21.   update accordingly one of the
      overlapping uSLAs
22.   add R to Requests
23. fi
24. return response
```

3.1.2 GRUBER uSLA Syntax

We have considered two syntaxes for uSLAs: a simpler one based on allocations and one based on WSLA.

3.1.2.1 Tuple-based Syntax Our starting point for the first approach is the Maui [51] syntax for specifying allocations. Maui supports three types of fair share limits: "at least limit", "average limit" and "at most specification". In the first case, when the utilization for a group goes below the limit, the mechanism increases the priority of the jobs from that group (expressed as a real number preceded by a "+"). In the second case, whenever the utilization for a group is different from the limit, the fair share mechanism either increases the priority (for under-utilization) or decreases the priority (for over-utilization) of jobs from that group. In the final case, when the utilization for a group goes above the limit, the fair share mechanism decreases the priority of jobs from that group (expressed as a real number preceded by a "–"). Our first syntactic form is represented as a set of *allocations* of the form:

```
 <resource-type, provider, consumer,
start, epoch-alloc, burst-alloc>
```

where:

```
resource-type ::=[ CPU | NET | STORAGE ]
provider ::=[ site-name | vo-name ]
consumer ::=[ vo-name | (vo-name,
               group-name) | ANY]
start ::= date-time | time | *
epoch-alloc ::= (interval | *,
                   percentage) | -
burst-alloc ::= (interval | *,
                   percentage) | -
ANY ::= matches any name
* ::= means instantaneous
- ::= means not specified
```

However, this syntax has its limitations for expressing sharing rules about resources of different types. First, this syntax does not provide a mechanism for specifying monitoring requirements. Second, it does not support the specification of complex conditions (AND, OR, etc.).

3.1.2.2 Schema-based Syntax We considered as a second approach, a uSLA syntax based on the WS-Agreement specification, to take advantage of its high-level structure SLA specification and of available parsers. The objective of a WS-Agreement specification is to provide standard means for establishing and monitoring service agreements. The specification draft comprises three major elements: a description format for agreement templates, a basic protocol for establishing agreements, and an interface for monitoring agreements at runtime.

For this syntax, we use a schema that includes from the WS-Agreement specification support for resource monitoring metrics and goal specifications [31, 43, 52]. The resource monitoring metrics describe how various utilizations must be measured or how these quantities should be collected from an underlying monitoring system. A goal specification provides support for describing the targeted allocations in a form that can be parsed by automated agents. The other elements (i.e., obligations and handlings violation) were considered beyond the scope and capacity of current site and VO RMs.

The schema of this grammar is described next. First, the monitoring metric element defines how a certain resource metric required for a guarantee should be measured.

```
<!- MonitoredMetric ->
<xsd:complexType name="Monitored-
  MetricType">
  <xsd:attribute name="name"
    type="xsd:string" />
  <xsd:attribute name="method"
    type="xsd:string" />
  <xsd:attribute name="type"
    type="xsd:string" />
  <xsd:attribute name="interval"
    type="xsd:integer" />
  <xsd:attribute name="notification"
    type="xsd:boolean" />
</xsd:complexType>
```

The next element of the grammar, MonitoredType, describes the entire list of monitored metrics required in enabling the considered uSLA. This list can have zero or more of required metrics that must be monitored.

```
<!-- Monitored Type -->
<xsd:complexType name="MonitoredType">
  <xsd:sequence>
    <xsd:element name="MonitoredMetric"
      type="MonitoredMetricType"
      minOccurs="0" />
</xsd:sequence>
  <xsd:attribute name="name"
    type="xsd:string" use="optional" />
</xsd:complexType>
```

The precondition element identifies of the entity for which the uSLA is defined and the name of the provider:

```
<!-- Precondition -->
<xsd:complexType name="PreconditionType">
  <xsd:sequence>
      <xsd:element name="consumer"
        type="xsd:string" minOccurs="0" />
    <xsd:element name="provider"
      type="xsd:string" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
```

The goal element describes the conditions under which a resource is provided. It uses the constraint element for defining conditions (with *LessEqual*, *Equal* and *GreaterEqual* corresponding to the semantics introduced earlier for – , <*space*>, + signs, while *Range* has a special meaning for time specifications):

```
<!-- Goal -->
<xsd:complexType name="GoalType">
  <xsd:sequence>
    <xsd:element type="Constraint-
      Type" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>
<!-- Constraint -->
<xsd:complexType name="ConstraintType">
  <xsd:attribute name="type"
    type="xsd:string"
    values="LessEqual, Equal, GreaterEqual,
      Range" />
  <xsd:element name="Metric"
    type="xsd:string" />
  <xsd:element name="Value"
    type="xsd:literal" />
</xsd:complexType>
```

Similarly to a MonitoringType element, a GuaranteeElement contains a list of all guarantees that a resource provider agrees to when providing the resources.

```
<!-- Guarantee Type -->
<xsd:complexType name="GuaranteeType">
  <xsd:sequence>
    <xsd:element name="Precondition"
      type="PreconditionType"
      minOccurs="0" maxOccurs="1" />
    <xsd:element name="Goal"
      type= "GoalType" minOccurs="0"
      maxOccurs="1" />
  </xsd:sequence>
  <xsd:attribute name="name"
    type="xsd:string" use="required" />
</xsd:complexType>
```

The final element of the grammar is the uSLA element, which is composed of several monitored and guarantee elements.

```
     <!-- usage SLA -->
<xsd:complexType name="uSLA">
  <xsd:attribute name="Monitored"
   type="MonitoredType" minOccurs="1"/>
  <xsd:attribute name="Guarantee"
   type="GuaranteeType" minOccurs="1"/>
  <xsd:attribute name="name"
   type="xsd:string" use="required"/>
</xsd:complexType>
```

The OSG/Grid3 example is represented using this syntax as follows. Three metrics are monitored (by means of MonALISA, for example, and these values are retrieved from a certain URL): *CPUBurst-Met-USATLAS*, *CPUBurst-Met-SDSSS* and *CPUBurst-Met-iVDGL*.

```
<uSLA name="Grid3 uSLA (Scenario 1)">

  <!-- Define Monitored Metrics (and
   acquisition mechanism) -->
  <Monitored>
   <MonitoredMetric name="CPUBurst-
     Met-USATLAS"
       method="http://URL/CPU?vo=
        USATLAS&t=5"
       interval="5" type="%"
        notification="true" />
   <MonitoredMetric name="CPUBurst-
     Met-SDSS"
       method="http://URL/CPU?vo=
        SDSS&t=5"
       interval="5" type="%"
        notification="true" />
   <MonitoredMetric name="CPUBurst-
     Met-iVDGL"
       method="http://URL/CPU?vo=
        iVDGL&t=5"
       interval="5" type="%"
        notification="true" />
  </Monitored>

  <!-- USTALAS minimal allocation -->
  <Guarantee name="CPUBurst-G-USATLAS">
    <precondition usage="required">
      <consumer name="USATLAS-Prod" />
      <provider name="UChicago" />
    </precondition>
    <goal usage="required">
     <Constraint type="GreaterEqual">
       <Metric value="CPUBurst-
         Met-USATLAS" />
       <Value value="40" />
     </Constraint>
    </goal>
  </Guarantee>

  <!-- SDSS minimal allocation -->
  <Guarantee name="CPUBurst-G-SDSS">
    <precondition usage="required">
     <consumer name="SDSS-Prod" />
     <provider name="UChicago" />
    </precondition>
    <goal usage="required">
     <Constraint type="GreaterEqual">
       <Metric value="CPUBurst-
         Met-SDSS" />
       <Value value="40" />
     </Constraint>
    </goal>
  </Guarantee>

<!-- iVDGL minimal allocation -->
<Guarantee name="CPUBurst-G-iVDGL">
    <precondition usage="required">
      <consumer name="iVDGL" />
      <provider name="UChicago" />
    </precondition>
    <goal usage="required">
      <Constraint type="GreaterEqual">
       <Metric value="CPUBurst-
         Met-iVDGL" />
       <Value value="20" />
      </Constraint>
     </goal>
  </Guarantee>
</uSLA>
```

3.2 The GRUBER Engine

The GRUBER engine represents the main component of the brokering infrastructure. We often call throughout this paper an engine instance a decision point (DP). Usually, all the other elements in the brokering infrastructure rely on communicating with one DP to perform their operations. It maintains a view of the available resources at each Grid site and invokes various algorithms for deciding the resource availabilities. Two main resource types are considered for brokering: low-level resources (e.g., CPU, disk, and network) and services (e.g., any Grid-enabled service). The GRUBER algorithms address differently these two resource types, conditioned mainly by the local site managers in each case (Condor [53], PBS [54], etc. for low-level resources vs. ARESRAN [24], SAML [55], etc for services).

3.2.1 CPU Brokering

GRUBER decides which consumers are *best* for a request from a CPU availability point of view by implementing the following logic:

- If there are fewer waiting jobs at a site than available CPUs or an extensible-limit uSLA is in place, then GRUBER assumes the job will start right away if an extensible-limit uSLA is in place.
- If there are more waiting jobs than available CPUs or if an extensible-limit uSLA is not in place, then GRUBER determines the VO's allocation, the number of jobs already scheduled, and the RM type. Based on this information, if the VO is:
  - under its allocation, GRUBER assumes that a new job can be started (in a time that depends on the local RM type: for example, in the Condor RM case a 15 min delay period might be enforced before a newly free machine can be acquired)
  - over its allocation, GRUBER assumes that a new job cannot be started (the running time is unknown for the jobs already running)

More precisely, for any job placement CPU-based decision a list of available sites is built and provided under the following algorithm to find those sites in the site set G that are available for use for job J (J keeps place for job characteristics as well in the following algorithm) from the VO number I (with the following definitions):

```
G  Grid Site Set ;
S  Matching Site Set;
```

```
algorithm get-avail-sites_cpu inputs
   (sites G, VO_i, job J) returns S
1. S = empty
2. for each site s in G do
# Apply one of the algorithms
   introduced in Section 4.1
3.     if commitment-uSLA_CPU
       (J, VO_i, s) == accept then
4.      add (s, S)
5.     else
6.        next
8. return S
```

3.2.2 Disk-space Brokering

Disk space brokering introduces additional complexities in comparison to CPUs, if job files cannot be fetched from some generic replication service. If an entitled-to-resources job becomes available, it is usually possible to delay scheduling other jobs, or to preempt them if they are already running. In contrast, a file that has been staged to a site cannot be "delayed," it can only be deleted. Yet deleting a file that has been staged for a job can result in livelock, if a job's files are repeatedly deleted before the job runs. So far, we have considered a UNIX quota-like approach. Usually, quotas just prevent one user on a static basis from using more than his limit. There is no adaptation to make efficient use of disk in the way a site CPU RM adapts to make efficient use of CPU (by implementing more advanced disk space management techniques). The set of disk-available site candidates is combined with the set of CPU-available site candidates and the intersection of the two sets is used for further scheduling decisions. More precisely, for scheduling decisions a list of site candidates that are available for use by a VO_i for a job with disk requirements J, in terms of provided disk space, is built by executing the following logic, with the following definitions:

```
G  Grid Site Set ;
S  Matching Site Set ;
F  File requirements for Job J
```

```
algorithm get-avail-sites_disk inputs
   (F, VO_i, G) returns S
1. S = empty
2. for each site s in G do
# Apply one of the algorithms
   introduced in Section 4.1
3.     if commitment-uSLA_disk
       (F, VO_i, s) == accept then
```

```
4.        add (s, S)
5.    else
6.        next
7. return S
```

3.2.3 Service (Higher-level Resource) Brokering

For service brokering, GRUBER uses an internal representation based on a variable time-slot representation, as introduced by Wolf et al. [57], and each uSLA has assigned its own time intervals. This structure allows for an unlimited number of uSLAs and unlimited or unknown periods of time intervals. The processing logic we propose and evaluate is based on the following algorithms, where:

G	Grid Site Set ;
S	Matching Site Set ;
R	Request ;

```
algorithm get-avail-sites_service
  inputs (R, $VO_i$, G) returns S
1. S = empty
2.    for each site s in G do
# Apply one of the algorithms
  introduced in Section 4.1
3.       if fixed-uSLA_service
         (R) == accept then
4.          add (s, S)
5.       else
6.          next
7. return S
```

3.3 Helpers and Provisioning Tools

In addition to the engine, the GRUBER infrastructure relies on various site monitors and queue managers. We describe in this section the current solutions already implemented and available for integration whenever GRUBER must be deployed.

3.3.1 uSLA Enforcers (PEPs) and Observers (POPs)

We describe here three solutions for site uSLA management and enforcement as implemented for GRUBER. The first solution considers the case of simple RMs unable to handle any type of arbitration among concurrent requests for resources. The second solution instead takes in consideration advanced site managers capable of enforcing complex uSLAs that must be discovered and published mainly at the Grid level. The final solution targets instead Grid service management.

Solution 1 (Stand-alone S-PEP) Our first solution does not require a usage policy-cognizant cluster RM. It works with any primitive batch system that has at least the following capabilities: provide accurate usage and state information about all scheduled jobs, job start/stop/held/remove capabilities, and running job increase/decrease priority capabilities. The S-PEP sits at the level of the local scheduler(s), checks continuously the status of jobs in all queues and invokes management operations on the cluster RM when required to enforce policy. In more detail, the S-PEP gathers site uSLAs, collects monitoring information from the local schedulers about cluster usage, computes CPU-usage parameters, and sends commands to schedulers to start, stop, restart, hold, and prioritize jobs. Our approach provides priority-based enforcements. The processing logic of our prototype S-PEP is based on the algorithms presented below, with the following definitions:

EP_i	Epoch allocation policy for VO_i
BP_i	Burst allocation policy for VO_i
Q_i	Set of queues with jobs from VO_i
BA_i	Burst Resource Allocation for VO_i
EA_i	Epoch Resource Allocation for VO_i
TOTAL	possible allocation on the site
Over-quota job	job of VO_j

```
procedure s-pep
1. while (true) do
2.     sleep N # (seconds)
3.     foreach $VO_i$ with $EP_i$
# Case 1: available and $BA_i < BP_i$
4.     if $\Sigma(BA_k)$<TOTAL & $BA_i$<$BP_i$ &
       $Q_i$ has jobs then
```

```
5.        release job J from some $Q_i$#
          (e.g., FIFO scheduling policy)
# Case 2: res. contention: fill $EP_i$
6.        else if $\Sigma(BA_k)$ == TOTAL & $BA_i < EP_i$
          & $Q_i$ has jobs then
7.            if j exists & $BA_j >= EP_j$ then
8.              suspend an over-quota job $Q_j$
9.            release job J from some $Q_i$# (e.g.,
              FIFO scheduling policy)
10.       foreach $VO_i$ with $EP_i$
11.           if $EA_i > EP_i$ then
12.             suspend jobs for $VO_i$
                from all $Q_i$
```

As a further clarification, BA or EA represents the share actually utilized by a VO. BP or EP represents upper values for these utilized shares. When BP or EP increases for example, the VO is entitled to more shares starting with the moment of the change. An important novelty of this S-PEP over a cluster RM is its capability to keep track of jobs under several RMs and to allow the specification of more complex usage policies without the need to change the actual cluster RM implementation.

Solution 2 (Policy-aware Scheduler) The second solution was developed and implemented with success in the context of the Grid3 environment. We decoupled the functionalities of the S-PEP in two major components and mapped to existing solutions: a standalone *site policy observation point* (S-POP) and the cluster RMs for resource allocation control. In this case, we assume that the cluster RM is able to enforce by itself the desired usage policies, which are provided by means of our S-POP module. Examples of such cluster RMs are Condor [58], Portable Batch System [59], and Load Sharing Facility [60], widely used on Grid3 [2]. The S-POP's main functions are to optionally provide and to translate to/from the RM local usage policies, and to monitor the actual resource utilization. An advantage of this approach is that the local administrators do not have to use an additional Grid component in managing their clusters.

At the RM level (sites), *owners* state how their resources must be allocated and used by different *consumers*. This statement represents the *high-level GOAL an owner* (*MP*) *wants to achieve*. Site administrators map MPs to different software RMs' semantics/syntax. The end product is a set of RM configuration files, named *the local POLICY* or *system configuration* (*SC*). For automated consumption, site policies are translated from SCs by automated tools into an *abstract usage policy* (*AP*) set, i.e., the uSLA syntax/semantic described above. SC descriptions are collected from the site RM configurations, filtered and, after translation, published through a specific monitoring system, e.g., VO-Ganglia [61], MonALISA [22, 34] or GRUBER-SiteMonitor [43, 62].

We have proposed three levels of description for the statement "site X gives ATLAS 30% over a month:"

- MP: a description of a site manager's policy for the site, e.g., MP (VOs) = "give ATLAS 30% over a month". I assume that simple English statements describe the MP set (site level);
- SC: an RM configuration: SC (VO) = <number of nodes, scheduler-type, scheduler-config>, which is written by the site administrators during the RM configuration process (site level);
- AP: An SC (VO) translated into a uSLA representation: *AP (VO, Site)* expresses *SC (VO)* in a scheduler-independent format and is published through a monitoring tool for resource brokering or other automated tools (Grid level).

The key point is determining how an SC maps to an AP. We have achieved this part by providing specialized SC-translators for each type of SC supported by a specific RM. The translator parses the configuration files or queries the resource provider RM, and outputs the resulting configuration directly into AP form. The information flow for this process is captured in a graphical way in Fig. 5.

Solution 3 (*Stand-alone Service S-PEP*) For Grid services, the solution adopted for uSLA management is the ARESRAN prototype. ARESRAN is a GT4 service for uSLA and reservation management, specification and enforcement at the level of a single site based on the Globus technology [38]. ARESRAN prototype is based on the authorization schemes implemented by the GT4, the so called *Policy Decision Point* (PDP). GT4 allows a chain of PDPs to be configured internally, with each PDP evaluating to an independent decision [63]. The authorization engine of the framework then uses a policy combination algorithm to combine the decisions returned by

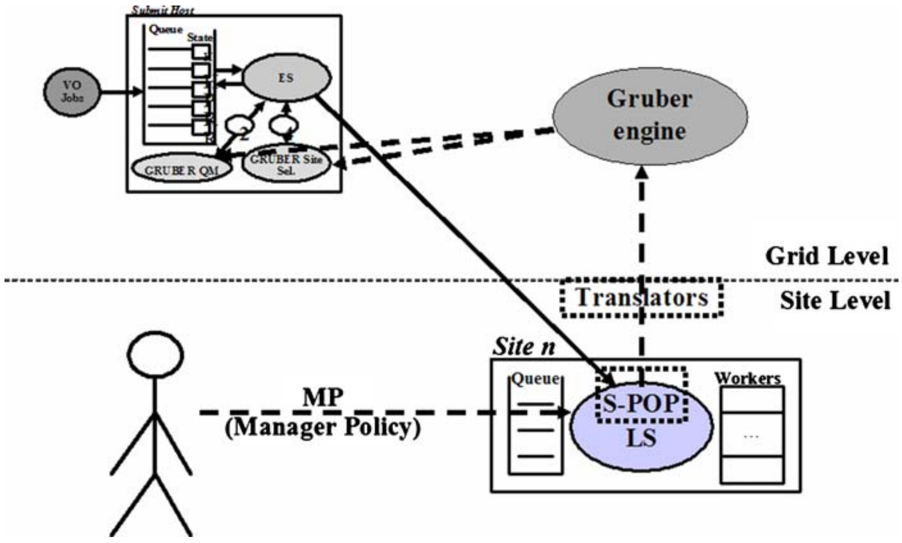


Fig. 5 Correlations MP, SC, and AP

each PDP to render a final decision. We have developed two specific ARESRAN PDPs – one for managing service reservations and one for lower level resource reservations such as compute nodes that are managed by the WS-GRAM service.

The overall ARESRAN architecture is described in Fig. 6. Whenever a service request comes in for a service managed through ARESRAN, its PDP steps in and verifies if the request is acceptable or not. Now, at the Grid level, GRUBER collects uSLAs from all individual ARESRAN services and provides brokering services for the consumers.

The main components here are the ARESRAN Service, the ARESRAN PDPs and the ARESRAN Reservation Database. These three components inter-

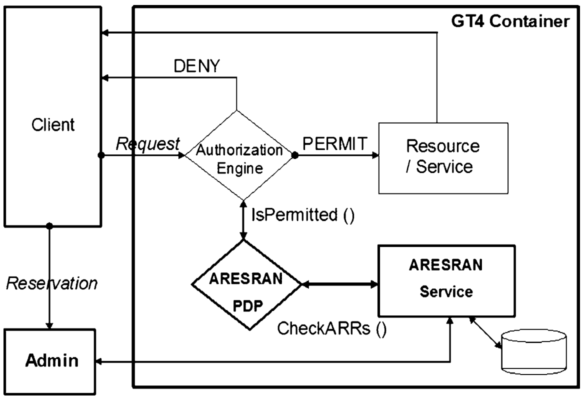


Fig. 6 ARESRAN architecture

communicate in order to ensure that requested resources or services are used accordingly. The specific details of these components are:

- *Service*: represents the reservation and uSLA engine of our prototype. Every time a new reservation is requested, the engine is invoked to verify if the new reservation can be honored. The verification procedure takes into account various information from the ARESRAN database and returns either ACCEPT, DENY or PROBABLY. If the reservation request is accepted, it is also saved to the ARESRAN database
- *Database*: stores reservations, uSLAs as well as information about the requests in progress. In this manner, ARESRAN has a complete view about the utilization of the services and resources that it manages. So far, the database is implemented only in memory, but future enhancements target the usage of a specific database system for this part of ARESRAN. Every time a reservation is served, various statistics are also saved, such as: request time, running time, remote client, etc. Based on such information, we believe the ARESRAN engine could later perform probabilistic reservations
- *PDPs*: authorize based on the rules stored in database about various services and resources. Each PDP returns either *REJECT* or *ACCEPT*. When a request is accepted (associated with a reservation), the database is also updated to reflect the current state of the managed resources

3.3.2 Queue Managers and VO-level uSLA Enforcement

GRUBER queue managers reside at the submission hosts and are responsible for determining how many jobs per VO or VO group can be scheduled at a certain moment in time and when to release them. Usually, a VO planner is composed of a job queue, a scheduler, and job assignment and enforcement components. Here, the last two components are part of GRUBER and have multiple functionalities. Queue manager components answer: "How many jobs should group Gm of VOn V be allowed to run?" and "When to start these jobs?" The queue manager is important for uSLA enforcement at the VO level and beyond. This mechanism also avoids site and resource overloading due to un-controlled submissions. The GRUBER queue manager implements the following algorithm (with the assumption that all jobs are held initially at the submission host), with the following definitions:

```
J  =  Job Id ;
Q  =  Job Queue ;
S  =  Site Set ;
G  =  All Site Set ;
VO =  Mapping Function jobId -> VO

procedure v-pep
1. while (true) do
2.     sleep N # (seconds)
3.     if Q != empty then
4.         get J from Q
5.     else
6.         next
7.     S = get-avail-sites(G, Vo(J), J)
8.     if S != empty then
9.         release J from Q
```

3.3.3 Site Selectors

While GRUBER provides mainly brokering services in distributed environments, we have extended it to provide also primitive scheduling services. By introducing site selector tools, the GRUBER infrastructure is able to select from the set of available sites the best site for running a job according to various scheduling policies. The site selector components answer the question: "Where is best to run next?" These site selectors are invoked by any automated tool requiring a list of available sites for scheduling (e.g., Euryale, KOALA or WMS).

The four main policies implemented by GRUBER's site selectors are: random assignment (G-RA), round-robin assignment (G-RR), least-recently-used assignment (G-LRU), and last-used assignment (G-LU). Each of these implements the scheduling policy described by their name. "GRUBER observant" (G-Obs) is a custom site selector that associates a further job to a site each time a job previously submitted to that site has started, but without bypassing the uSLA enforced at the site. In effect, this fifth site selector fills up what a site provides by associating jobs until site's limit is reached.

3.4 GRUBER Extensions

Several extensions were performed over time for GRUBER. The most notable ones are described in this section: high-level resource brokering, distributed uSLA management, client scheduling among the decision points and human interfacing for easy system checking.

3.4.1 DI-GRUBER (DIstributed GRUBER)

Managing uSLAs within environments that integrate participants and resources spanning many physical institutions can become a challenging problem in practice. A single unified uSLA management decision point providing brokering decisions over hundreds to thousands of jobs and sites can become a bottleneck in terms of reliability as well as performance. DI-GRUBER, an extension of the GRUBER infrastructure, was developed as a distributed Grid uSLA-based resource broker that allows multiple decision points to coexist and cooperate in real-time. DI-GRUBER targets to provide a scalable management service with the same functionalities as GRUBER but in a distributed approach. [64] It is a two layer resource brokering service (see Fig. 7), capable of working over large Grids, extending GRUBER with support for multiple scheduling decision points that cooperate by periodically exchanging various status information.

3.4.2 WS-Index Service Support

As already described, the ability to bring online a decision point is important in a large and dynamic

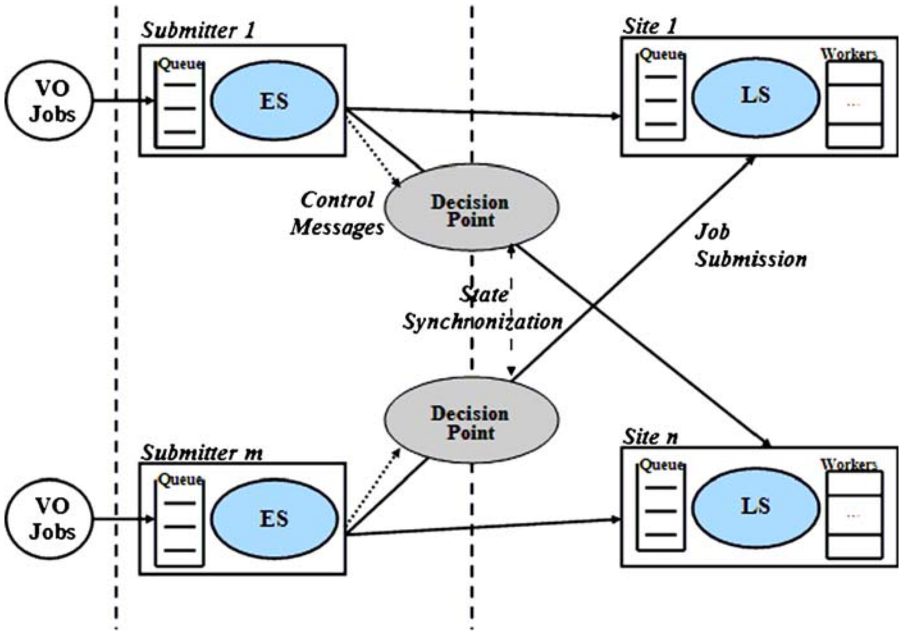
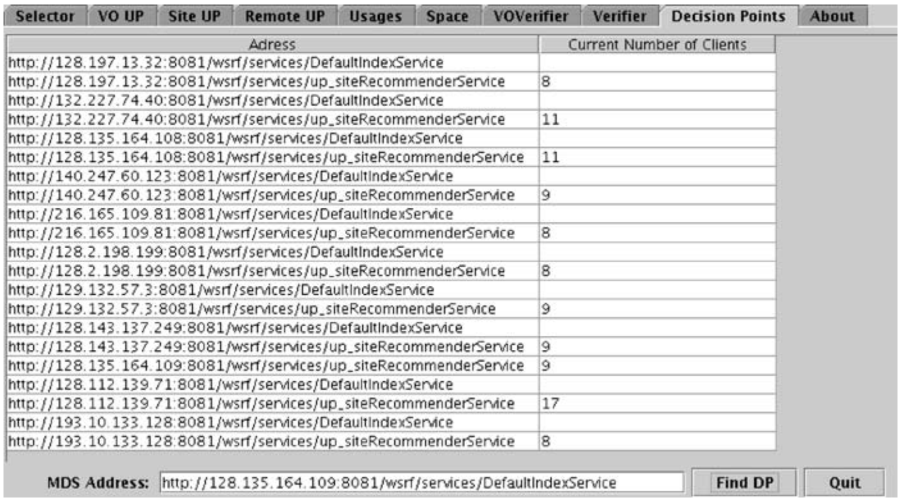


Fig. 7 DI-GRUBER architecture

Grid. Our previous work on GRUBER and DI-GRUBER [64] assumed a static environment, and hence did not offer flexibility for dynamic environments. The DI-GRUBER implementation evolved to support dynamic environments through the use of the WS-Index Service [39] provided with GT4; our solution uses the functionalities offered by the WS-Index Service for service registering and querying. In our implementation, each DI-GRUBER decision point registers with a predefined WS-Index Service at startup, while it is automatically deleted when it vanishes. Further, all decision points and clients can use this registry to find information about the existing infrastructure and select the most appropriate point of contact. Whenever we use the term "most appropriate", we refer to metrics such as load and number of clients already connected. A decision point scheduling policy can easily be incorporated by each client, the default one being *least-used* policy (in terms of number of clients). In Fig. 8 is presented a view achieved by means of the GRUBER graphical console. Now, whenever a new client boots (at a submission point), it can easily find which decision point is most appropriate. Also, whenever a decision point stops



Adress	Current Number of Clients
http://128.197.13.32:8081/wsrf/services/DefaultIndexService	
http://128.197.13.32:8081/wsrf/services/up_siteRecommenderService	8
http://132.227.74.40:8081/wsrf/services/DefaultIndexService	
http://132.227.74.40:8081/wsrf/services/up_siteRecommenderService	11
http://128.135.164.108:8081/wsrf/services/DefaultIndexService	
http://128.135.164.108:8081/wsrf/services/up_siteRecommenderService	11
http://140.247.60.123:8081/wsrf/services/DefaultIndexService	
http://140.247.60.123:8081/wsrf/services/up_siteRecommenderService	9
http://216.165.109.81:8081/wsrf/services/DefaultIndexService	
http://216.165.109.81:8081/wsrf/services/up_siteRecommenderService	8
http://128.2.198.199:8081/wsrf/services/DefaultIndexService	
http://128.2.198.199:8081/wsrf/services/up_siteRecommenderService	8
http://129.132.57.3:8081/wsrf/services/DefaultIndexService	
http://129.132.57.3:8081/wsrf/services/up_siteRecommenderService	9
http://128.143.137.249:8081/wsrf/services/DefaultIndexService	
http://128.143.137.249:8081/wsrf/services/up_siteRecommenderService	9
http://128.135.164.109:8081/wsrf/services/up_siteRecommenderService	9
http://128.112.139.71:8081/wsrf/services/DefaultIndexService	
http://128.112.139.71:8081/wsrf/services/up_siteRecommenderService	17
http://193.10.133.128:8081/wsrf/services/DefaultIndexService	
http://193.10.133.128:8081/wsrf/services/up_siteRecommenderService	8

MDS Address: http://128.135.164.109:8081/wsrf/services/DefaultIndexService

Fig. 8 Decision points allocation interface (PlanetLab experimental testbed)

responding to a client, this client automatically queries the registry and selects a different point of contact.

We consider that this approach is less error-prone than the static solution, and, additionally, it offers the support for dynamically bootstrapping new decision points whenever new ones register with the WS-Index Service. While we have not implemented this facility 100% yet, a human operator can easily perform such an operation (starting a new GT4 container with a DI-GRUBER decision point web service deployed). Additionally, if a pool of decision points are maintained in background and forced to register with the WS-Index Service only when needed, the operation would be 100% automated.

While dynamic DI-GRUBER decision point bootstrapping might be difficult to automate in a generic environment, the solution we have devised for such environments is semi-automatic. Every time a client fails to communicate or to connect with a decision point, it registers with the WS-Index Service a request fault. These faults are then used by a human operator in order to bring up new DI-GRUBER instances and stabilize the brokering infrastructure whenever required. As future work, we envisage to fully automate such operations by means of Grid technologies where possible. Such faults can be consumed by a specialized entity that can dynamically start new decision points by means of WS-GRAM [38]. For example, in the OSG/Grid3 scenarios considered here, whenever the condition for bringing up a new decision occurs, a special job is submitted to a site and a new container is started. In a more specialized context, a dedicated pool of nodes can be used for bringing up such decision points and really used only when necessary. For the remaining time, the dedicated pool might be used for other Grid specific operations.

3.4.3 Human Interfacing

Firstly, accurate monitoring is important if we are to understand how the framework actually performs in different situations (the *verifier* concept introduced by Dumitrescu et al. [1]). As a first step towards this goal, we have developed mechanisms for measuring how resources are used by each VO and by the Grid, overall. The monitoring tool built for GRUBER is a graphical interface able to present the current allocations and uSLAs at each decision point and over in the managed Grid infrastructure. This interface connects to a decision point, collects the local or generic view and presents it in an easy to visualize mode (see Fig. 9).

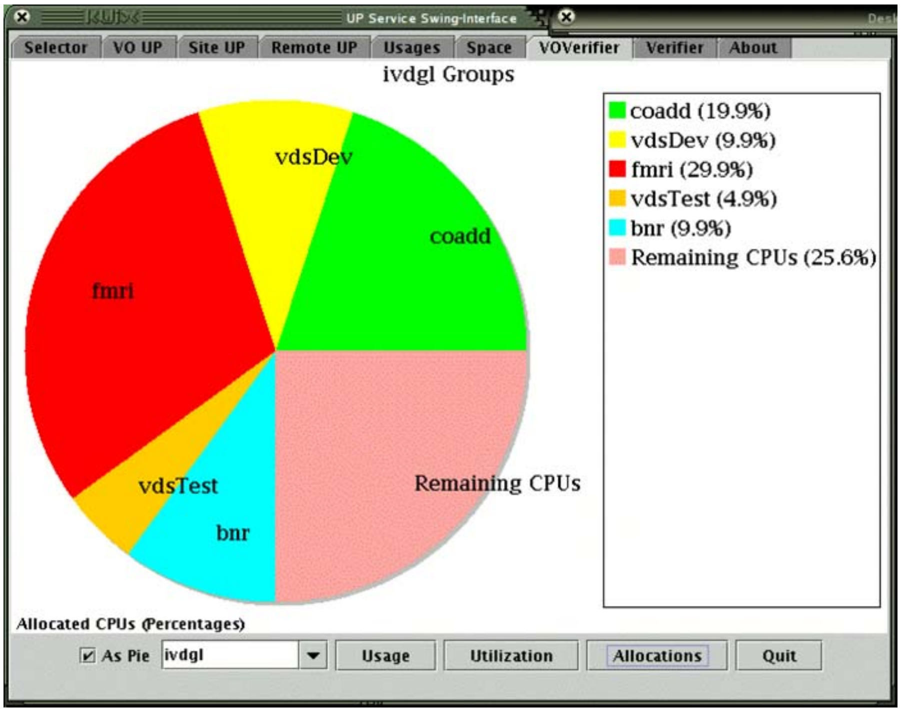


Fig. 9 Resource allocation scenario

In order to avoid gathering large amount of information, we also introduced various summation operations for different metrics. Practically from a human verifier point of view, this interface answers the question "Are uSLAs adequately enforced by each decision point?" and "What are the utilizations and allocations of different resource in the Grid?"

Also, the same graphical interface provides support for uSLA specification at group, VOs and site levels. The uSLAs can be entered and associated either with a site, a VO or a group. In another approach, various WS-Agreement like rules can be specified that are parsed when required to perform various job steering operations. Further, all uSLAs specified at a certain decision point are distributed to all other decision points if not marked as *private*. While this solution seems not very scalable (when going towards hundreds of decision points), we assume that for a Grid one hundred times larger than today Grid3, it is sufficient. As an additional note, uSLAs are associated with the decision point that distributed them and they can be deleted only by the same point of decision.

4 The Usage of GRUBER

GRUBER was tested with success in the OSG/Grid3 [2] environment for raw resource brokering for various workloads. Also, we tested its performance for higher-level (Grid) service brokering on an ad-hoc Grid testbed deployed on PlanetLab [70].

4.1 Testing Environment

The first testing environment, OSG/Grid3, uses Euryale [65] as a system to run jobs over Grid3/OSG [2]. Euryale uses Condor-G [30] (and Globus' WS-GRAM [38]) to submit and to monitor jobs at sites. It takes a late binding approach in assigning jobs to sites, meaning that site placement decisions are made immediately prior to running the job, rather than in an earlier planning phase. Euryale also implements a simple fault tolerance mechanism by means of job re-planning whenever a failure is discovered. We use the Euryale planner as our job submission tool while also interfacing with the GRUBER infrastructure. A tool called DagMan executes the Euryale prescript and postscript. The prescript calls out to the external GRUBER site-selector to identify the site on which the job should run, rewrites the job submit file to specify that site, transfers necessary input files to that site, registers transferred files with the replica mechanism, and deals with re-planning. The postscript file transfers output files to the collection area, registers produced files, checks on successful job execution, and updates file popularity.

DAS-2 environment [66], a wide-area distributed computer of 200 Dual Pentium-III computer nodes, represents a second example where GRUBER can be used with success. The environment is built out of clusters of workstations, which are interconnected by SurfNet, the Dutch university Internet backbone for wide-area communication, whereas Myrinet, a popular multi-Gigabit LAN, is used for local communication. The Grid scheduling technologies used in this environment are the Sun Grid Engine (SGE) [67] and KOALA [66]. KOALA has been designed and implemented by the PDS group in Delft in the context of the Virtual Lab for e-Science (VL-e) research project. The main feature of KOALA is its support for co-allocation, that is, the simultaneous allocation of resources in multiple clusters to single applications which consist of multiple components. Applications are executed using KOALA runners, which must be installed on all submission hosts and which handles the reservations of resources and the submission of applications. KOALA provides different kinds of runners and each runner is specialized for a different application type. These runners can be directly interfaced with GRUBER to take advantage of uSLAs for scheduling decisions. In this approach KOALA's runners invoke the GRUBER site selectors for a detailed list of available sites and their specific resources.

4.1.1 Usage Performance Metrics

We have used five metrics to evaluate the effectiveness of GRUBER raw-resource brokering performance over OSG/Grid3. We note that these metrics are independent, in the sense that a smaller total execution time does not imply a higher speedup.

- *Comp*: the percentage of jobs that complete successfully.

$$\text{Comp} = (\mathit{Completed\ Job}) / \#_{jobs} * 100.00$$

- *Util*: average resource utilization, the ratio of the per-job CPU resources consumed (ET_i) to the total CPU resources available as a percentage:

  $$\text{Util} = \sum\nolimits_{i=1..N} ET_i \big/ \left(\#_{cpus} * \Delta t\right) * 100.00$$

- *Delay*: average time per job (DT_i) that elapses from when the job arrives in a resource provider queue until it starts:

  $$\text{Delay} = \sum\nolimits_{i=1..N} DT_i \big/ \#_{jobs}$$

- *Time*: the total execution time for the workload.
- *Speedup:* the serial execution time to the Grid execution time for a workload. We note the definition implies a comparison between the running times on similar resources – practically, the first time in sequence and the second time in parallel.

For the service brokering tests, we used two additional metrics: # *of requests* and *response time* (for various operations as presented in Table 3).

4.1.2 Environment Characteristics

We performed all raw-resource brokering experiments on Grid3 (December 2004), which comprises around 30 sites across the U.S., of which we used 15. Each site is autonomous and managed by different local resource managers, such as Condor, PBS, and LSF. Each site enforced different uSLAs which were collected through GRUBER-SiteMonitor and used in scheduling workloads. We submit all jobs within the iVDGL VO, under a VO uSLA that allowed a maximum of 600 CPUs in parallel (fixed-limit uSLA at the VO level). Furthermore, we used also a fixed-limit uSLA at the VO group level where each individual workload corresponded to separate iVDGL group; the uSLA enforced was that any group can not get more than 25% of iVDGL CPUs, i.e., 150.

For the service brokering measurements, we deployed 10 ARESRAN managed WSRF services on PlanetLab. Each instance was providing Grid service for consumption while GRUBER was used for service brokering for the ad-hoc Grid. Each service ran inside a GT4 container on a PlanetLab node, except the GRUBER and WS-Index Service that ran inside the same container on a node at the University of Chicago. PlanetLab nodes were Linux-based PCs connected to the PlanetLab overlay network with worldwide distribution. All nodes are connected via 10 Mb/s network links (with 100 Mb/s on several nodes), have processor speeds exceeding 1.0 GHz IA32 PIII class processor, and at least 512 MB RAM. The clients were situated in the same network as the GRUBER node (UofC LAN), and ran on a node having an Intel Pentium 2.0 GHz processor, 1 GB of memory, 100 MBit/s network connectivity, and running the Linux-SuSe9.1 OS. We must also note that the PlanetLab configuration between the two experiment cases was different (and also a few months difference in time) – as an explanation for the difference in the overall performance.

4.1.3 Workloads and Settings

We used a single job type in all our raw-resource brokering experiments, the sequence analysis program BLAST. A single BLAST job has an execution time of about an hour (the exact duration depends on the CPU), reads about 10–33 KB of input, and generates about 0.7–1.5 MB of output: i.e., an insignificant amount of I/O. We used this BLAST job in two workload different configurations. In 1×1 K, we have a single workload of 1,000 independent BLAST jobs, with no inter-job dependencies. This workload is submitted once. Finally, in the 4×1 K case, the 1×1 K workload is run in parallel from four different hosts and under different VO groups. Also, each job can be re-planed at most four times through the submission infrastructure.

For service brokering tests, we opted for a simple service that is provided by default with the GT4 container, namely the *SecureCounterService*. The ARESRAN's PDP was enabled on all the containers providing services, while each instance of ARESRAN was also registering with a pre-defined WS-Index Service in order to support such a dynamic environment [64]. The GRUBER framework was aware only of the services registered with this WS-Index Service and performed brokering and reservations functionalities only for those. The uSLAs enforced at each site were *fixed-limit* uSLAs, with the following values (allowed number of requests over one hour): 10, 16, 27, 46, 77, 128, 214, 357, 595 and 992 for the #1 case and 0, 0, 0, 0, 0, 0, 0, 0, 0, 1000 for #2 case. Each experiment ran for 1 h.

Table 1 Results of four GRUBER scheduling policies for 1×500 workloads

	G-RA	G-RR	G-LU	G-LRU
Comp(%)	100	100	100	100
Util (%)	34.04	33.19	30.3	25.41
Delay (s)	9,202	6,700	6,169	9,125
Time (s)	28,116	24,225	21,362	20,434
Speedup	67.32	60.22	63.12	51.77

4.2 Low-level Resource Brokering Example on OSG/Grid3

In Table 1 are captured the results achieved by running a BLAST workload composed of 1×500 jobs [71] over Grid3. Here, in the ideal case, the values are: Comp = 100, Util = 25.00, Delay = 3,600, Time = 3,000, and Speedup = 150, while the metrics were defined by Dumitrescu et al. [71]. In this case GRUBER besides enforcing the required uSLAs also provides a 65 value for the *Speedup* metric in average. Taking in account that not all jobs in Grid3 were scheduled using GRUBER, we consider these values as encouraging from a user perspective that otherwise might have to wait three times longer when using a uSLA-unaware scheduling strategy (see Table 2).

We also compared GRUBER-based scheduling performance with an un-aware of any site uSLAs scheduling approach. In the first of the two methods, we use G-LU and G-Obs. The third alternative, S-RA, associates each job to a site selected at random. Table 2 shows the results obtained on Grid3 for the 1×1 K workload. We found out that a GRUBER site selector achieves a three times better performance than the one of S-RA site selector (resource utilization drops a few order of magnitude while the total execution time is almost three times higher).

Table 2 G-LU, G-Obs and S-RA strategies: performance for 1×1,000 workloads

	G-LU	G-Obs.	S-RA
Comp (%)	99.3	97.3	60.2
Util (%)	14.56	12.59	0.57
Delay (s)	50.50	62.01	121.0
Time (s)	33,300	40,320	80,280

4.3 Service Brokering Example on an Ad-hoc Grid Deployed on PlanetLab

Next, we present our experiments with the integrated GRUBER-ARESRAN infrastructure. Firstly, we consider the case when all service requests are done through GRUBER, secondly by means of a random scheduling strategy and thirdly by means of a round robin strategy. Case #1 shows the worst case performance of GRUBER as all provided services are 100% available; this case will favor simple scheduling approaches, such as the random or round robin assignment as the scheduling decision is not very important because of the abundance of available resources; furthermore, this case also shows the overhead incurred by the GRUBER assignment in relation to the trivial random or round robin assignments. Case #2 shows the best scenario for the GRUBER engine as a large portion of all provided services are not available; this case will favor a scheduling approach that can make good and informed decisions regarding the scheduling decisions in order to utilized the little available resources. Table 3 depicts the results for the client situated on the same network as the GRUBER engine for the three scenarios under the two different cases mentioned above.

For case #1, we can observe that the total number of request completed in one hour differs greatly between the third approaches (177 vs. 321 and, respectively, 312). These results show that the brokering request has taken an important ratio of the total execution time. However, in case #1, all provided services were initially 100% available, so the brokering decisions did not provide any real help in the beginning. Once the allocations started to fill up, the importance of GRUBER decisions have increased, but are not very apparent in the case #1 experiments.

For case #2, we see the potential benefits from the GRUBER assignments as we were able to obtain many more assignments than with either the random or the round robin assignment (150 vs. 59 and 52). We conclude that the utility of the GRUBER service brokering engine occurs only when the amount of services is large, custom advance reservations are performed in the system or the number of the requests is large and can potentially exhaust the available allocations.

Table 3 Service brokering performance results (Metrics: number of request, GRUBER infrastructure response time, tested service response time and tested service reject time)

Scheduling strategy	GRUBER assg (#1)	GRUBER assg (#2)	Random assg (#1)	Random assg (#2)	Round robin (#1)	Round robin (#2)
# of request	177	150	321	59	312	52
GRUBER resp	8.98	9.78	0	0	0	0
Service resp	11.45	16.55	10.84	17.45	11.04	17.06
Reject resp	0	0	8.89	10.18	9.28	9.85

5 The Performance of GRUBER

In this section we focus on the experimental results that capture some of the GRUBER capabilities in terms of both scalability and accuracy. Of course, these results are captured for the distributed version of GRUBER, because they make more sense in this scenario.

5.1 Testing Environment

For the GRUBER infrastructure performance measurements, we used a simulated Grid environment 10 times larger than the current OSG/Grid3 environment. The decision points were deployed on PlanetLab, providing brokering services while no real submission was actually performed.

5.1.1 Infrastructure Performance Metrics

We use three metrics to evaluate the effectiveness of DI-GRUBER: *Average Response Time* (Response), *Average Throughput* (Throughput), and *Average Scheduling Accuracy* (Accuracy).

We define *Response* as follows, with RT_i being the individual job time response and N being the number of jobs processed during the execution period:

$$\text{Response} = \sum\nolimits_{i=1..N} RT_i / N$$

Throughput is defined as the number of requests completed successfully by the service per unit time. Finally, we define the scheduling accuracy for a specific job (SA_i) as the ratio of free resources at the selected site to the total free resources over the entire Grid. *Accuracy* is then the aggregated value of all scheduling accuracies measured for each individual job:

$$\text{Accuracy} = \sum\nolimits_{i=1..N} (SA_i)/N$$

5.1.2 Environment Characteristics and Workloads

For the service brokering measurements of this section, we used similar settings as in the previous section for service brokering: 10 ARESRAN managed WSRF services on PlanetLab, while each instance GRUBER was used for service brokering for the ad-hoc Grid. Therefore, we skip the testbed description. Using DiPerF, we performed several tests where the decision points were exchanging status information under various settings: predefined brokering mesh connectivity (all, one half and, respectively, one quarter), predefined time intervals (1, 3, 10 and 30 min) and predefined number of decision points (1, 3 and 10). The workloads were generated by means of DiPerF, with jobs submission simulation from 120 submission hosts every 1 s.

5.2 Scalability Test Results and Comparison with a Peer-to-Peer Service

Figure 10 reports the experiments performed when using 1, 3 and 10 GRUBER decision points on PlanetLab. As can be easily observed, the results show improvement in terms of *Throughput* and *Response Time* when moving from 1 decision point to 10 decision points. The *Throughput* metric's value increases practically linearly with the number of decision points, reaching a constant value of 5 queries per seconds for 3 decision points, while going us up

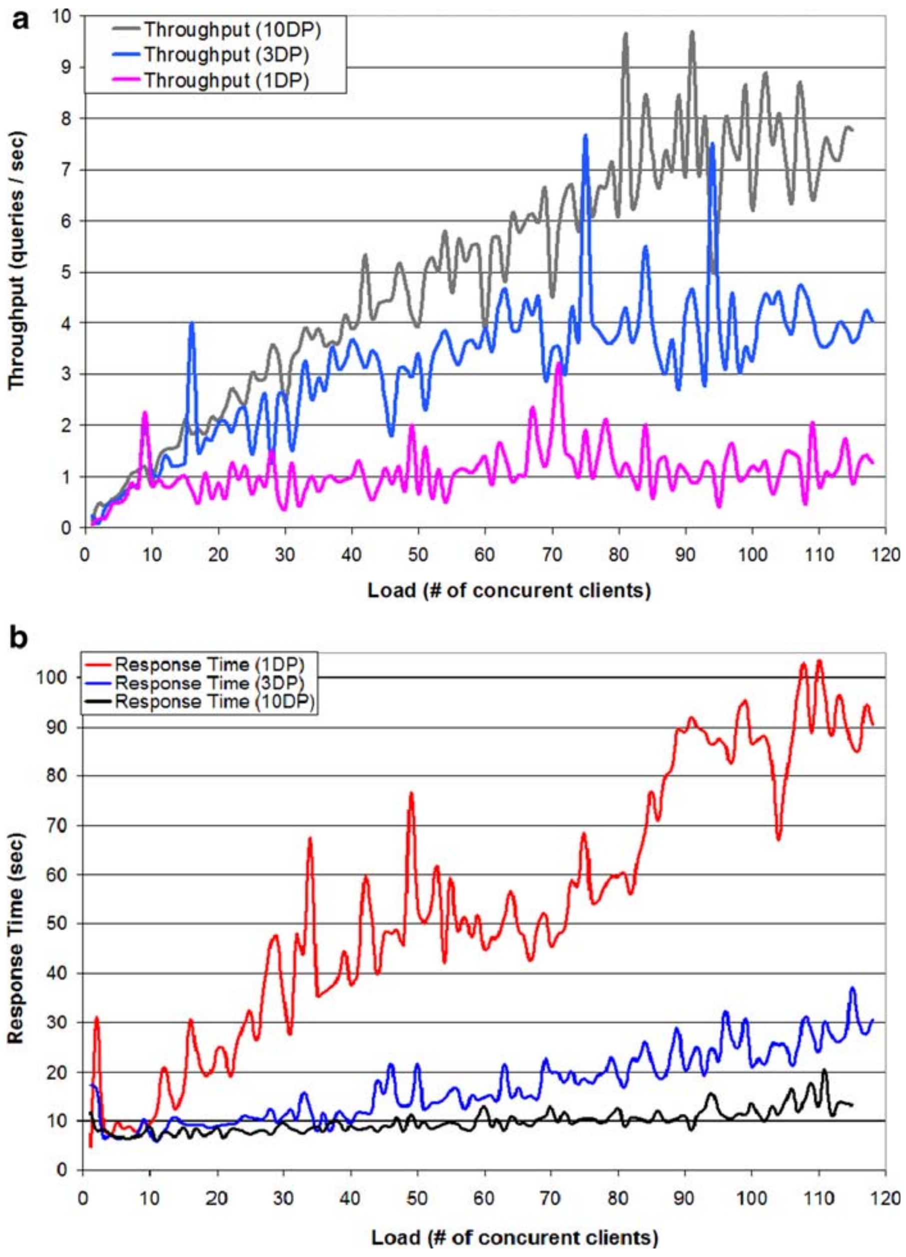


Fig. 10 DI-GRUBER throughput and response for 1, 3 and 10 decision points

as 16 queries per second for 10 decision points (*Throughput* is defined as the number of requests completed successfully by the service per time unit). The results are convincing in our view that moving to a distributed infrastructure provides real advantages from a performance point of view.

For convincing the reader that even though DI-GRUBER's transaction throughput seems low compared to 'other transaction processing systems,' we have performed further performance studies by means of DiPerF [72] on PlanetLab for a pretty well know distributed lookup service. The service chosen for testing was the PAST application, built on top of the PASTRY substrate [42].

The chosen setup was very similar to the one used for DI-GRUBER: the same PlanetLab nodes (around

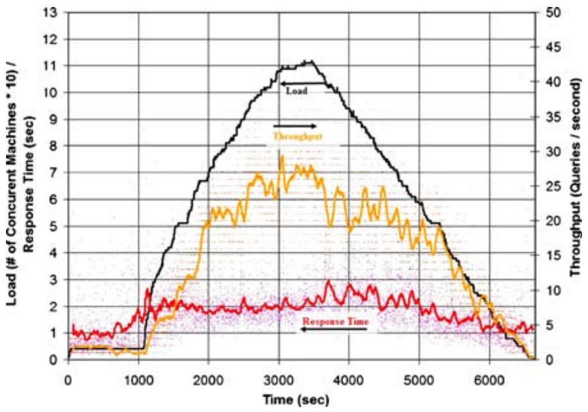


Fig. 11 PAST network response time (*left axis*) and throughput (*right axis*) for a variable load (*left axis*10*) on 120 PlanetLab nodes

120). This time we used five machines for running permanent PAST nodes, while the rest ones were brought up dynamically, joining and leaving the network in a controlled manner. Again, we used only one of the five nodes as the main contact point (a node situated at the University of Chicago). The rest ones were maintained as backup and to mimic the DI-GRUBER network. The length of the experiment was again one hour, while each joining node requested a lookup and an insert operation every second (or, if the previous operation took more than one second, at soon as the previous operation ended).

Our performance results are presented in Fig. 11. The measurements show that for insert and lookup operations, the PAST's response time is around 2.5 s with a higher variance in the beginning (the stabilization of the P2P network), while the throughput goes as up as 27 transaction per second in average. Also, the message lost rate for this ad-hoc network was pretty high compared with the one of DI-GRUBER. However, the network stabilization delay is higher for the P2P system (first 18% of the experimental time) compared with DI-GRUBER clients' instantaneous network join operation. Our last note is that all operations were performed and measured on the local nodes (insertion followed by lookup); each node was responsible to propagate the results further (thus the higher response time and lower throughput than in the case of employing the continuation).

Table 5 DI-GRUBER accuracy function of the exchange time interval for three decision points

Exchange interval (min)	Accuracy (%)
1	89
3	87
10	86
30	83

5.3 GRUBER Accuracy Performance Results

We consider three main directions of analysis: mesh connectivity, synchronization among decision points and the total number of decision points of the infrastructure. For each dimension, our results and considerations (where available) are captured in the following sections.

5.3.1 Accuracy with Mesh Connectivity

First, we measure *Accuracy* of the brokering infrastructure function of the decision points' average connectivity. We consider practically three cases: *full connectivity* (all DPs see each other), *half connectivity* (DP collects information only from half of all the others), and *one-fourth connectivity* (DP collects information only from a quarter of all the others). The results were achieved by means of the DI-GRUBER infrastructure in all three above configurations and are captured in Table 4.

We can observe that the performance of the brokering infrastructure drops substantially with connectivity degree of each individual decision point; in a

Table 4 DI-GRUBE accuracy function of the infrastructure mesh connectivity

Connectivity (N=10)	Accuracy (%)
N-1	75
N/2	62
N/4	55

Table 6 DI-GRUBER accuracy function of the number of decision points

Number of decision points	Accuracy (%)
1	98
3	89
10	75

nutshell, *Accuracy* drops almost linearly with clients' connectivity degree.

5.3.2 Accuracy with Time Exchange Intervals

The results in Table 5 show that, for a three decision point infrastructure, a three to ten minutes exchange interval is sufficient for achieving almost 90% *Accuracy*. However, this accuracy value depends also on the number of the jobs scheduled by the decision points.

5.3.3 Accuracy with the Number of Decision Points

Next, we analyze the performance of DI-GRUBER and its strategies for providing accurate scheduling decisions function of the number of decision points in the infrastructure. Table 6 depicts the accuracy performance. We note that the accuracy drops to 75% in the 10 decision points case.

6 Summary and Conclusions

We have presented a Grid resource broker, GRUBER, for representing and managing resource allocation policies in a multi-site, multi-VO environment. GRUBER is an infrastructure for uSLA specification and enforcement in large and dynamic distributed environments. It is our result of a uSLA-based Grid management infrastructure as described by Dumitrescu et al. [1, 21]. We note that GRUBER is a complex service: a query to a decision point may include multiple message exchanges between the submitting client and the decision point, and multiple message exchanges between the decision points and the job manager in the Grid environment. In a WAN environment with message latencies in the 100 s of milliseconds, a single query can easily take multiple of seconds to serve. We expect that performance will be significantly better in LAN environments. However, one of GRUBER's design goals was to offer resource brokering in a WAN environment such as multi-site, multi-VO Grids.

Managing uSLAs within large virtual organizations that integrate participants and resources spanning multiple physical institutions is a challenging problem. Maintaining a single unified decision point for uSLA management is a problem that arises when many users and sites need to be managed. We provide a solution, namely the GRUBER infrastructure with the distributed variation, to address the question on how uSLAs can be stored, retrieved and disseminated efficiently in a large distributed environment. We believe that all these features presented in this paper make GRUBER capable working not only in large Grid environments, but also in dynamic and heavily-loaded environments where automatic recovery can be a problem. The novelty of this paper consists in introducing and evaluating a model and architecture for generic Grid resource and service brokering, uSLA-based provisioning and advance reservation in large distributed and dynamic environments.

Acknowledgments Authors would like to thank Anne Rogers for her feedback on the material in this paper. Also, we thank Jens-S Vöckler and Michael Wilde for their support and help in integrating Euryale and GRUBER. Some of the experimental results reported in this paper were supported by the NSF Information Technology Research GriPhyN project, under contract ITR-0086044.

References

1. Dumitrescu, C., Wilde, M., Foster, I.: A model for usage policy-based resource allocation in Grids. In: Proceedings of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005), Stockholm, Sweden, pp. 191–200 (2005) (ISSN: 0-7695-2265-3)
2. Foster, I., et al.: The Grid 2003 production Grid: principles and practice. In: Proceedings of the 13th International Symposium on High Performance Distributed Computing (HPDC), pp. 236–245 (2004) (ISSN: 1082–8907)
3. Foster, I.: Grid computing. In: Proceedings of the Advanced Computing and Analysis Techniques in Physics Research (ACAT). AIP Conference Proceedings, Chicago, IL, vol. 583, pp. 51–56 (2000)
4. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: enabling scalable virtual organizations. Int. J. Supercomput. Appl. **2150**, 200–222 (2001) (ISBN: 3-540-42495-4)
5. Dan, A., Davis, D., Kearney, R., Keller, A., King, R., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M., Youssef, A.: Web services on demand: WSLA-driven automated management. IBM Syst. J. **43**, 136 (2004)
6. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: a protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing, Edinburgh, Scotland (2002)

7. Czajkowski, K., Dan, A., Rofrano, J., Tuecke, S., Xu, M.: WS-agreement: agreement-based Grid service management (OGSI-Agreement), Version 0. [Online: http://forge.gridforum.org/projects/graap-wg/document/Draft_OGSI-agreement_Specification/en/1/Draft_OGSI-Agreement_Specification.doc]
8. Gimpel, H., Ludwig, H., Dan, A., Kearney, R.: PANDA: specifying policies for automated negotiations of service contracts. In: Proceedings of the 1st International Conference on Service Oriented Computing, pp. 287–302. Trento, Italy (2003)
9. Verma, D.C.: Policy Based Networking, Architecture and Algorithm. New Riders, Indianapolis, IN (2000 November)
10. Dumitrescu, C.: INTCTD: a peer-to-peer approach for intrusion detection. In: Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'06), Singapore (2006) (ISBN: 0-7695-2585-7)
11. Verma, D.C.: Simplifying Network Administration using Policy based Management. IBM, UK (2004)
12. Lamanna, D., Skene, J., Emmerich, W.: SLang: a language for defining service level agreements. In: Proceedings of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems, Puerto Rico, pp. 100–106. IEEE-CS Press (2003 May)
13. LHC Computing Project (2004)
14. Ranganathan, K., Foster, I.: Decoupling computation and data scheduling in distributed data-intensive applications. In: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, pp. 352. Edinburgh, Scotland (2002) (ISSN: 1082-8907)
15. Mambelli M.: Capone and VDS: The University of Chicago and Argonne National Laboratory: Chicago [Online: http://griddev.uchicago.edu/swhome/atgce/] (2005)
16. Annis, J., Kent, S., Szalay, A.: The SDSS-GriPhyN Challenge Problems: Cluster Finding, Correlation Functions and Weak Lensing. FermiLab, Batavia, IL (2001)
17. Maltsev, N., Sulakhe, D., D'Souza, M.J., Glass, E., Rodriguez, A., Syed, M., Zhang, Y.: GNARE: Genome Analysis Research Environment. 2005, Argonne National Laboratory/Chicago [Online: http://compbio.mcs.anl.gov/gnare/gnare_home.cgi] (2006)
18. Foster, I., Voeckler, J., Wilde, M., Zhao, Y.: Chimera: a virtual data system for representing, querying, and automating data derivation. In: Proceedings of the Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop. IEEE Computer Society, Washington, DC (1995 May)
19. Iosup, A., Dumitrescu, C., Epema, D., Liu, H., Wolters, L.: An analysis of four long-term Grid traces. Technical University of Delft: Delft, Netherlands [Online: http://pds.twi.tudelft.nl/reports/2006/PDS-2006-003/PDS-2006-003.pdf] (2006)
20. Open Science Grid (OSG) [Online: http://www.opensciencegrid.org/] (2004)
21. Dan, A., Dumitrescu, C., Ripeanu, M.: Connecting client objectives with resource capabilities: an essential component for Grid service management infrastructures. In: Proceedings of the 2nd ACM International Conference on Service Oriented Computing (ICSOC'04), pp. 57–64. New York, NY (2004) (ISSN 1-58113-871-7)
22. Legrand, I., Newman, H., Galvez, P., Voicu, E., Cirstoiu, C.: MonALISA: a distributed monitoring service architecture in computing. In: Proceedings of the High Energy Physics (HEP), La Jolla, CA (2003)
23. DOE Science Grid PKI Certificate Policy and Certification Practice Statement (2002)
24. Dumitrescu, C.: ARESRAN: A WSRF-based resource reservation service for Grid service. [Online: http://peopellcs.uchicago.edu/~cldumitr/ARESRAN] (2005)
25. Pearlman, L., Welch, V., Foster, I., Kesselman, C., Tuecke, S.: A community authorization service for group collaboration. In: Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, pp 55–59. Monterey, CA (2002) (ISBN: 0-7695-1611-4)
26. The Globus Project Team: CAS – community authorization service. [Online: http://www.nsf-middleware.org/Lists/Products/DispForm.aspx?ID=47] (2006)
27. Zhao, T., Karamcheti, V.: Expressing and enforcing distributed resource agreements. In: Proceedings of High Performance Networking and Computing Conference (SC'2000), pp. 62. Dallas, Texas (2000) (ISSN 0-7803-9802-5)
28. Raman, R.: Matchmaking Frameworks for Distributed Resource Management. PhD Thesis, University of Wisconsin (2000)
29. Foster, I., Roy, A., Sander, V., Winkler, L.: End-to-End Quality of Service for High-end Applications. Computer Communications 27(14). Kluwer, Norwell, MA (2004) (ISBN: 1375–1388)
30. Thain, D., Tannenbaum, T., Livny, M.: Condor and the Grid. In: Berman, F., Hey, A.J.G., Fox , G. (eds.) Grid Computing: Making The Global Infrastructure a Reality. Wiley, New York, NY (2003) (ISBN: 0-470-85319-0)
31. Ludwig, H., Dan, A., Kearney, B.: Cremona: an architecture and library for creation and monitoring WS-Agreements. In: Proceedings of the ACM International Conference on Service Oriented Computing (ICSOC'04), New York, NY (2004)
32. In, J., Avery, P., Cavanaugh, R., Ranka, S.: Policy based scheduling for simple quality of service in Grid computing. In: Proceedings of the International Parallel & Distributed Processing Symposium (IPDPS), p. 23. Santa Fe, New Mexico (2004) (ISBN: 0-7695-2132-0)
33. Buyya, R.: GridBus: A Economy-based Grid Resource Broker. The University of Melbourne, Melbourne, Australia (2004)
34. Foster, I.: The Grid: a new infrastructure for 21st century science. Phys. Today **55**(2), 42–47 (2002)
35. Mueller, E.T., Moore, J.D., Popek, G.J.: A nested transaction mechanism for LOCUS. In: Proceedings of the 9th ACM Symposium on Operating System Principles (SOSP), Bretton Woods, New Hampshire (1983)
36. Stonebraker, M., et al.: Mariposa: a wide-area distributed database system. VLDB J. **5**(1), 48–63 (1996)
37. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: Grid Services for Distributed Systems Integration. IEEE Computer **35**(6), 37–46 (2002)
38. Humphrey, M., Wasson, G., Jackson, K., Boverhof, J., Rodriguez, M., Bester, J., Gawor, J., Lang, S., Foster, I., Meder, S., Pickles, S., McKeown, M.: State and events for

web services: a comparison of five WS-resource framework and WS-notification implementations. In: Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), Research Triangle Park, NC, 24–27 July 2005
39. Czajkowski, K., et al.: Grid information services for distributed resource sharing. In: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing. San Francisco, IEEE Computer Society, Los Alamitos, CA (2001)
40. Thompson, M.R., Essiari, A., Mudumbai, S.: Certificate-based authorization policy in a PKI environment. ACM Trans. Inf. Syst. Secur. **6**(4), 566–588 (2003)
41. Lupu, E.: A role-based framework for distributed systems management, in Department of Computing. PhD thesis, University of London, London (1998)
42. Rowstron, A., Druschel, P.: Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. Lect. Notes Comput. Sci. **2218**, 329–350 (2001)
43. Dumitrescu, C., Foster, I.: GRUBER: a Grid resource SLA broker. In: Proceedings of the 11th International Euro-Par Conference, pp. 465. Portugal (2005) (ISBN: 3-540-28700-0)
44. The University of Wisconsin: UWMadisonCMS Open Science Grid Site Policy Page. University of Wisconsin, Madison, WI (2006)
45. FNAL: FNAL: GPFARM Site Policy for OSG. FNAL (2006)
46. USCMS: USCMS: OSG Policy Pages. USCMS (2006)
47. Keahey, K., Araki, T., Lane, P.: Agreement-based interactions for experimental science. In: Proceedings of the 10th International Euro-Par Conference, p. 399. Italy (2004) (ISBN: 3-540-22924-8)
48. Kay, J., Lauder, P.: A Fair Share Scheduler. University of Sydney, AT&T Bell Labs (1998)
49. Epema, D.H.J., Livny, M., van Dantzig, R., Evers, X., Pruyne, J.: A worldwide flock of condors: load sharing among workstation clusters. Future Gener. Comput. Systs. **12**, 53–65 (1996) (ISSN: 0167-739X)
50. Wolf, L.C., Steinmetz, R.: Concepts for reservation in Advance. Multimed. Tools Appl. **4**(3), 255–278 (1997) (ISSN 1380–7501) (Kluwer)
51. Maui Team: Maui Scheduler. Center for HPC Cluster Resource Management and Scheduling. [Online: http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php]
52. Keller, A., Ludwig, H.: The WSLA framework: specifying and monitoring service level agreements for web services. J. Netw. Syst. Manag. **11**(1), 57–81 (2003)(Plenum)
53. Litzkow, M.J., Livny, M., Mutka, M.W.: Condor – a hunter of idle workstations. In: Proceedings of the 8th International Conference on Distributed Computing Systems, pp. 104–111. San Jose, CA (1998) (ISBN: 0-8186-0865-X)
54. Altair Grid Technologies. OpenPBS (Portable Batch System) (2004) [Online: http://www.openpbs.org/]
55. Foster, I., Kesselman, C.: Globus: a toolkit-based Grid architecture. In: The Grid: Blueprint for a Future Computing Infrastructure, pp. 259–278. Morgan Kaufmann, San Mateo, CA (1998)
56. Dumitrescu, C., Foster, I.: Usage policy-based CPU sharing in virtual organizations. In: Proceedings of the 5th International Workshop on Grid Computing, pp.53–60. Pittsburgh, PA (2004) (ISSN: 1550-5510)
57. Wolf, L.C., Steinmetz, R.: Concepts for resource reservation in advance. Multimed. Tools Appl. **4**(3): 255–278
58. Tannenbaum, T., Wright, D., Miller, K., Livny, M.: Condor – a distributed job scheduler. In: Berman, F., Hey, A.J.G., Fox, G. (eds.) Grid Computing: Making the Global Infrastructure a Reality. Wiley, New York, NY (2003) (ISBN: 0-470-85319-0)
59. Henderson, R., Tweten, D.: Portable batch system: external reference specification. Technical report, NASA, Ames Research Center (1996)
60. Platform, User's Guide: (2006)[Online: http://www.platform.com/Products/Platform.LSF.Family/]
61. Dumitrescu, C.: Policy Research for iVDGL. 2004, The University of Chicago/GriPhyN Project NSF Review 2004. Chicago, USA (2004) [Online: http://poeple.cs.uchicago.edu/~cldumitr/]
62. Dumitrescu, C., Wilde, M., Foster, I.: Usage policies at the site level in Grid. iVDGL/GriPhyN Project: The University of Chicago (2006) [Online: http://poeple.cs.uchicago.edu/~cldumitr/]
63. Constandache, I.: Policy based dynamic negotiation for Grid services authorization. In: L3S Research Center. University of Hannover, Hannover, Germany (2005)
64. Dumitrescu, C., Raicu, I., Foster, I.: DI-GRUBER: a distributed approach for Grid resource brokering. In: Proceedings of the Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC'2005), p. 38. Seattle, WA (2005) (ISBN: 1-59593-061-2)
65. Vöckler, J.-S., Wilde, M., Foster, I.: The GriPhyN Virtual Data System. GriPhyN Technical Report, The University of Chicago (2002) [Online: http://www.griphyn.org/]
66. Mohamed, H.H., Epema, D.H.J.: Experiences with the KOALA co-allocating scheduler in multiclusters. In: Proceedings of the 5th IEEE/ACM Int'l Symp. on Cluster Computing and the GRID (CCGrid2005), Cardiff, UK (2005 May)
67. SUN: Sun Grid Engine. (2004) [Online: http://www.sun.com]
68. LCG: LHC – The Large Hadron Collider Project [Online: http://lcg.web.cern.ch/LCG/] (2006)
69. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The Data Grid: towards an architecture for the distributed management and analysis of large scientific data sets. J. Netw. Comput. Appl. **23**, 187–200 (2001) [Online: http://www.globus.org/]
70. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: PlanetLab: an overlay testbed for broad-coverage services. ACM SIGCOMM Comput. Commun. Rev. **33**(3), 3–12 (2003) (ISSN: 0146-4833)
71. Dumitrescu, C., Raicu, I., Foster, I.: Experiences in running workloads over Grid3. In: Proceedings of the Grid and Cooperative Computing (GCC2005), pp. 274–286. Beijing, China (2005) (ISBN: 3-540-30510-6)
72. Dumitrescu, C., Raicu, I., Ripeanu, M., Foster, I.: DiPerF: automated distributed performance testing framework. In: Proceedings of the 5th IEEE/ACM International Workshop in Grid Computing (Grid'04), pp. 289–296. IEEE Computer Society, Los Alamitos, CA (2004) (ISBN: 0-7695-2256-4)