GRUBER: A Grid Resource Usage SLA Broker

Catalin L. Dumitrescu[1] and Ian Foster[1,2]

[1] Computer Science Department, The University of Chicago,
5801 S. Ellis Ave., Chicago, IL, 60637
cldumitr@cs.uchicago.edu
[2] Mathematics and Computer Science Division, Argonne National Laboratory,
9700 S. Cass Ave., MCS/221, Argonne, IL, 60439
foster@mcs.anl.gov

Abstract. Resource sharing within grid collaborations usually implies specific sharing mechanisms at participating sites. Challenging policy issues can arise in such scenarios that integrate participants and resources spanning multiple physical institutions. Resource owners may wish to grant to one or more virtual organizations (VOs) the right to use certain resources subject to local usage policies and service level agreements, and each VO may then wish to use those resources subject to its usage policies. This paper describes GRUBER, an architecture and toolkit for resource usage service level agreement (SLA) specification and enforcement in a grid environment, and a series of experiments on a real grid, Grid3. The proposed mechanism allows resources at individual sites to be shared among multiple user communities.

1 Introduction

Resource sharing issues arise at multiple levels when sharing resources. Resource owners may want to grant to VOs the right to use certain amounts of their resources, and thus want to express and enforce the usage policies under which these resources are made available. We measure the impact of their introduction by means of *average resource utilization*, *average response time*, *average job completion*, *average job re-planning*, and *workload completion time* [1]. We distinguish here between "resource *usage* policies" (or SLAs) and "resource *access* policies." 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 usage SLAs 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 usage service level agreement (SLA) steps in to govern *how much* of the resource the user is permitted to consume.

GRUBER is an architecture and toolkit for resource usage SLA specification and enforcement in a grid environment. The novelty of GRUBER consists in its capability to provide a means for automated agents to select available resources from VO level on down. It focuses on computing resources such as computers, storage, and networks; owners may be either individual scientists or sites; and VOs are collaborative groups, such as scientific collaborations. A VO [2] 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 [3], a VO corresponds to either one or several users, depending on local access policies. 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.

We focus in this paper on the following questions: "*How usage SLAs are handled in grid environments?*", and *"What is the gain for taking in account such usage SLAs?"*. We build on much previous work concerning the specification and enforcement of local scheduling policies [1],[4],[5],[17], for negotiating SLAs with remote resource sites [6],[7], and for expressing and managing VO policy [8]. We describe in detail the usage SLA problem at several levels, introduce an infrastructure that allows the management of such usage SLAs for virtual and real resources, and compare various approaches for real scenarios encountered in the Grid3 context [3].

2 Motivating Scenario

In the context of grids comprising numerous participants from different administrative domains, controlled resource sharing is important because each participant wants to ensure that its goals are achieved. We have previously defined [1] three dimensions in the usage policy space: resource providers (sites, VOs, groups), resource consumers (VOs, groups, users), and time. Provider policies make resources available to consumers for specified time periods. Policy makers who participate in such collaborations define resource usage policies involving various levels in this space. We extend here the work proposed in [1],[4] with usage policies at the level of virtual organizations and beyond.

Usage SLA specification, enforcement, negotiation, and verification mechanisms are required at multiple levels within such environments. *Owners* want convenient and flexible mechanisms for expressing the policies that determine how many resources are allocated to different purposes, for enforcing those policies, and for gathering information concerning resource usage. *VOs* want to monitor SLAs under which resources are made available.

User and *group jobs* are the main interested parties in resources provided by sites and resources. They use resources in accordance with allocations specified at different level, in a hierarchic fashion and similar to LSF approach at a cluster level. Run-to-completion is a usual mode of operation, because jobs tend to be large, making swapping expensive. Also, workload preemption on several nodes is difficult in a coordinated fashion and even more difficult when a distributed file system is used for data management. Thus, we assume that jobs are preempted only when they violate certain usage rules specified by each individual provider. We note also that site resource managers such as LSF, PBS, and NQS typically only support run-to-completion policies [9].

Algorithms and *policies* capture how jobs are assigned to host machines [1],[4]. The question "*Which is the best approach for different environment models?*" is an old-age question conditioned by many parameters that vary from case to case. Usually what just appears to be an SLA "parameter" can have greater effect on the performance users get from their computing resources than various metrics reported through a monitoring system about resource availabilities.

The problem domain is expressed as follows: a grid consists of a set of resource provider *sites* and a set of *submit hosts*; each site contains a number of processors and some amount of disk space; a three-level hierarchy of *users*, *groups*, and *VOs* is defined, such that each user is a member of exactly one group, and each group is member to exactly one VO; users submit jobs for execution at submit hosts. A job is specified by four attributes: VO, Group, Required-Processor-Time, Required-Disk-space; a *site policy statement* defines site usage SLAs by specifying the number of processors and amount of disk space that sites make available to different VOs; and a *VO policy statement* defines VO usage SLAs by specifying the fraction of the VO's total processor and disk resources (i.e., the aggregate of contributions to that VO from all sites) that the VO makes available to different groups.

Within this environment, the *usage SLA-based resource sharing problem* involves deciding, at each submit host, which jobs to route to which sites, and at what time, for execution, to both (a) satisfy site and VO usage SLAs and (b) optimize metrics such as resource utilization and overall job and workload execution time. We note that this model is one of resource sub-allocation: resources are owned by sites, which apportion them to VOs. VOs in turn apportion their "virtual" resources to groups. Groups could, conceptually, apportion their sub-allocation further, among specific users. Without loss of generality, we simplify both this discussion and our implementation by sub-allocating no further than from VOs to groups.

3 GRUBER Architecture

GRUBER is composed of four principal components, as we describe. The (a) *GRUBER engine* implements various algorithms for detecting available resources and maintains a generic view of resource utilization in the grid. Our implementation is an OGSI service capable of serving multiple requests and based on all the features provided by the GT3 container (authentication, authorization, state or state-less interaction, etc). The (b) *GRUBER site monitoring component* 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. (c) *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. Finally, the (d) *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 overall GRUBER architecture is presented in Fig. 1.

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

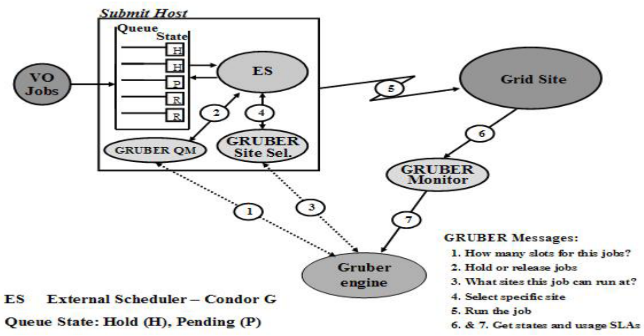

Fig. 1. GRUBER Architecture

3.1 GRUBER Engine

GRUBER decides which sites are *best* for a job by implementing the following logic:

- If there are fewer waiting jobs at a site than available CPUs, then GRUBER assumes the job will start right away if an extensible usage policy is in place [1].
- If there are more waiting jobs than available CPUs or if an extensible usage policy is not in place, then GRUBER determines the VO's allocation, the number of jobs already scheduled, and the resource manager type. Based on this information:
  - o if the VO is under its allocation, GRUBER assumes that a new job can be started (in a time that depends on the local resource manager type).
  - o if the VO is 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.

```
fn get-avail-sites(sites G, VO i, job J)
1. for each site s in G do
2.     # Case 1: site over-used by $VO_i$
3.     if $EA_i$ > $EP_i$ for VO I at site s
4.         next
5.     # Case 2: un-allocated site
6.     else if $_k(BA_k)$at s < s.TOTAL - J &&
                  ($BA_i$ + J < $BP_i$ || extensible $BP_i$) then
7.         add (s, S)
8. return S
```

with the following definitions:

```
S   = Site Set ; k = index for any VO != $VO_i$
$EP_i$ = Epoch Usage SLA for $VO_i$ ; $BP_i$ = Burst SLA for $VO_i$
$BA_i$ = Burst Utilization for $VO_i$ ; $EA_i$ = Epoch Utilization
TOTAL = upper limit allocation on the site
```

The list of possible solutions is further provided as input to a task assignment policy algorithm that makes the actual submission decisions (e.g., round robin).

3.2 GRUBER Queue Manager / Site Selectors

GRUBER queue manager is 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. The site selector component answers: "*Where is best to run next?*", while the queue manager answers: "*How many jobs should group G_m of VO_n V be allowed to run?*" and "*When to start these jobs?*"

The queue manager is important for SLA 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):

```
while (true) do
    if Q != empty
        get j from Q
    else
        next
   S = get-avail-sites(G, Vo(j), j)
   if S != empty
       s = schedule(j, S)
       run(j,s)
```

with the following definitions:

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

3.3 Disk Space Considerations

Disk space management introduces additional complexities in comparison to job management. 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. As a consequence, a different approach has been devised. As a concrete example, a site can become heavily loaded with a one VO jobs and because of which other jobs are either in the local queue in an idle state waiting for their turn. But this does not stop the submission of more jobs.

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 resource manager 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.

3.4 GRUBER Usage SLA Language

In the experiments described in this paper we use a usage SLA representation based on Maui semantics and WS-Agreement syntax [4],[7],[11]. Allocations are made for processor time, permanent storage, or network bandwidth resources, and there are at least two-levels of resource assignments: to a VO, by a resource owner, and to a VO user or group, by a VO. We started from the Maui's semantics in providing support for fair-share rule specification [12]. Each entity has a fair share type and fair share percentage value, e.g., VO_0 15.5, VO_1 10.0+, VO_2 5.0-. The sign after the percentage indicates if the value is a target (no sign), upper limit (+), or lower limit (-).

We extend the semantics slightly by associating both a consumer and a provider with each entry; extending the specification in a recursive way to VOs, groups; and users, and allowing more complex sharing rules. In our approach, $Site_1$ makes its CPU resources available to consumer VO_0 subject to two constraints: VO_0 is entitled to 10% of the CPU power over one month; and with any burst usage up to 40% of the CPU power for intervals smaller than one day. Not all parameters are required and in a recursive fashion, a similar usage SLA is specified for VO entities.

4 Experimental Studies

We now present our experimental results. We first describe the metrics that we use to evaluate alternative strategies, afterwards introduce our experimental environment, and finally present and discuss our results.

4.1 Metrics

We use five metrics to evaluate the effectiveness of the different site selector strategies implemented in GRUBER. **Comp** is the percentage of jobs that complete successfully. **Replan** is the number of replanning operations performed. **Time** is the total execution time for the workload. **Util** is average resource utilization, the ratio of the per-job CPU resources consumed (ET_i) to the total CPU resources available, expressed as a percentage:

$$\text{Util} = \Sigma_{i=1..N} ET_i / (\#_{cpus} * \Delta t) * 100.00$$

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

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

4.2 Experiment Settings

We used a single job type in all our 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 kilobytes of input, and generates about 0.7-1.5 megabytes of output: i.e., an insignificant amount of I/O. We used this BLAST job in two workload different configurations. In 1x1K, we have a single workload of 1000 independent BLAST jobs, with no inter-job dependencies. This workload is submitted once. Finally, in the 4x1K case, the 1x1K 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.

We performed all 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 enforces different usage policies which are collected by our site SLA observation point and used in scheduling workloads. We submit all jobs within the iVDGL VO, under a VO usage policy that allows a maximum of 600 CPUs. Furthermore, we submitted each individual workload under a separate iVDGL group, with the constraint than any group can not get more than 25% of iVDGL CPUs, i.e., 150.

4.3 Results

Table 1 and Table 2 give results for the 1x1K and 4x1K cases, respectively. We see considerable variation in the performance of the various site selectors.

Table 1. Performance of Four GRUBER Strategies for 1x1K

	G-RA	G-RR	G-LRU	G-LU
Comp (%)	97	96.7	85.6	99.3
Replan	1396	1679	1440	1326
Util (%)	12.85	12.28	10.63	14.56
Delay (s)	49.07	53.75	54.69	50.50
Time (hr)	8.19	10.45	22.23	9.25

In the 1x1K case, G-LU does significantly better than the others in terms of jobs completed and G-RA does significantly better than the others in execution time. G-LRU is clearly inferior to the others in both respects. Note that as a single job runs for about one hour, the minimum possible completion time is ~1000/150 ≈ 6.66 hours. Thus the best execution time achieved is ~22% worse than the minimum, which is an acceptable result. Note also the relatively high number of replanning events in each case (a mean of ~1.5 per job), another factor that requires further investigation.

Table 2. Performance of Four GRUBER Strategies for 4x1K

	G-RA	G-RR	G-LRU	G-LU
Comp (%)	98.2	98.7	87.9	91.7
Replan	1815	1789	1421	2409
Util (%)	13.51	14.02	11.05	11.52
Delay (s)	66.62	64.41	68.97	63.96
Time (hr)	11.21	10.5	13.51	13.49

In the 4x1K case, results are somewhat different. Each submitter is allocated a limit of 25% of the VO's total resource allocation of 600 CPUs. Thus, in principle, all four submitters can run concurrently, but in practice we can expect greater contention in site queues and when sites are overloaded with non-GRUBER work.

The results in Table 4 are the means across the four submitters. We see some interesting differences from Table 3. G-LU's completion rate drops precipitously, presumably for some reason relating to greater contention. Replanning events increase somewhat, as does scheduling delay. The total execution times for G-RA and G-LU increase, although more runs will be performed to determine whether these results are significant. Fig. 2 provides a more detailed picture of the job completion rates for the experiments of Table 4. We see that the inferior performance of G-LRU and G-LU is accentuated by the long time spent on a few final jobs.

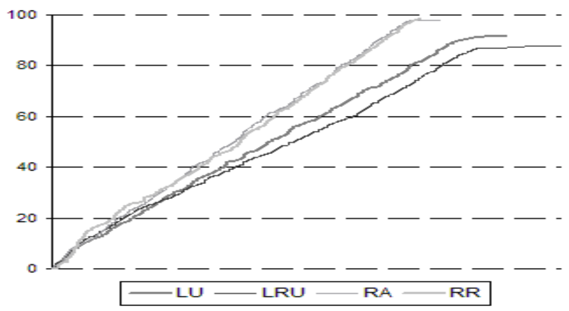

Fig. 2. Job Completion Percentage vs. Time

4.4 Results Variance

In practice, we may expect to see considerable variation in results over different runs, due to our lack of full control over the Grid3 environment. As a first step towards quantifying this variation, we ran ten identical 1x1K runs with the random assignment strategy (G-RA). Our results, in Table 3, show considerable variation for some quantities (Replan, Util, and Delay) and little variation for other quantities (Comp and Time). However, the small variance in completion time and later studies not reported here confirm our observations.

Table 3. 10 1x1K Runs with G-RA

Metric	Average	Std. Deviation	Std. Dev. As %
Comp (%)	95.11	1.23	1.3
Replan	2000	308	15.4
Util (%)	18.77	3.10	16.5
Delay (s)	78.05	15.39	19.7
Time (hr)	7.97	0.33	4.1

4.5 Site Selector Variations and Comparisons

We compare GRUBER performance with that of two other methods. In the first of the two methods, “GRUBER observant” or G-Obs, a site selector associates a further job to a site each time a job previously submitted to that site has started. In effect, the site selector fills up what a site provides by associating jobs until site’s limit is reached. The second alternative, S-RA, associates each job to a site selected at random.

Table 4 shows the results obtained on Grid3 for the 1x1K workload. We find that the best standard GRUBER site selector achieves a performance comparable to that of the GRUBER observant (G-Obs) selector.

Table 4. G-LU, G-Obs and S-RA Strategies: Performance for 1x1K

	G-LU	**G-Obs.**	**S-RA**
Comp (%)	99.3	97.3	60.2
Replan	1326	284	1501
Util (%)	14.56	12.59	0.57
Delay (s)	50.50	62.01	121.0
Time (h)	9.25	11.2	22.3

The results indicate that GRUBER's automated mechanism for SLAs tuning (as described in Section 3.2) makes its site selectors comparable in terms of job scheduling performance. On the other hand, compared with GRUBER site selectors, the naive random assignment site selector policy performs two to three times worse in terms of our metrics. An important metric to observe is the number of re-planning operations. While the observant site selector had around 300 operations, GRUBER performed around 1300 re-scheduling operations and the naive site selector around 1500. The difference comes from the fact that the simple round robin algorithm selects from all the sites and not only from the candidates identified as available.

5 Related Work

Fair share scheduling strategies seek to control the distribution of resources to processes so as to allow greater predictability in process execution. These strategies were first introduced in the early 1980s in the context of mainframe batch and timeshared systems and were subsequently applied to Unix systems [13],[14].

Irwin et al. investigate the question of scheduling tasks according to a user-centric value metric – called utility [6]. Sites sell the service of executing tasks instead of raw resources. The entire framework is centered on selling and buying services, with users paying more for better services and sites paying penalties when they fail to honor the agreed commitments. The site policies are focused on finding winning bids and schedule resource accordingly. This approach is different from our work here, as a more abstract form of resources is committed under different SLAs.

In et al. describe a novel framework for policy based scheduling of grid-enabled resource allocations [15]. The framework has several features. First, the scheduling strategy can control the request assignment to grid resources by adjusting resource usage accounts or request priorities. Second, efficient resource usage management is achieved by assigning usage quotas to intended users. Third, the scheduling method supports reservation based grid resource allocation. Fourth, Quality of Service (QoS) feature allows special privileges to various classes of requests, users, groups, etc. The difference with our approach consists in the fact that we do not assume a centralized point of usage policy specification, but a more distributed approach of specification and enforcements at the site level.

6 Conclusions

We have presented and evaluated an approach to representing and managing resource allocation policies in a multi-site, multi-VO environment. We also introduced a grid resource broker, called GRUBER, and the experiments we performed with several approaches in task assignment policies. GRUBER is an architecture and toolkit for resource usage SLAs specification and enforcement in a grid-like environment. It is the prototype of a VO policy enforcement point as described by Dumitrescu et al. [1]. While the results presented here are preliminary, they are encouraging and some of the methods described here are pursued further in the Grid3 and OSG contexts.

There are still problems not fully explored in this article. For example, our analysis did not consider the case of cluster administrators that *over-subscribe* local resources, in the sense of a local policy that states that 40% of the local CPU power is available to VO_1 and 80% is available to VO_2. A second issue not discussed in this report is the hierarchic grouping and allocation of resources based on policy. Generally, VOs will group their users under different schemes. While this is an important problem for our context, we leave it as an open problem at the current stage.

References

1. Dumitrescu, C. and I. Foster. "Usage Policy-based CPU Sharing in Virtual Organizations". in *5th International Workshop in Grid Computing*. 2004.
2. Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", in *International Journal of Supercomputer Applications*, 2001. 15(3): p. 200-222.
3. Foster, I., et al., "The Grid2003 Production Grid: Principles and Practice", in *13th International Symposium on High Performance Distributed Computing*. 2004.
4. Dan, A., C. Dumitrescu, and M. Ripeanu. "Connecting Client Objectives with Resource Capabilities: An Essential Component for Grid Service Management Infrastructures", in *ACM International Conference on Service Oriented Computing (ICSOC'04)*. 2004. NY.
5. Altair Grid Technologies, LLC, *A Batching Queuing System,* Software Project, 2003.
6. Irwin, D., L. Grit, and J. Chase., "Balancing Risk and Reward in a Market-based Task Service", in *13th International Symposium on High Performance Distributed Computing*.
7. IBM, *WSLA Language Specification, Version 1.0.* 2003.
8. Pearlman, L., et al. "A Community Authorization Service for Group Collaboration", in *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*. '02.
9. Schroeder, B. and M. Harchol-Balter. "Evaluation of Task Assignment Policies for Super Computing Servers: The Case for Load Unbalancing and Fairness", in *Cluster Computing*.
10. Legrand, I.C., et al. "MonALISA: A Distributed Monitoring Service Architecture", in *Computing in High Energy Physics*. 2003. La Jolla, CA.
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. Cluster Resources, Inc., *Maui Scheduler*, Software Project, 2001-2005.
13. Henry, G.J., *A Fair Share Scheduler.* AT&T Bell Laboratory Technical Journal, 1984.
14. Kay, J. and P. Lauder, "A Fair Share Scheduler", University of Sydney, AT&T Bell Labs.
15. In, J., P. Avery, R. Cavanaugh, and S. Ranka, "Policy Based Scheduling for Simple Quality of Service in Grid Computing", in *International Parallel & Distributed Processing Symposium (IPDPS)*. April '04. Santa Fe, New Mexico.
16. Buyya, R., "GridBus: A Economy-based Grid Resource Broker", 2004, The University of Melbourne: Melbourne.
17. Dumitrescu, C. and I. Foster, "GangSim: A Simulator for Grid Scheduling Studies", in *Cluster Computing and Grid (CCGrid),* 2005, Cardiff, UK.