Extending a Distributed Usage SLA Resource Broker to Support Dynamic Grid Environments

Catalin L. Dumitrescu[*], Ioan Raicu[*], Ian Foster[*+]

*Computer Science Department
The University of Chicago
catalind@cs.uchicago.edu

+Mathematics and Computer Science Division
Argonne National Laboratory
foster@mcs.anl.gov

Abstract

DI-GRUBER is a distributed Grid brokering service, with multiple decision points. Previously, the membership relationship among the decision points was statically defined. This limited the deployment in a dynamic environment where VOs appear and vanish frequently. Here we report on the DI-GRUBER enhancements with support for WS-MDS Index Service that allow the scheduling infrastructure to operate in VO-centric more dynamic environments. The underlying mechanisms provide each decision point the necessary information regarding the location of other decision points.

One interesting difference in this approach is that each decision point can have only a partial view of the brokering infrastructure, and hence the brokers' performance suffers to some degree. We also measure the trade off between the degree of connectivity in the mesh network, and the performance of the brokering infrastructure, and compare with previous results on the correlation between scheduling accuracy and the amount of partial knowledge.

1. Introduction

The motivating scenarios for our work are large grid environments in which virtual organizations (VOs) appear and vanish in a dynamic manner. Such VOs might be companies requiring outsourcing services, or scientific laboratories that want to participate temporarily in different collaborations with access to other resources.

Thus, we distinguish between two types of entities participating in these scenarios: *providers* and *consumers*. They may be nested: a *provider* may function as a middleman, providing access to resources to which the provider has itself been granted access by some other *provider*. While sharing policies issues can arise at multiple levels in such scenarios, the dynamicity of such an environment is also a problem. *Providers* want to express (and enforce) various usage service level agreements (uSLAs) under which resources are made available to *consumers*. *Consumers* want to access and interpret uSLA statements published by *providers*, in order to monitor their agreements and guide their activities. Both *providers* and *consumers* want to verify that uSLAs are applied correctly.

We extend here our work about constructing a scalable and dynamic resource management service that supports uSLA expression, publication, discovery, interpretation, enforcement, and verification in grid environments. This problem encompasses challenging and interrelated scheduling, information synchronization, and scalability issues. We build on much previous work concerning the specification and enforcement of local resource scheduling policies [1, 2] the GRUBER broker [3], and the DI-GRUBER variation [4]. GRUBER addresses issues regarding how uSLAs can be stored, retrieved, and disseminated efficiently in a distributed environment, specifically grids. DI-GRUBER addresses also issues such as managing large grid environments and state maintenance among its decision points, which are in fact GRUBER instances that inter-communicate. DI-GRUBER extends GRUBER by introducing support for multiple scheduling decision points, and loosely synchronizations via periodic information exchange.

In this paper we present three major enhancements to the DI-GRUBER two layer brokering infrastructure. The improvements are: WS-MDS Index-based infrastructure discovery, support for uSLA automated reconciliation and decision point overload signaling. We believe that these improvements make DI-GRUBER capable working not only in large grid environments, but also in dynamic and heavily-loaded environments where automatic recovery becomes also a problem. Here, we prove our belief correct by means of measuring both the capability and performance of the extended DI-GRUBER. We are also interested in gaining insights about uSLA reconciliation and dynamic management strategies for future work.

The rest of this article is organized as follows. We first provide a more detailed description of the problem that we address. Next, we introduce some background information about the environment where DI-GRUBER is supposed to work. In section 3 we detail the enhancements performed on DI-GRUBER framework, while in section 4 we measure the performance of the new infrastructure and also compare with the previous results we achieved by using DI-GRUBER. The paper ends with related work and our conclusions about the results and lessons we learnt.

1.1. Problem Statement

This work targets grids that may comprise hundreds of institutions and thousands of individual investigators and various institutions with institutions and VOs arising and vanish often [5]. More, each individual investigator and institution may participate in, and contribute resources to, multiple collaborative projects that can vary widely in scale, lifetime, and formality. DI-GRUBER focuses on providing a brokering infrastructure for such an environment, providing also scalable and self-organizing services for such communities. Thus, we examine techniques for providing reliable support for resource brokering by means of DI-GRUBER. For example, an important problem mentioned before is how to determine dynamically the number of decision points required for such large grid scenarios [4].

1.2. Dynamic Decision Points Bootstrap Considerations

DI-GRUBER is a distributed Grid brokering service, with multiple decision points. Previously, the membership relation among the decision points was statically defined by means of local configuration files. Such proved to be a limitation for deployment in dynamic environments where various entities (sites, VOs, or groups) may appear and vanish frequently. Our approach in solving this problem is the introduction of WS-MDS Index registration support that allows individual decision points and clients discover each other automatically without any human intervention. This underlying mechanism provides each DI-GRUBER decision point and client the necessary information regarding the existence of all the other decision points, as well as a generic view of the infrastructure and its instantaneous status.

Thus WS-Index Service becomes the central point for joining or leaving the brokering network. One problem that we do consider is that the WS-Index Service becomes the bottle neck of the infrastructure; however our previous experiments proved that cannot be the case [6]. Figure 1 present the results of a performance measuring experiment performed on WS-Index Service in PlanetLab {Chun B., 3, July 2003 #3617} with 288 machines all over the world. This test was very interesting due to the fact that the throughput achieved while all 288 machines were concurrently accessing the WS-MDS Index was around 200 queries per second on average. Although the WS-MDS Index managed to service all the 288 clients concurrently, its efficiency in terms of sustaining a high throughput clearly dropped over 200 machines in a wide-are network [7].

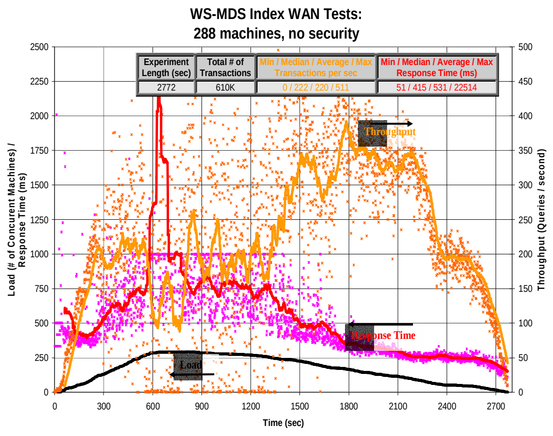


Figure 1: WS-MDS Index WAN Tests with no security (288 clients running on 288 physical nodes in PlanetLab in a WAN connected via 10 Mb/s links); tunable parameters: utilized 288 concurrent clients, with each client starting every 2 seconds; left axis – load, response time; right axis – throughput (GT4)

1.3. uSLA Management Issues

Regarding uSLA management, one problem explored in this paper is the how the uSLAs should be exchanged in order to maintain a coherent view managed environment at each DI-GRUBER decision point. Several operations have to be considered, such us uSLA propagation, reconciliation and removal. These operations may occur whenever new DI-GRUBER decision points join or leave the brokering network, brought up either by a new VO or resource provider.

Another problem faced in practice is the necessity for privacy when sensitive computing resources are shared. In certain cases, some consumers (either users or VOs) can require various levels of privacies about their resources or work to be executed (job types and priorities, data movement and characteristics). Thus, the maintenance of a *private* DI-GRUBER decision point could be a necessity in such situations. This issue can be encountered from the VO level on down to individual users. The problem becomes even more sensible when dealing with commercial entities. The enhanced DI-GRUBER addresses this problem by providing the option for each decision point to either publish its local information or not to other decision points in the network. As a future refinement, such a private decision point can be enhanced to publish its local database only to a subset of peers that meet certain requirements.

2. Background Information

We now introduce the main concepts and tools used in this paper that are necessary for the experiments in this paper or required in a real deployment by DI-GRUBER in order to perform its functionalities.

2.1. DI-GRUBER Decision Point (GRUBER)

GRUBER [3] is a prototype Grid V-PEP and S-PEP infrastructure that implements the brokering functionalities required for steering workloads in a distributed environment based on uSLAs. GRUBER was the main component used before [3] for job scheduling over a real grid, namely the Grid3 environment [8]. It is able to perform job scheduling based on notions such as VO, group VO, and USLAs at various levels. The main four principal components are described next and illustrated in Figure 2.

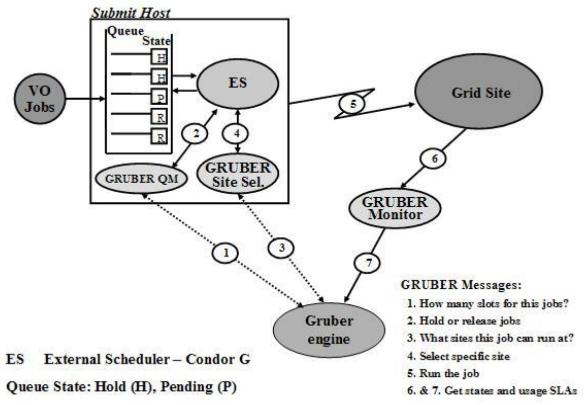


Figure 2: GRUBER Architecture

The *GRUBER engine* is the main component of the architecture. It implements various algorithms for detecting available resources and maintains a generic view of resource utilization in the grid.

The *GRUBER site monitor* is a data provider for the GRUBER engine. This component is optional and can be replaced with various other grid monitoring components that provide similar information, such as MonaLisa or Grid Catalog.

A *GRUBER client* represents a standard GT client that allows communication with other GRUBER components and the GRUBER engine, such as the GRUBER site selectors that we introduce next.

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 least recently used task assignment policies.

Finally, the *GRUBER queue manager* is a GRUBER client that resides on a submitting host. This component monitors VO policies and decides how many jobs to start and when. It interacts with the GRUBER engine to obtain site selection recommendations.

Currently, GRUBER is implemented as both an OGSI service and a WS (WS) service based on the Globus Toolkit (GT3 and respectively GT4). In the experiments performed for this paper, we have used the WS version of GRUBER engine and the site selectors, but not the queue manager. In this configuration, GRUBER might seem to operate as a site recommender because it does not enforce VO-level uSLAs. However, we assume that all clients comply with the recommendations and that there is no need for enforcement.

GRUBER does not itself perform job submission, but as shown in Figure 2 can be used in conjunction with various grid job submission infrastructures. Previously, we have interfaced GRUBER for real job executions with the Euryale planner [19] largely used on Grid3. We also believe that GRUBER would work with other similar grid planner, such Pegasus [8].

2.2. DI-GRUBER

Managing uSLAs within environments that integrate participants and resources spanning many physical institutions can become a challenging problem. A single unified uSLA management decision point providing brokering decisions over hundreds to thousands of jobs and sites can easily become a bottleneck in terms of reliability as well as performance. DI-GRUBER, an extension to the GRUBER prototype, was developed as a distributed grid uSLA-based resource broker that allows multiple decision points to coexist and cooperate in real-time.

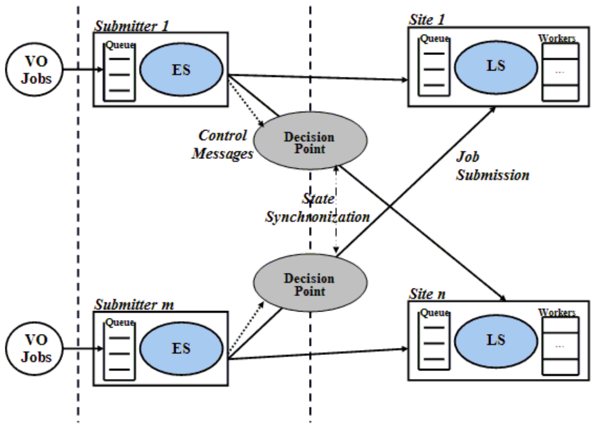


Figure 3: DI-GRUBER Architecture

DI-GRUBER targets to provide a scalable management service with the same functionalities as GRUBER but in a distributed approach [9, 4]. It is a two layer resource brokering service (Figure 3), capable of working over large grids, extending GRUBER with support for multiple scheduling decision points that cooperate by periodically exchanging various status information [4]. While this system has proved some strong improvements over the centralized approach provided by GRUBER, it still lacks a few important features that were later implemented and analyzed in this paper.

2.3. DI-GRUBER uSLA Semantics

DI-GRUBER understands both a *consumer* and a *provider* as an entity that has certain characteristics and requirements. These consumers and providers are users and groups, and VOs and sites respectively, allowing either simple sharing rules similar to MAUI specifications or complex sharing rules as defined in the WS-Agreement. In the second approach allocations are expressed as WS-Agreement goals and requirements introducing a finer granularity for the rules' specification. We based DI-GRUBER uSLA specification on a subset of WS-Agreement, taking advantage of the refined specification and the high-level structure. [10, 11]

2.4. Information Dissemination Strategies

An important issue for a decentralized brokering service is how uSLAs and utilization information are disseminated among decision points. We need to aggregate correctly partial information gathered at several points; without a correct aggregation of the partial information, wrong decisions can result in workload starvation and resource under-utilization.

This problem can be addressed in several ways. In a first approach, both resource usage information and uSLAs are exchanged among decision points. In a second approach, only utilization information is exchanged. As a possible variation on these two approaches, whenever new sites are detected, their status is incorporated locally by each decision point, which means each decision point has only a partial view of the environment. In a third approach, no usage information is exchanged and each decision point relies only on its own mechanisms for detecting grid status.

While the second approach was experimented with success before [4], we focus here on the first approach for information synchronization among the decision points. This analysis introduces additional complexities required for uSLA tracking and management correctly at each decision point.

2.5. Open Science Grid

We envisage that DI-GRUBER can be used in real grid environments that are ten to hundreds times bigger than today Open Science Grid (OSG: previously known as Grid3 [8]). OSG is a multi-virtual organization environment that sustains production level services required by various physics experiments. The infrastructure comprises more than 50 sites and 4500 CPUs, over 1300 simultaneous jobs and more than 2 TB/day aggregate data traffic. The participating sites are the main resource providers under various conditions.

Thus, we consider for the experiments in this paper an environment similar to OSG but ten times larger and with much higher rates of job scheduling. DI-GRUBER provides the required uSLA-based solution for job scheduling decisions for environments similar to OSG, by providing a means for informed site selection at the job level and beyond. In a simpler case, it can act also as a monitoring infrastructure that offers more information than only current resource utilizations.

2.6. PlanetLab Testbed

PlanetLab [13] is a geographically distributed platform for deploying, evaluating, and accessing planetary-scale network services. PlanetLab is a shared community effort by a large international group of researchers, each of whom gets access to one or more isolated "slices" of PlanetLab's global resources via a concept called distributed virtualization. PlanetLab now comprised over 500 nodes (Linux-based PCs or servers connected to the PlanetLab overlay network) distributed worldwide. Almost all nodes are connected via 10 Mb/s network links (with 100Mb/s on several nodes), have processor speeds exceeding 1.0 GHz IA32 PIII class processor, and at least 512 MB RAM.

3. DI-GRUBER Enhancements

Maintaining a local and static view of all the decision points in brokering architecture might be a challenging problem and in most cases a cumbersome one. DI-GRUBER was developed as a distributed uSLA-based grid resource broker that allows multiple decision points to coexist and cooperate in real-time. The problem is that without a supporting mechanism for dynamic discovery of the brokering infrastructure, some of the advantages offered by this infrastructure may become impractical. The main problem arises from maintaining the list of decision points at each location in the infrastructure (both decision point and client locations).

Next, we explore the capabilities and enhancements introduced to the WS-MDS Index based DI-GRUBER infrastructure.

3.1. Control Console

Firstly, accurate monitoring is important if we are to understand how the framework actually performs in different situations (the *verifiers* concepts introduced in [9]). 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 DI-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 easy to visualize mode (Figure 4).

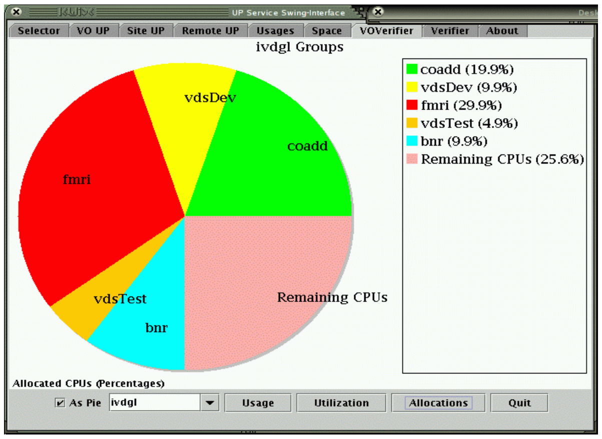


Figure 4: Resource Allocation Example

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?"*.

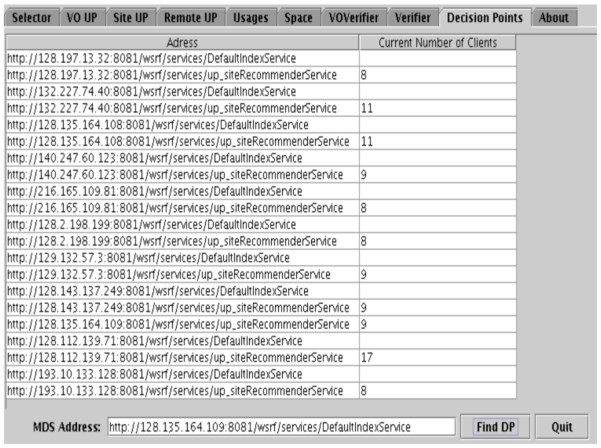
Also, the same graphical interface provides support for uSLA specification at group, VOs and site levels. The uSLAs can be entered and associated either with a site, a VO or a group. In another approach, various WS-Agreement like rules can be specified that are parsed when required to perform various job steering operations. Even though this element is important for managing a grid infrastructure for job allocations, we consider such an example beyond the scope of this paper, as being already presented elsewhere [3].

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 is sufficient (as also presented in paragraph 4.2). 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.

3.2. Decision Point Bootstrap Implementation

As already described, the ability to bring up a decision point is important in a large and dynamic grid. While this problem was not addressed before [4], we address also this problem here. Our solution uses the functionalities offered by the WS-MDS Index Service for service registering and querying.

In our implementation, each DI-GRUBER decision point registers with a predefined WS-MDS 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. When we use the term "most appropriate", we refer to metrics such as load and number of clients already connected. In Figure 5 is presented such a view (achieved by means of the same 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 responding to a client, this client automatically queries the registry and selects a different point of contact.



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

Figure 5: Decision Points View

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 WS-MDS Index Service. While we do not have implemented this facility yet [4], a human operator can easily perform such an operation (starting a new GT4 container where a DI-GRUBER decision point was already developed).

Additionally, if a pool of decision points are maintained in background and forced to register with the WS-MDS Index Service only when needed, the operation is 100% automated.

3.3. uSLAs Synchronization Approach

The next problem we focus on is the uSLA synchronization and reconciliation among the DI-GRUBER decision points. There are two main cases that we consider: uSLA decision point sets are disjunctive and uSLA decision points sets are not disjunctive.

In the first case, each decision point acquires the rest of the uSLAs during synchronization operations. These uSLAs are stored locally and used whenever a job decision is required. The advantage and simplicity of this solution consists in the fact that no reconciliation is necessary. However, this solution cannot be applied always in practice because some VOs might have several DI-GRUBER decision points that overlap partially one another in terms of brokered sites. In such situation, the next case has to be considered.

In the second case, besides uSLAs exchanges, additional reconciliation operations have to be performed [12]. In our implementation, the uSLAs are merged. We do believe that simple merging operations are enough for the MAUI-like rules. In the WS-Agreement-based cases, rules are instead parsed on the fly when needed and if all are satisfied then a set of available sites is generated.

The algorithms used to handle the situations presented above are presented next (they are generic enough to cover both situations):

```
procedure uSLA_combination
        arguments uSLA_set[DPs], local_uSLA
        returns local_uSLA
1 foreach uSLA_set (S) in uSLA_set[DPs] do
2     if S already exists in local_uSLA then
3         update/replace S in local_uSLA
4     else
5         add S to local_uSLA
6     end // if
7 end // foreach
end
```

```
procedure uSLA_parsing
        arguments local_uSLA
        returns final_action
1 foreach rule (R) in the local_uSLA do
3     action = analyze (R)
4     final_action = MIN (action, final_action)
5 end // foreach
end
```

where:

DP	= DI-GRUBER decision point
Action	= the action that to be performed according to the uSLA set R
final_action	= the action that is finally considered
local_uSLA	= uSLA set saved locally

4. Empirical Results

Here we report on some previous results [4] as well as new results achieved through the WS-MDS Index Service infrastructure. We used between one and ten GT4 DI-GRUBER decision points deployed on PlanetLab nodes [13]. Each decision point maintained a view of the configuration of the global DI-GRUBER environment, via periodic exchanges (in the experiments that follow every three minutes) with other decision points of information about recent job dispatch operations. The decision points get information about their neighbors through a predefined Index Service running on a different computer.

The three metrics used in this chapter are **Throughput, Response** and **Accuracy**, defined as follows.

Response is defined by the following formula (with RT_i being the individual job time response and N being the number of jobs processed during the execution period):

$$\mathbf{Response} = \Sigma_{i=1..N} RT_i / N$$

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:

$$\mathbf{Accuracy} = \Sigma_{i=1..N} (SA_i) / N$$

4.1. Previous Results

In the previous experiments, we used composite workloads that overlay work for 60 VOs and 10 groups per VO. The experiment duration was one hour in all cases, and jobs were submitted every second from a submission host. Each of a total of about 120 submission hosts ("clients") maintained a connection with only one DI-GRUBER decision point, selected randomly in the beginning — thus simulating a scenario in which each submission site is associated statically with a single decision point.

The emulated environment was composed of 300 sites representing 40,000 nodes (a grid approximately ten times larger than OSG today). Each site is composed of one or more clusters. The emulated configuration was based on OSG configuration settings in terms of CPU counts, network connectivity, etc [4]. As an additional note, the previous results were achieved on a DI-GRUBER prototyped in a pre-release of GT4.

With three decision points, **Throughput** increases slowly to about 4 job scheduling requests per second when all testing machines are accessing the service in parallel. The service **Response** time is also smaller (about 26 seconds) on average compared with the previous results (about 84 seconds). With 10 decision points, the average service **Response** time decreased even further to about 13 seconds, and the achieved **Throughput** reached about 7.5 queries per second during the peak load period. [4]

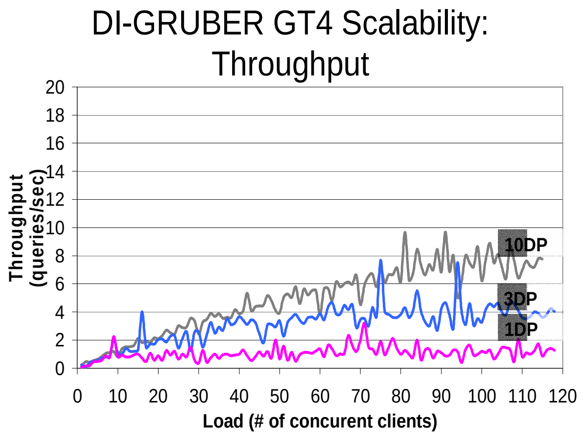


Figure 6: DI-GRUBER Scalability Throughput (1, 3, and 10 DI-GRUBER Decision Points)

As can be observed in Figure 6 and Figure 7, the distributed service provides a symmetrical behavior with the number of concurrent machines that is independent of the state of the grid (lightly or heavily loaded). This result verifies the intuition that for a certain grid configuration size, there is an appropriate number of decision points that can serve the scheduling purposes under an appropriate performance constraint. The overall improvement in terms of throughput and response time is two to three times when a three-decision point infrastructure is deployed, while for the ten-decision point infrastructure the throughput increased almost five times relative to the centralized approach.

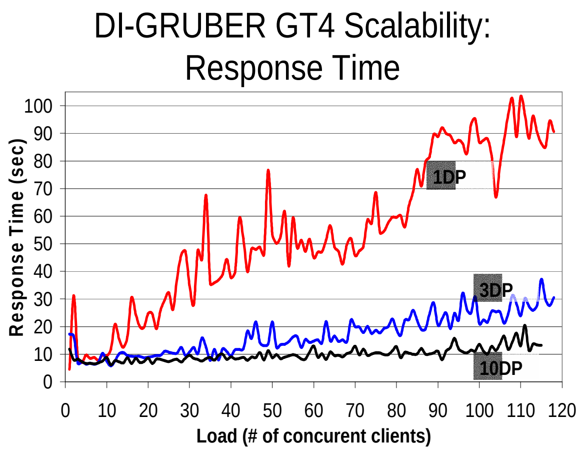


Figure 7: DI-GRUBER Scalability Response Time (1, 3, and 10 DI-GRUBER Decision Points)

However, by means of simulations [4] we have previously concluded that a total of 5 decision points was enough for handling the workloads floating through the framework. The main issue remained here, was to achieve a DI-GRUBER implementation for dynamically detecting infrastructure decision points overloads.

4.2. New Results

Here we report on new experiments we performed using DI-GRUBER on PlanetLab. We have to mention that we used this time a final GT4 release based implementation and at the same time, all peer discovery operations were performed by means of the WS-MDS Index Service running on a dedicated computer. In addition, the clients were configured to re-connect to an available decision point whenever an error occurred.

Again, the composite workloads overlaid work for 60 VOs and 10 groups per VO. The experiment duration was also one hour in all cases, and jobs were submitted every second from a submission host (120 submission hosts again).

4.2.1. Enhanced DI-GRUBER Scalability

The environment was similar as in the previous experiments, the PlanetLab environment [13]. Also, the same set of nodes was used for tests, but 6 months later. The results show some improvement in terms of both **Response** and **Throughput**. Practically, the clients got a better repartition over the decision points, and achieved a more stable response time compared with the previous example. The **Response** metric's value is always less than 30 seconds for 3 decision points, and less than 10 seconds for 10 decision points. The **Throughput** metric's value shows even more improvements, reaching a constant value of 5 queries per seconds for 3 decision points, while goes us up to 16 queries per second for 10 decision points. On average, we found the enhanced DI-GRUBER to offer modest improvements for 3 decision points (19% higher throughput and 8% lower response time) and significant improvements for 10 decision points (68% higher throughput and 70% lower response times).

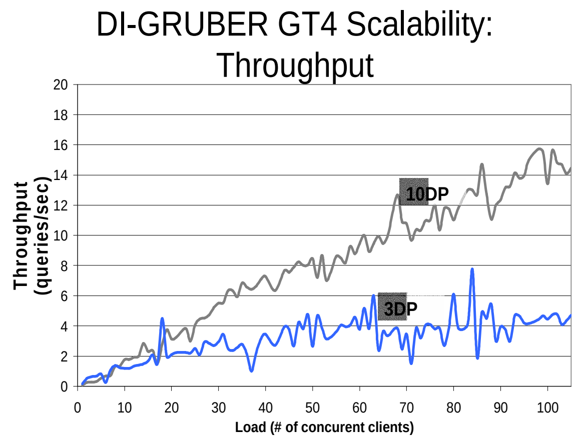


Figure 8: Enhanced DI-GRUBER Scalability Throughput (3, 10 Decision Points)

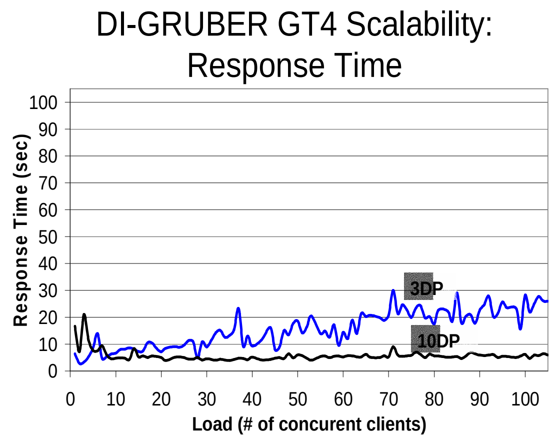


Figure 9: Enhanced DI-GRUBER Scalability Response Time (3, 10 Decision Points)

Next, we present a comparison in terms of the performance of handled jobs during the experiment time interval. The improvements show that practically more jobs were handled in the same time interval by the new DI-GRUBER, by a factor of 1.56 in the 3 DP case and 1.84 in the 10 DP case (see Table 1).

While these results are encouraging from a performance point of view, the main gains are however the capacity of the infrastructure to automatically re-arrange whenever a decision point fails.

Table 1: DI-GRUBER (GT 3.9.5) vs. MDS-based DI-GRUBER (GT 4.0) Performance

Jobs	GT3.9.5 3 DPs	GT3.9.5 10 DPs	GT4 3 DPs	GT4 10 DPs
Handled	24048	37593	31762	69208
Not Handled	1893	2567	3505	10823
Total	25941	40160	35267	80031

4.2.2. Enhanced DI-GRUBER Tests

In order to prove that WS-MDS Index Service-based service is indeed scalable enough to support larger DI-GRUBER infrastructures than considered till now, we have tested the capacity of our framework with 120 decision points. In this case we have measured the regularities of the registrations to the WS-MDS Index Service, as well as, the load on the actual node running the WS-MDS Index Service. These results are presented in Figure 10.

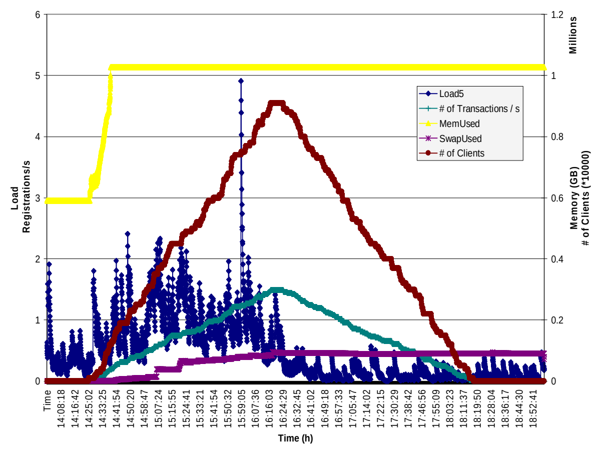


Figure 10: Infrastructure Performance

We can observe here that the higher load on the node running the registration service was higher only during the initial registration of the DI-GRUBER decision points with the main WS-MDS Index Service. Once the registrations stopped, all load on the node dropped to "normal". However, our tests started a new decision point every 60 seconds, case which can not occur in practice often. Also, the memory utilization increased fast, but once all the physical memory was allocated, the swap part increased much slower. As an additional note regarding this behavior, we have to note that in our implementation each decision points registers with the local WS-MDS Index Service, which performs further an up-stream registration with the central service. This approach is helpful in practice because it provides the possibility to duplicate the central registration point, and avoid possible bottlenecks. Of course, the drawback is the higher load on the node running the central WS-MDS Index Service.

4.2.3. Dynamic Bootstrap Signaling

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-MDS 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 based on some simple policies can dynamically start new decision points by means of WS-GRAM service. For example, in the OSG 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 remining time, the dedicated pool might be used for other grid specific operations.

5. Accuracy with Mesh Connectivity

In this section we focus on comparing the performance of the brokering infrastructure function of the connectivity of the decision point connectivity. The comparisons are done by means of the **Accuracy** metric, as defined before. Also, we consider a few cases, as follows: full connectivity (each of the DPs collects information from all the others), half connectivity (each of the DPs collects information only from half of all the others), and one-fourth connectivity (each of the DPs collects information only from a quarter of all the others).

In order to achieve this connectivity, we used practically several WS-MDS Index Service registration points (1 in the first case, 2 in the second case and 3 in the third case). The DI-GRUBER decision points were configured to register to one WS-MDS Index Service while obtaining the list of available peers from a different registration point is such a way to assure full connectivity in 1, 2 or 4, respectively, steps. The results we have obtained by measurement are presented next, after we review our previous results achieved before based on complete static configuration lists.

5.1.1. Previous Results

Next, we present our previous analyzes on the performance of the GT4 DI-GRUBER and its strategies for providing accurate scheduling decisions We present here only the results from an infrastructure complexity point of view, because our next analyze focuses on this problem. [4]

Table 2 depicts the overall performance of GT4 DI-GRUBER in the scenarios introduced in section 4. The values under the "All Requests" section provide an overall view of the implementation's performance (even though these results take in consideration also the 1 and 3 decision points based infrastructure).

Table 2: GT4 DI-GRUBER Overall Performance

	Decision Points	% of Req	# of Req	QTime	Norm QTime	Util	Accuracy
Requests Handled by GRUBER	*1*	53%	3852	0	0.000	3%	98%
	3	92%	24048	452	0.018	16%	90%
	10	93%	37593	2501	0.066	35%	75%
Requests NOT Handled by GRUBER	*1*	47%	3382	0	0.000	7%	-
	3	8%	1893	36	0.019	4%	-
	10	7%	2567	220	0.085	6%	-
All Requests	*1*	100%	7234	0	0.000	10%	94%
	3	100%	25941	660	0.025	20%	81%
	10	100%	40160	3017	0.075	41%	68%

We note that the accuracy drops with the complexity of the infrastructure, while the number of jobs handled by the infrastructure increases substantial (one order from 1 to 10 decision points). [4]

5.1.2. Enhanced DI-GRUBER Results

Now, we present the new results achieved by means of the WS-MDS Index Service based DI-GRUBER and for two additional configurations where the decision points had only partial knowledge about the entire infrastructure. We achieved this by using one central WS-MDS Service where all decision points registered and which was queried by the clients (in order to achieve a good repartition of requests), while the decision points queried 2 (or 3) other services. In this way, the decision point did not have full knowledge about the existence of all the other points in the system. Achieved results are captured in Table 3.

Table 3: WS-MDS based DI-GRUBER Performance

	# of MDS	Util	Accuracy
Requests Handled by GRUBER	1	35%	75%
	2	27%	62%
	3	20%	55%
Total Request	1	41%	68%
	2	30%	60%
	3	21%	50%

We can observe that the performance of the scheduling brokering infrastructure drops substantially with the smaller connectivity of each individual decision point. As an additional note, the Util parameter is low because jobs do not start all in the beginning over the resources, but they are scheduled every second during the entire execution period. Figure 11 provides an intuitive way for realizing that the performance drops almost linearly with the number of WS-MDS Index Services. The relation between the DPs' connectivity (**Con**) and the number of registry services is:

$$\mathbf{Con = ABS \mid D / M \mid}$$

where D is the number of decision points in the system, while M is the number of WS-MDS Index Services used for registration.

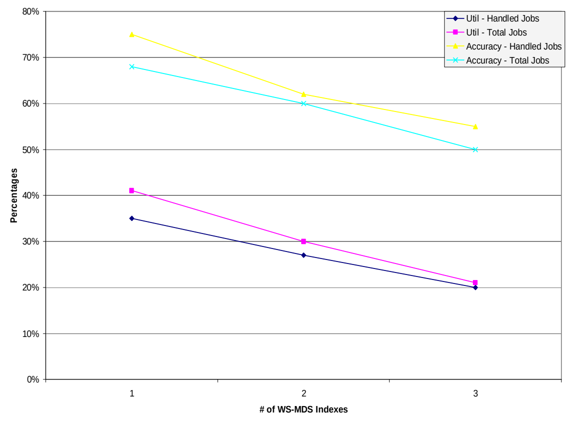


Figure 11: DI-GRUBER Infrastructure Performance based on decision points connectivity

6. Related Work

A policy based scheduling framework for grid-enabled resource allocations is under development at the University of Florida [1]. 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. One important difference of DI-GRUBER is the lack of an assumption of a centralized scheduling point.

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 [14]. An important difference is that GridBus does not support the notions of sites, submission hosts, and virtual organizations or groups.

Cremona is a project developed at IBM as a part of the ETTK framework [15, 16]. 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. DI-GRUBER instead targets a different environment model, in which the main players are VO and resource providers with opportunistic needs (free resources are acquired when available).

7. Conclusions and Future Work

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 [5]. Also, when such environments are also dynamic, the problem becomes even more complex. We have provided here a solution for enhancing DI-GRUBER in order to address the question on how uSLAs can be stored, retrieved and disseminated efficiently in a large and dynamic distributed environment. *The key question this paper addresses is the reconciliation and management of a brokering infrastructure, DI-GRUBER in our case, in large and dynamic Grid environments.*

We note that DI-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 100s of milliseconds, a single query can easily take multiple of seconds to serve. We expect that performance will be significantly better in a LAN environment. However, one of DI-GRUBER's design goals was to offer resource brokering in a WAN environment such as grids.

As previously stated, while the transaction rate for the DI-GRUBER service is fairly low compared to other transaction processing systems, this rate proved to be sufficient in the Grid3 context [17]; furthermore, these other transaction processing systems were designed to be deployed in a LAN environment. Also, the transaction speed increases linearly with the number of decision points deployed over a grid. DI-GRUBER performance can be improved further by porting it to a C-based Web services core, such as is supported in GT4 [18]. The performance of DI-GRUBER could also be enhanced further simply by deploying it in a different environment that would have a tighter coupling between the resource broker (DI-GRUBER) and the job manager (Euryale); this approach would reduce the complexity of the communication from two layers to one layer.

Also, by increasing the number of decision points (cooperating brokers that communicate via a flooding protocol) the throughput climbs to approximately 70 transactions/second with a low response time. This observation leads us to conclude that the required number of "decision" nodes to ensure scalability in a two-layer scheduling system like DI-GRUBER is relatively small.

The WS-MDS Index Service approach proved to improve the capabilities of the DI-GRUBER framework. The performance results presented in section 4 are encouraging and we also showed that DI-GRUBER can scale up to hundreds of decision points, an infrastructure that can handle grids more than 1000 larger than today's OSG size. We also learnt that when each decision point has only a partial view of the brokering infrastructure the brokering infrastructure performance decrease almost linearly with the number of registry services used in the system.

Acknowledgements: This work was supported by the NSF Information Technology Research GriPhyN project, under contract ITR-0086044.

Bibliography

1. In, J. and P. Avery. *Policy Based Scheduling for Simple Quality of Service in Grid Computing*. in *International Parallel & Distributed Processing Symposium (IPDPS)*. April '04. Santa Fe, New Mexico.
2. Dumitrescu, C. and I. Foster. *Usage Policy-based CPU Sharing in Virtual Organizations*. in *5th International Workshop in Grid Computing*. 2004.
3. Dumitrescu, C. and I. Foster. *GRUBER: A Grid Resource SLA Broker*. in *Euro-Par*. 2005. Portugal.
4. Dumitrescu, C., I. Raicu, and I. Foster. *DI-GRUBER: A Distributed Approach for Grid Resource Brokering*. in *SC'05*. 2005. Seattle.
5. Foster, I., *The Grid: A New Infrastructure for 21st Century Science.* Physics Today. 55(2): p. 42-47.
6. Dumitrescu, C., et al. *DiPerF: Automated DIstributed PERformance testing Framework*. in *5th International Workshop in Grid Computing*. 2004.
7. Raicu, I., *A Performance Study of the Globus Toolkit® and Grid Services via DiPerF, an automated DIstributed PERformance testing Framework*, in *Computer Science*. 2005, The University of Chicago: Chicago. p. 100.
8. Foster, I. and others. *The Grid2003 Production Grid: Principles and Practice*. in *IEEE International Symposium on High Performance Distributed Computing*. 2004: IEEE Computer Science Press.
9. Dumitrescu, C., M. Wilde, and I. Foster. *A Model for Usage Policy-based Resource Allocation in Grids*. in *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*. 2005. Stockholm, Sweden.
10. IBM, *WSLA Language Specification, Version 1.0.* 2003.
11. Ludwig, H., A. Dan, and B. Kearney. *Cremona: An Architecture and Library for Creation and Monitoring WS-Agreements*. in *ACM International Conference on Service Oriented Computing (ICSOC'04)*. 2004. New York.
12. Thompson, M.R., *Secure and Reliable Group Communication*. 1999, Lawrence Berkeley National Laboratory: California.
13. Chun B., D.C., T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, *PlanetLab: An Overlay Testbed for Broad-Coverage Services.* ACM Computer Communications Review, 3, July 2003. 33(3).
14. Buyya, R., *GridBus: A Economy-based Grid Resource Broker*. 2004, The University of Melbourne: Melbourne.
15. IBM, *Web Services*. 2002.
16. Dan, A., et al., *Web Services on Demand: WSLA-driven automated management*, S. Journal, Editor. 2004, IBM. p. 136.
17. Foster, I. and e. al. *The Grid2003 Production*

Grid: Principles and Practice. in *13th International Symposium on High Performance Distributed Computing*. 2004.

18. M. Humphrey, G.W., K. Jackson, J. Boverhof, M. Rodriguez, Joe Bester, J. Gawor, S. Lang, I. Foster, S. Meder, S. Pickles, and M. McKeown. *State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations*. in *4th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*. 24-27 July 2005. Research Triangle Park, NC.
19. Voeckler, J., "Euryale: Yet Another Concrete Planner", in *Virtual Data Workshop*, May 18th, 2004.