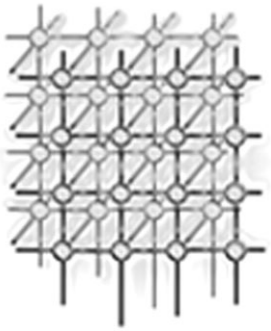
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2007; **19**:945–963
Published online 28 September 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1091

Usage SLA-based scheduling in Grids



Catalin L. Dumitrescu[1,*,†], Ioan Raicu[2] and Ian Foster[2,3]

[1]*Department of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands*
[2]*Computer Science Department, The University of Chicago, 5801 South Ellis Avenue, Chicago, IL 60637, U.S.A.*
[3]*Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, MCS/221, Argonne, IL 60439, U.S.A.*

SUMMARY

Managing usage service level agreements (uSLAs) within environments that integrate participants and resources spanning multiple physical institutions is a challenging problem. Running workloads in such environments is often a similarly challenging problem owing to the scale of the environment, and to the resource partitioning based on various sharing strategies. Also, a resource may be taken down during a job execution, be improperly set up or fail job execution. Such elements have to be taken into account whenever targeting a Grid environment for problem solving. In this paper we explore uSLA-based scheduling on a real Grid, Grid3, by means of a specific workload (the BLAST workload) and a specific scheduling framework, GRUBER (an architecture and toolkit for resource uSLA specification and enforcement). The paper provides extensive experimental results and comparisons with other scheduling strategies. We also address, in great detail, the performance of different uSLA-based site selection strategies and the overall performance in scheduling workloads over Grid3 with workload sizes ranging from 10 to 10 000 jobs. Copyright © 2006 John Wiley & Sons, Ltd.

Received 17 February 2006; Accepted 5 May 2006

KEY WORDS: Grid computing; scheduling; USLAs

1. INTRODUCTION

The motivating scenarios for our work are large Grid environments in which *providers* wish to grant *consumers* the right to use certain *resources* for some agreed-upon time period. Providers might be companies providing outsourcing services, or scientific laboratories that provide

*Correspondence to: Catalin L. Dumitrescu, Department of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands.
†E-mail: c.dumitrescu@ewi.tudelft.nl

Contract/grant sponsor: European Commission CoreGrid IST Project; contract/grant number: 004265
Contract/grant sponsor: NSF Information Technology Research GriPhyN Project; contract/grant number: ITR-0086044

different collaborations with access to their computers or other resources. Providers and consumers 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. Issues involving usage service level agreements (uSLAs) can arise at multiple levels in such scenarios. Providers want to express (and enforce) the 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.

1.1. Problem domain and target environment

The problem domain can be expressed as follows [1,2]: 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 virtual organizations (*VOs*) is defined, such that each user is a member of exactly one group, and each group is member to exactly one VO; and 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 uSLAs 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 uSLAs 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.

Grid3 represents such a multi-VO that sustains production level services required by various physics experiments. The infrastructure is composed of more than 30 sites and 4500 CPUs, over 1300 simultaneous jobs and more than 2 TB a day. The participating sites are the main resource providers under various conditions. We consider in this paper that all of these sites are governed by uSLAs [3–7].

We distinguish here between 'resource *usage* policies' (or uSLAs) and 'resource *access* policies' [6,8]. Resource access policies typically enforce authorization rules. 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 a resource access policy, then the resource usage policy steps in to govern *how much* of the resource the user is permitted to consume. GRUBER is an architecture and toolkit for resource uSLA specification and enforcement in a Grid environment. The novelty of GRUBER consists of its capability to provide a means for automated agents to select available resources (i.e. computers, storage, and networks) from VO level on down. Owners may be either individual scientists or sites, while VOs are collaborative groups (i.e. scientific collaborations). A VO is a group of participants who seek to share resources for some common purpose. From the perspective of a single site in Grid3, 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.

In this paper, we focus on the problems that can occur in Grid3, because various elements affect workload execution times. We measure the impact by means of *average resource utilization*, *average response time*, *average job completion*, *average job re-planning* (Replan), *workload completion time* (Time), and *job completion gain* (Speedup) [2,9]. We present our detailed results for scheduling BLAST workloads over one of the largest U.S. Grids, the Grid3 environment [10].

DOI: 10.1002/cpe

1.2. Goals

The specification, enforcement, negotiation, and verification mechanisms of uSLAs are required at multiple levels within Grid3 [1]. *Sites* 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 uSLAs 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 levels, in a hierarchic fashion and similar to the 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 pre-emption on several nodes is difficult in a co-ordinated fashion and even more difficult when a distributed file system is used for data management. Thus, we assume that jobs are pre-empted only when they violate certain usage rules specified by each individual provider. We note also that site resource managers such as LSF [11], PBS [5], and NQS typically only support run-to-completion policies.

Algorithms and *policies* [12] capture how jobs are assigned to host machines. The question '*Which is a good scheduling strategy for different environments?*' is an age-old question conditioned by many parameters that vary from case to case. Usually what appears to be a uSLA 'parameter' can have a greater effect on the performance that users enjoy from their computing resources than various metrics reported through a monitoring system about resource availabilities.

1.2.1. uSLAs, resource providers, and consumers in Grid3

Thus, running various size workloads over Grid3 can become a challenging problem when resources are shared under various constraints [1,13]. For any environment, even short periods of overloading can decrease the performance that the users enjoy from the system. In any Grid context, such overloading scenarios can be only partial as a Grid is a large composition of resources spread in various administrative domains. Here, we try to identify some of the main challenges users may face when submitting workloads in such environments and to provide also some simple means for improving their achieved performance. We assume in our scenario that various sites are used at particular times and we are interested in measuring how well the overall performance is maintained over larger time intervals.

Grid3 is composed of resources provided based on various uSLA [1]. Furthermore resources are aggregated at the VO level and provided on similar uSLA means to groups and users. Our uSLA scheduling framework, GRUBER, deals with two classes of entity: resource providers and resource consumers. A physical site is a resource provider; a VO is a consumer (consuming resources provided by a site) and a provider (providing resources to users, groups or workload types). We assume that each provider–consumer relationship is governed by an appropriate SLA.

1.2.2. uSLA-based resource sharing

The entire Grid environment is described 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.

DOI: 10.1002/cpe

- A three-level hierarchy of *users*, *groups*, and *VOs* is defined such that each user is a member of one group, and each group is a member of 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 uSLAs by specifying the number of processors and amount of disk space that the site makes available to each VO.
- A *VO policy statement* defines VO uSLAs 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.

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.

2. BACKGROUND INFORMATION AND RELATED WORK

Some background information on the execution environment is important to understand better the experimental setup and results. The execution environment is based on the VDS toolkit developed in the GriPhyN project context. In the next few sections, we discuss Euryale as a concrete planner, GRUBER as a resource broker, and disk space considerations. Finally, we discuss related work.

2.1. Euryale as a concrete planner

Euryale [14] is a complex system aimed at running jobs over a Grid and, in particular, over Grid3 [10]. The approach used by Euryale is to rely on the Condor-G capabilities to submit and monitor jobs at sites. It takes a late binding approach in assigning such jobs to sites. In addition, Euryale allows a simple mechanism for fault tolerance by means of job re-planning when a failure is discovered.

During a workload execution, DagMan executes the Euryale's pre- and postscripts, the heart of the Euryale concrete planner, which also contain the Euryale's execution logic. The prescript calls out to the external site selector, rewrites the job submit file, 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. To run things in the Grid, Euryale needs knowledge about the available resources, or sites. An important feature of Euryale is its capacity to invoke external site selectors, such as our resource broker, GRUBER, in job scheduling [15].

2.2. GRUBER as a resource broker

GRUBER is the scheduler we used for the site selection. It is composed of four principal components [15]. The *GRUBER engine* represents the main component of the architecture. It implements various algorithms for detecting available resources and maintains a generic view of

DOI: 10.1002/cpe

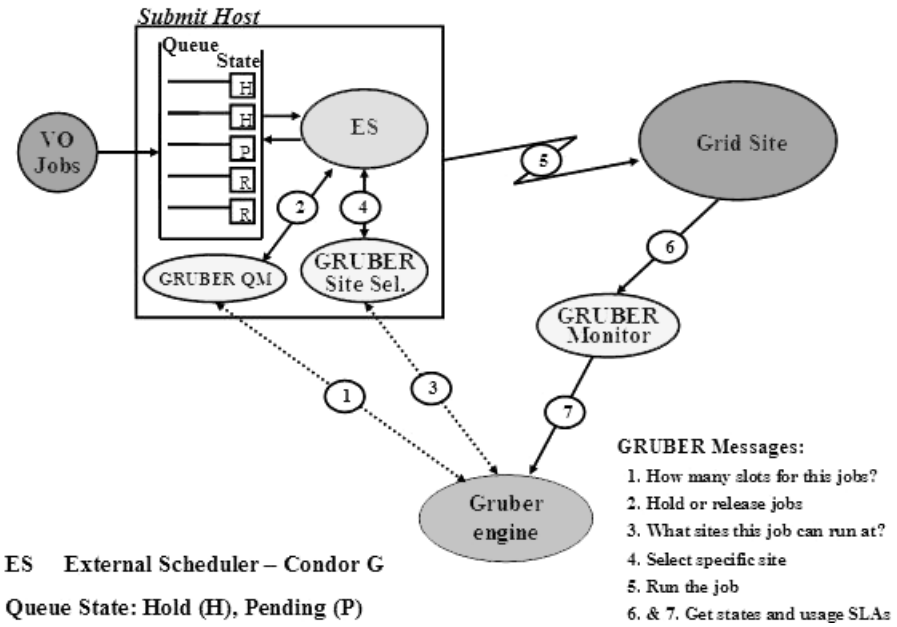


Figure 1. GRUBER architecture.

resource allocations and utilizations. The *GRUBER site monitoring component* is one of the data providers for the GRUBER engine. This component is optional and can be replaced with various other Grid-monitoring components that will provide similar information, such as MonaLisa [16] or Grid Catalog [10]. So far, we are unaware of any complete replacement. *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 random assignment (*G-RA*), round robin (*G-RR*), least used (*G-LU*), or last recently used (*G-LRU*) task assignment policies. The overall GRUBER architecture is presented in Figure 1.

We note that one can use just the GRUBER engine and site selectors, without the GRUBER queue manager. This option makes GRUBER only a site recommender, without having the capacity to enforce any uSLA expressed at the VO level, while enforcing the uSLA at the site level by means of removing a site for an already over-quota VO user at that site.

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 that the job will start right away if an extensible usage policy is in place [9].
- 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:

DOI: 10.1002/cpe

- 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);
- 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** $\Sigma_k(BA_k)$at s < s.TOTAL - J **&&** ($BA_i + J < BP_i$ || **extensible** BP_i) **then**
7. add (s, S)
8. **return** S

Here S is the Site Set, k is the index for any VO ! = VO_i, EP_i is the Epoch uSLA for VO_i, BP_i is the Burst service level agreement (SLA) for VO_i, BA_i is the Burst Utilization for VO_i, EA_i is the Epoch Utilization and TOTAL is the 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).

2.3. Disk space considerations

Disk space management introduces additional complexities in comparison to job management [15]. For the experiments in this paper, we have extended GRUBER to deal with multiple resource issues and here we describe our work in more detail. If an entitled-to-resources job becomes available, it is usually possible to delay scheduling other jobs, or to pre-empt 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 one VO job and consequently other jobs are in the local queue in an idle state waiting for their turn. This does not stop the submission of more jobs. As a result, there may be a great deal of other input data on the site and the disk used space will keep on growing. On the other hand, the other jobs do not have the opportunity to finish and delete their files. The rate of input data being copied over to the site is higher than the rate of completion of jobs, making the disk space to fill up.

So far, we have considered a UNIX quota-like approach. Usually, quotas simply prevent one user on a static basis from using more than his hard 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). 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.

DOI: 10.1002/cpe

1. **for each** site s **in** site list G **do**
2. # Case 1: *over-used site by VO$_i$*
3. **if** $IA_i > IP_i$ for VO i at site s
4. **next**
5. # Case 2: *un-allocated site*
6. **else**
7. **if** $\Sigma_k(IA_k) < s.TOTAL - J$ **&&** $IA_i + J < IP_i$ **then**
8. add (s, S)
9. **return** S

Here S is the Site Set, k is the index for any VO ! = VO_i, IP_i is the Instantaneous uSLA for VO_i, IA_i is the Instantaneous Resource Usage for VO_i and TOTAL is the upper limit allocation on the site.

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.

2.4. Related work

There are several other production workloads running over Grid3, such as the QuarkNet Project, SDSS/CoAdd, GADU, or fMRIDC. While these workloads are important from the GriPhyN project point of view, they offer few elements for comparison with the work described here. First, these workloads are run mostly for their results and not for measuring various Grid3 execution capacities. The closest workload in scope is SDSS/CoAdd; however, we do not to date have information about various metrics [17]. In addition to the iVDGL workloads running over Grid3, there are other challenging problems to solve. For example, the ATLAS 'VO' and applications focus on Monte Carlo simulation of the physics processes that will occur in high-energy proton collisions at LHC; SDSS runs various problems related to galaxy clusters identification or pixel-level analysis of astronomical data, etc. [10]. Regarding similar tools to GRUBER for job steering and scheduling over a Grid, the most similar tools in intent, to the authors' knowledge, are the Pegasus framework [18] and KOALA Grid scheduler [19].

Pegasus is designed with the idea in mind that Grid applications today are no longer monolithic codes; rather, they are being built from existing application components [18]. These applications are defined by workflows, where the activities in the workflow are individual application components and the dependencies between the activities reflect the data and/or control flow dependencies between the components. A workflow can be described in an abstract form, in which the workflow activities are independent of the Grid resources used to execute the activities. Pegasus takes the abstract workflow and maps it practically to the available Grid resources. The concrete workflow is afterwards given to Condor's DAGMan [20] for execution.

KOALA is a Grid scheduler that has been designed, implemented, and deployed by the PDS group in Delft on the DAS-2 multicluster system in the context of the Virtual Lab for e-Science project. The main feature of KOALA is its support for co-allocation, the simultaneous allocation of resources in multiple clusters of the DAS to single applications that consist of multiple components. Currently, KOALA supports processor and data co-allocation in that it starts the components of a single application on different clusters at the same time, and it transfers the input files of the components to their proper locations prior to application execution.

The Grid Service Broker [21], a part of the GridBus Project, mediates access to distributed resources by:

DOI: 10.1002/cpe

(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;
(f) collating and presenting results.

The broker supports a declarative and dynamic parametric programming model for creating Grid applications [21]. An important difference is that GridBus does not yet support the notions of sites, submission hosts, and virtual organizations or groups.

3. EXPERIMENTAL SETUP

The results default in this paper focus on three dimensions:

- we pursue the path of comparing three scheduling alternatives in order to measure uSLA-based resource scheduling performance;
- we focus on the expected performance when running workloads of various sizes over Grid3;
- we compare the performance of GRUBER-provided scheduling policies in order to discover which one is more appropriate in our environment.

In the following sections, we introduce the metrics [15] that we use to evaluate the alternative strategies, and our experimental environment.

3.1. Metrics

We use five metrics to evaluate the effectiveness of workload performance execution. 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.

$$\text{Comp} = (\text{Completed Jobs})/\#_{\text{jobs}} \times 100.00$$

- Replan: the number of performed replanning operations.
- Util: average resource utilization, the ratio of the per-job CPU resources consumed (ET_i) to the total CPU resources available as a percentage:

$$\text{Util} = \Sigma_{i=1,\ldots,N}\text{ET}_i/(\#_{\text{cpus}} \times \Delta t) \times 100.00$$

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

$$\text{Delay} = \Sigma_{i=1,\ldots,N}\text{DT}_i/\#_{\text{jobs}}$$

- Time: the total execution time for the workload.
- Speedup: the serial execution time to the Grid execution time for a workload. Note that the definition implies a comparison between the running times on similar resources—practically, the first time in sequence and the second time in parallel.
- Spdup75: the serial execution time to the Grid execution time for 75% of the workload.

DOI: 10.1002/cpe

Table I. Usage SLAs in our scenario.

VO	Target	Current	Demand	Level
USCMS	60	50	50	OK
USATLAS	20	15	30	Under
IVDGL	10	10	100	OK
LIGO	5	3	3	OK
SDSS	5	22	50	Over

All metrics are important: an adequate environment will both maximize delivered resources and meet owner intents. For example, Table I presents a simple case scenario in which several VOs have various SLAs, resource requests, and resource utilizations at a particular resource provider. The column names have the following meanings.

- Target: the usage SLA for the consumer at the resource provider, as a percentage of site capacity;
- Current: the current utilization for the consumer at the provider, as a percentage;
- Demand: the current resource demand from the consumer for the provider, as a percentage (it cannot be less than Current);
- Level: how our criteria are met (OK when Current = MIN(Target, Demand)).

3.2. Experimental setup

We used a single job type in all of our experiments—the sequence analysis program BLAST. BLAST has been widely and routinely used for sequence analysis and represents the essential component in most of bioinformatics and life science applications. A single BLAST job has an execution time of around 40 min (the exact duration depends on the CPU), reads around 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 these BLAST workloads in three different sets of workload configuration:

(1) small workloads of 10, 50, and 100 jobs that are scheduled at once;
(2) medium workloads of 500 to 1000 jobs that are submitted in several steps in order to honor the VO usage SLAs;
(3) large workloads of 10 000 jobs.

We also have to note that these workloads were practically borrowed from bioinformatics people that used them for their production on Grid3.

We performed all experiments on Grid3, which comprises around 30 sites across the U.S.A., of which we used 15. Each site is autonomous and managed by different local resource managers, such as Condor, PBS, or LSF. Each site enforces different usage policies that are collected by the GRUBER site monitor. For example, Table II gives CPU allocations per VO on five Grid3 sites on 9 July 2004 as collected through the GRUBER site monitor. We submitted all jobs within the iVDGL VO, under a VO usage policy that allows a maximum of 600 CPUs. Furthermore, we submitted each individual

DOI: 10.1002/cpe

Table II. Grid3 CPU allocations on 9 July 2004.

		VO allocations (%)		
Site name	Number of CPUs	iVDGL	Atlas	USCMS
T2cms0.sdsc.edu	76	0.62	24.74	0.40
nest.phys.uwm.edu	305	0.00	7.28	0.00
uscmsb0.ucsd.edu	3	11.68	11.68	11.68
xena.hamptonu.edu	1	25.00	25.00	25.00
garlic.hep.wisc.edu	101	3.01	3.01	3.01

workload under a separate iVDGL group, with the constraint than no one group can get more than 25% of iVDGL CPUs, i.e. 150.

We also configured GRUBER to employ a re-planning policy, by which a starving job was removed after a predefined time interval (20 min here) and resubmitted for rescheduling. If a job was submitted unsuccessfully 10 times or it was reported as application level 'failure' by a site, then it is considered a failure. All submissions were performed without withholding or setting any special priorities at sites; practically, GRUBER had to find resources while all workloads ran in parallel. The VO usage SLA limited the submitting group to approximately 150 jobs at a time.

Regarding the site selectors, they were already introduced by Dumitrescu *et al.* [15] and used in a similar fashion. In summary, G-RA represents GRUBER random-assignment site strategy, G-RR represents GRUBER round-robin site strategy, G-LU represents GRUBER least-used site assignment strategy, and G-LRU represents least-recently-used site strategy [15].

4. WORKLOAD EXECUTION RESULTS

Next, we focus on the execution results achieved on Grid3 for the BLAST workloads. It is important to note that Speedup represents one of the most important metrics in our view, because it shows how fast jobs complete over Grid3 instead of running on a single computer with power comparable to any of the nodes in Grid3.

4.1. Scheduling policy comparisons

First, we present some of our previous results about the comparison of GRUBER performance in scheduling jobs over Grid3 with other two methods. The first of the two other methods, 'GRUBER observant' (G-Obs), is a site selector that 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 the site's limit is reached. The second alternative, S-RA, associates each job to a site selected at random. Also, note that for this set of experiments we used a more aggressive termination condition; namely, a job was allowed to be re-planned only three more times [15].

Table III contains our previous results obtained on Grid3 for a 1×1000 workload [15]. As mentioned, we have found out that the second-best standard GRUBER site selector achieves

DOI: 10.1002/cpe

Table III. G-LU, G-Obs and S-RA strategies: performance for 1×1000.

	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

a performance comparable to that of the GRUBER observant (G-Obs) selector. The results indicated that GRUBER's automated mechanism for uSLA tuning 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 performed two to three times worse in terms of our metrics. An important metric to observe was the number of re-planning operations. While the observant site selector had about 300 operations, GRUBER performed around 1300 re-scheduling operations and the naive site selector about 1500. The difference comes from the fact that the simple round-robin algorithm selects from all of the sites and not only from the candidates identified as available.

4.2. Execution performance

We now turn to the second set of experiments we have performed; namely, the expected performance capturing. We classify these results in three subsets: small (1–100 jobs), medium (101–999 jobs) and large (more than 1000 jobs) workloads.

4.2.1. Small workload results

Table IV shows the results for the 1×10 jobs workloads. In the ideal case, these values are: Comp = 100, Replan = 0, Util = 1.25, Delay = 0, Time = 3000, and Speedup = 10. As can be seen, the speedup is 2.5 to 3.5 times smaller, owing to the probability of jobs ending on sites with a local resource manager that does not behave as expected. The job starvation was 'detected' after a time interval comparable with the execution time (20 min versus 40 min). However, 75% of the jobs do complete in a time interval closer to the ideal case of a speedup of 10.

Table V shows the results for the 1×50 jobs workloads. In the ideal case, these values are: Comp = 100, Replan = 0, Util = 6.25, Delay = 0, Time = 3000, and Speedup = 50. The same situation as before was encountered in this set of experiments: several jobs starved and their execution time affected the overall speedup. The speedup of 75% of the jobs instead is more than the half of the ideal speedup, proving that most of the jobs do complete close to the optimal time.

Table VI shows the results for the 1×100 jobs workloads. In the ideal case, these values are: Comp = 100, Replan = 0, Util = 12.50, Delay = 0, Time = 3000, and Speedup = 100. Again, similarly to the 1×50 workloads, the execution performance is half for 75% of the workloads and drops for the entire workload.

DOI: 10.1002/cpe

Table IV. Results and 90% confidence intervals of four policies for 1 × 10 workloads.

	G-RA	G-RR	G-LU	G-LRU
Comp (%)	100	100	100	100
Replan	34.1 ± 5.51	47.5 ± 9.26	8.6 ± 1.83	13.6 ± 2.18
Util (%)	0.36 ± 0.05	0.31 ± 0.07	0.55 ± 0.10	0.50 ± 0.04
Delay (s)	3262 ± 548	4351 ± 824	1162 ± 376	801 ± 313
Time (s)	12 436 ± 1191.4	13 966 ± 2208.8	8787 ± 158	7653 ± 205.9
Speedup	2.33 ± 0.25	2.21 ± 0.35	3.6 ± 0.6	3.46 ± 0.45
Spdup75	3.72 ± 0.59	3.46 ± 0.51	5.32 ± 0.67	5.66 ± 0.55

Table V. Results and 90% confidence intervals of four policies for 1 × 50 workloads.

	G-RA	G-RR	G-LU	G-LRU
Comp (%)	100	100	100	100
Replan	35 ± 14	51.1 ± 28	48.8 ± 10.8	78.8 ± 9.51
Util (%)	1.18 ± 0.25	1.44 ± 0.27	1.89 ± 0.43	1.76 ± 0.18
Delay (s)	1420 ± 713	583 ± 140.4	653.8 ± 202	1260 ± 528.7
Time (s)	8035 ± 990.4	9654 ± 603.5	8549 ± 898	9702 ± 1247.3
Speedup	16.35 ± 1.17	14.12 ± 0.90	15.16 ± 2.42	12.76 ± 0.71
Spdup75	30.84 ± 5.70	35.36 ± 2.79	35.41 ± 2.48	24.36 ± 2.28

Table VI. Results and 90% confidence intervals of four policies for 1 × 100 workloads.

	G-RA	G-RR	G-LU	G-LRU
Comp (%)	100	100	100	100
Replan	228.7 ± 21	39.9 ± 13.8	124.7 ± 17	230 ± 20.3
Util (%)	2.86 ± 0.30	3.48 ± 0.59	3.51 ± 0.7	1.87 ± 0.46
Delay (s)	1691 ± 198	529 ± 92.67	640 ± 93.4	1244 ± 387.9
Time (s)	10 350 ± 565.9	9013 ± 1025.1	9716 ± 1130	7507 ± 2325.1
Speedup	22.43 ± 1.55	30.15 ± 3.43	28.02 ± 5.4	19.24 ± 1.56
Spdup75	47.38 ± 3.24	77.19 ± 3.26	73.54 ± 2.0	35.86 ± 3.72

DOI: 10.1002/cpe

Table VII. Results and 90% confidence intervals of four policies for 1 × 500 workloads.

	G-RA	G-RR	G-LU	G-LRU
Comp (%)	100	100	100	100
Replan	925 ± 103.5	816 ± 245.6	680 ± 139.3	1024 ± 154.2
Util (%)	34.04 ± 4.55	33.19 ± 2.39	30.3 ± 4.7	25.41 ± 5.6
Delay (s)	9202 ± 1716.8	6700 ± 816.6	6169 ± 407	9125 ± 6117.8
Time (s)	28 116 ± 2881	24 225 ± 035.9	21 362 ± 1250	20 434 ± 4100
Speedup	67.32 ± 5.6	60.22 ± 3.26	63.12 ± 3.41	51.77 ± 5.94
Spdup75	98.43 ± 8.7	111.69 ± 9.81	113.2 ± 8.82	101.48 ± 10.05

Table VIII. Results of four GRUBER strategies for 1 × 1000 workloads.

	G-RA	G-RR	G-LU	G-LRU
Comp (%)	97	96.7	99.3	85.6
Replan	1396	1679	1326	1440
Util (%)	12.85	12.28	14.56	10.63
Delay (s)	49.07	53.75	50.50	54.69
Time (s)	29 484	37 620	33 300	80 028
Speedup	140.3^{+}	113.1^{+}	122^{+}	101.4^{+}
Spdup 75	173.5	159.3	161.4	127.8

4.2.2. *Medium workload results*

Table VII shows the results for the 1 × 500 jobs workloads. Here, in the ideal case, the values are: Comp = 100, Replan = 0, Util = 25.00, Delay = 3600, Time = 3000, and Speedup = 150. The size of the workloads makes the execution performance increase, and practically matches the ideal speedup for the 75% of the workload, only half for the overall workload.

4.2.3. *Large workload results*

Next, we report on previous results where we used a more aggressive scheduling approach. In this approach, the number of retries was limited to five versus ten job retries. Also, in these measurements some of the sites were not properly configured and jobs failed immediately. These workloads provide insights about GRUBER's scalability. Our results are captured in Table VIII. Round-robin and random-assignment achieve the best performance. The lower completion rates are explained by the low number of retries (five) and the missing BLAST environment configuration at a few sites. All of these factors explain the lower performance achieved in these cases.

DOI: 10.1002/cpe

Table IX. Results of four GRUBER strategies for 4 × 1000 workloads.

	G-RA	G-RR	G-LU	G-LRU
Comp (%)	98.2	98.7	91.7	87.9
Replan	1815	1789	2409	1421
Util (%)	13.51	14.02	11.52	11.05
Delay (s)	66.62	64.41	63.96	68.97
Time (s)	40 356	37 800	48 564	48 636
Speedup	77.3^{+}	74.1^{+}	71^{+}	60^{+}
Spdup75	105.6	102.9	93.8	82.4

The results in Table IX are the means across the four submitters. We see some interesting differences from Table VIII. G-LU's completion rate drops precipitously, presumably for some reason relating to greater contention. The total execution times for G-RA and G-LU increase, although more runs are required to determine the significance of these results.

Furthermore, in the 10 000 workloads, the completion rates drop even more, as the probability of failures increases linearly with the number of jobs (GRUBER maintains a constant load on the available sites). An important observation is that while Speedup does not achieve the 150 value (the maximum possible), it comes very close to this value for the 1000 and 10 000 workloads (140 and 145, respectively).

4.2.4. Failure analysis

We now turn our attention to analyzing the most common errors we have faced in running the BLAST workloads. We split this analysis into two steps. The first step focuses on the analysis of large workloads (presented in Tables III, VIII, and IX). These workloads were run initially and helped in tracking down various Grid3 site errors. The second step focuses on the analysis of the errors for the small and medium workloads (presented in Tables IV–VII) and which were run afterwards. Also, there are two types of failure we consider important to clarify our notation. Job failure is defined as the incapacity to run a job at a certain site, an error that results in job rescheduling (proving also our infrastructure robustness). Workload failure is defined as the incapacity to complete a workload 100% and, usually, this error is caused by the impossibility of completing one or several jobs after a fixed number of retries (five for 1000 and 10 000 workloads, ten for the small and medium workloads).

A failed job is not scheduled a second time to the same site, because Euryale keeps track separately of a bad site for each job. Thus, the same job is not resubmitted to the same site after a failure, but a different job could be. The re-submission operation of Euryale keeps track of site failures only for the current job, because it asks what are the available sites for job X from set S of sites but does not provide specific feedback to GRUBER about the execution result. Thus, the GRUBER engine does not have direct knowledge about site failures, but instead traces differences between how many jobs run versus how many slots the site reports and tries to reconcile these numbers (what we call automated uSLA violation management).

DOI: 10.1002/cpe

Table X. Results of four GRUBER strategies for 1 × 10 000 workloads.

	G-RA	G-RR	G-LU	G-LRU
Comp (%)	91.75	91.88	77.88	73.58
Replan	18 000	23 900	27 718	24 350
Util (%)	24.3	23.3	20.0	17.6
Delay (s)	86.63	85.17	89.01	90.45
Time (s)	226 k	260 k	295 k	349 k
Speedup	137^{+}	145.4^{+}	134^{+}	98.3^{+}
Spdup75	156.2	163	139.6	98.3^{+}

Table XI. Error percentages of 15 000 BLAST jobs submitted as 1000 workloads (percentages are computed as the ratio of current errors to the total number of errors).

Error description	Number of errors	Error percentages (%)
Remote local scheduling timeout (20 min)	10 204	29
Remote application execution failure	6846	19
Remote transient authentication error	361	1
Transient database failure (RLS)	2158	6
Remote unset environment	13 034	38
Remote transient GridFTP failure	1141	3
Remote Globus/GRAM environment error	526	1
Others	8	0

The causes for workload failures are, in most cases, the small number of retries used during these tests (five instead of ten) and in a few cases the DAGMan crashed during workload management. Job failures are instead due to the temporary failure of the RLS server used to stage in and out data (overloading issues), gatekeepers overloading and transient authentication errors, transient RLS error, etc. (Table XI). In Table XI we give the percentages of errors that caused job re-planning for a sample of 15 000 jobs run over Grid3.

Most of these errors were reported and fixes were performed or are still expected to be in the future for the signaled problems. However, we also have to note that Grid3 performance in executing BLAST workloads has already increased between the first set of experiments (large workloads and the results captured in the previous section) and the second set (small and medium workloads). The error results for small and medium workloads are reported in Table XII. As can be observed, most of the errors here are due to various transient errors in the infrastructure (RLS database, GridFTP servers) or due to the more than 20 min queue time.

4.3. Statistical analysis

While previous results provide useful insight about how Grid3 performs in executing workloads when GRUBER is the steering mechanism, further analysis is required to identify how the scheduling

DOI: 10.1002/cpe

Table XII. Error percentages of 28 000 BLAST jobs submitted as smaller workloads (percentages are computed as the ratio of current errors to the total number of errors).

Error description	Number of errors	Error percentages (%)
Remote local scheduling timeout (20 min)	15 810	42
Remote application execution failure	5820	15
Remote transient authentication error	3152	8
Transient database failure (RLS)	5416	14
Remote unset environment	1	0
Remote transient GridFTP failure	5202	14
Remote Globus/GRAM environment error	1769	4
Others	39	0

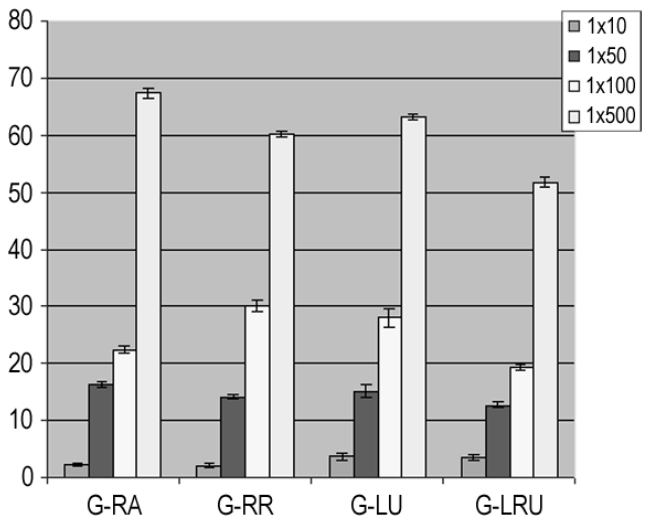


Figure 2. Speedup comparisons among workloads.

strategies have performed comparatively. Figure 2 presents the speedup performance over all runs and the confidence intervals at 90%. Note the small confidence intervals for all runs, which express low standard deviation and the strength of our results across the runs and configurations.

Further, we use the T-test to correlate the results of these experiments. The T-test is usually used for comparing the results of two alternative approaches with the claim that the results are significantly different. For purposes of comparison, we use tournament trees and T-tests as comparison operators. These results are captured in Table XIII. The null hypothesis and alternative hypothesis that we set up to conduct the T-test are as follows.

- H_0 (null hypothesis): any given two runs have comparable performance.
- H_a (alternative hypothesis): prove that H_0 is false; two runs do not have same performance.

DOI: 10.1002/cpe

Table XIII. Tournament tree (T-tests) results.

	G-RA versus G-RR	G-LU versus G-LRU	G-RA versus G-LU
1×10	0.09 (?)	0.17 (?)	0.0005 (T)
1×50	0.0005 (T)	0.0005 (T)	0.0005 (T)
1×100	0.0005 (T)	0.0005 (T)	0.0005 (T)
1×500	0.0005 (T)	0.0005 (T)	0.0005 (T)

The null hypothesis is the one that we want to reject as not being true, while the alternative hypothesis is the one that we want to accept as being true. Our alternative hypothesis is two-sided as we test that the runs are different, which implies either $<$ or $>$; we essentially test $2 \times P(T >$ critical_value) to be less than 0.05. The goal is to obtain a probability that the T value will be greater or less than the critical value. The P value needs to be less than 0.05 for the results to be statistically significant, which implies that H_a is true with 95% confidence for the corresponding comparisons; the lower the P value, the better the confidence. If a P value that is less than 0.05 cannot be found, then the sample space is not statistically significant, and hence more samples must be obtained. The results from Table XIII show that for all workloads other than the smallest, the results are statistically significant with at least a 99.95% confidence. Regarding the smallest workload of 1×10 (for which we had ten sample runs), the number of samples in our experiment do not seem to be enough, and hence more experiments would have to be performed for the 1×10 workload.

5. CONCLUSIONS

Running workloads in Grid environments is often a challenging problem owing to the scale of the environment and to the resource participation based on various sharing strategies. A resource may be taken down during job execution, be improperly set up, or simply fail job execution. Such elements have into be taken into account when targeting a Grid environment.

In this paper we have explored some of the issues that occur on a real Grid, namely Grid3, by means of a specific workload, the BLAST workload, and a specific scheduling framework, GRUBER—an architecture and toolkit for resource uSLA specification and enforcement. During these experiments we faced various problems as described above, and we have quantified what performance a Grid user should expect. In addition, we observed that, for our brokering mechanism for medium workloads, G-RA performs best with a 90% confidence interval, while G-LU performed best for smaller workloads. We also note that G-LRU performed worst for all tested workloads.

'*While Grids have become almost commonplace, the use of good Grid resource management tools is far from ubiquitous because of the many open issues of the field, including the multiple layers of schedulers, the lack of control over resources, the fact that resources are shared, and that users and administrators have conflicting performance goals*' [22]. Thus, one of the future trends in this domain targets tools for workload execution and management [23,24]. The main focus is to have frameworks in which workflows on the Grid can be easily expressed, to support monitoring of the state through

DOI: 10.1002/cpe

visual components, and to have a system that is easy to maintain and deploy. Another important trend is to offer high-level programming models for Grids that shield the programmer from low-level details such as interactions between the application and the underlying middleware system, thus allowing concentration on the application logic only. One such example is the higher-order components (HOCs) paradigm [25], where reusable patterns of parallelism are made available to Grid application programmers as a collection of Web services. A third important trend is the Knowledge Grid [26] that aims to provide a dynamic, self-organizing, and self-managing system to support the development of intelligent computing services. Ongoing work includes the exploration of interconnection semantics and the enriching of e-Science and e-Culture.

ACKNOWLEDGEMENTS

Some of the results 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. *Proceedings of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, Stockholm, Sweden, 2005. IEEE Computer Society Press: Los Alamitos, CA, 2005; 191–200.
2. Dumitrescu C, Foster I. GangSim: A simulator for Grid scheduling studies. *Proceedings of Cluster Computing and the Grid*, Cardiff, U.K., 2005, vol. 2. IEEE Computer Society Press: Los Alamitos, CA, 2005; 1151–1158.
3. Foster I *et al.* The Grid2003 Production Grid: Principles and practice. *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society Press: Los Alamitos, CA, 2004; 236–245.
4. Dan A, Dumitrescu C, Ripeanu M. Connecting client objectives with resource capabilities: An essential component for Grid service management infrastructures. *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04)*, New York, 2004. ACM Press: New York, 2004; 57–64.
5. Altair Grid Technologies. OpenPBS (Portable Batch System), 2004. http://www.openpbs.org/.
6. Maui Team. Maui Scheduler, Center for HPC Cluster Resource Management and Scheduling, 2005. http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php.
7. In J, Avery P. Policy based scheduling for simple quality of service in Grid computing. *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, NM, April 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004.
8. Henry GJ. A fair share scheduler. *AT&T Bell Laboratory Technical Journal* 1984; **3**(8).
9. Dumitrescu C, Foster I. Usage policy-based CPU sharing in virtual organizations. *Proceedings of the 5th International Workshop in Grid Computing*, Pittsburgh, PA, 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004; 53–60.
10. Vöckler J-S, Wilde M, Foster I. The GriPhyN virtual data system. *GriPhyN Technical Report*, The University of Chicago, Chicago, IL, 2002. Available at: http://www.griphyn.org.
11. Platform User's Guide, 2006. http://www.platform.com/Products/Platform.LSF.Family/.
12. Pearlman L, Welch V, Foster I, Kesselman C, Tuecke S. A community authorization service for group collaboration. *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks*, Monterey, CA, 2002. IEEE Computer Society Press: Los Alamitos, CA, 2002; 55–59.
13. Foster I, Kesselman C, Tuecke S. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications* 2001; **15**(3):200–222.
14. Voeckler J, Wilde M. Euryale: Yet another concrete planner (presentation). *Proceedings of the 1st Virtual Data Workshop*, Chicago, IL, 2004. Available at: http://www.griphyn.org.
15. Dumitrescu C, Foster I. GRUBER: A Grid Resource SLA Broker. *Proceedings of the 11th International Euro-Par Conference*, Lisboa, Portugal, 2005 (*Lecture Notes in Computer Science*, vol. 3648). Springer: Berlin, 2005.
16. Legrand IC *et al.* MonALISA: A distributed monitoring service architecture. *Proceedings of the Computing in High Energy Physics*, La Jolla, CA, 2003.
17. Avery P *et al.* An international virtual-data grid laboratory for data intensive science, 2001. http://www.griphyn.org.

DOI: 10.1002/cpe

18. Deelman E, Blythe J, Gil Y, Kesselman C, Mehta G, Patil S, Su M-H, Vahi K, Livny M. Pegasus: Mapping scientific workflows onto the Grid. *Proceedings of the 2nd Across Grids Conference*, Nicosia, Cyprus, 2004 (*Lecture Notes in Computer Science*, vol. 3165). Springer: Berlin, 2004; 11–20.
19. Mohamed HH, Epema DHJ. The design and implementation of the KOALA co-allocating Grid scheduler. *Proceedings of the European Grid Conference*, Amsterdam, 2005 (*Lecture Notes in Computer Science*, vol. 3470). Springer: Berlin, 2005; 640–650.
20. Litzkow M, Livny M, Mutka M. Condor—A hunter of idle workstations. *Proceedings of the 8th International Conference on Distributed Computing Systems*, San Jose, CA, 1998. IEEE Computer Society Press: Los Alamitos, CA, 1998; 104–111.
21. Buyya R. GridBus: A economy-based Grid resource broker, The University of Melbourne, Melbourne, 2004. http://www.gridbus.org.
22. Nabrzyski J, Schopf J, Weglarz J (eds.). *Resource Management for Grid Computing*. Kluwer: Dordrecht, 2003.
23. Foster I, Voeckler J, Wilde M, Zhao Y. Chimera: A virtual data system for representing, querying, and automating data derivation. *Proceedings of the Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop*, May, 1995.
24. Amin K *et al*. Abstracting the Grid. *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE Computer Society Press: Los Alamitos, CA, 2004.
25. Dünnweber J, Gorlatch S. HOC-SA: A Grid service architecture for higher-order components. *Proceedings of the International Conference on Services Computing*, Shanghai, China, September 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004.
26. Zhuge H. The future interconnection environment. *IEEE Computer* 2005; **38**(4):27–33.

DOI: 10.1002/cpe