VIRTUAL DOMAIN SHARING IN E-SCIENCE BASED ON USAGE SERVICE LEVEL AGREEMENTS

Cătălin L. Dumitrescu
CoreGRID Institute on Programming Models
Mathematics and Computer Science Department, The University of Münster, DE
dumitres@uni-muenster.de

Alexandru Iosup, Ozan Sonmez, Hashim Mohamed, and Dick Epema
CoreGRID Institute on Scheduling and Resource Management
Electrical Engineering, Mathematics and Computer Science, Tech. University of Delft, NL
{A.Iosup,O.O.Sonmez,H.H.Mohamed,D.H.J.Epema}@tudelft.nl

Abstract Today's Grids, Peer-to-Peer infrastructures or any large computing collaborations are managed as individual virtual domains (VDs) that focus on their specific problems. However, the research world is starting to shift towards world-wide collaborations and much bigger problems. For this trend to realize, the already existing collection of many resources and services needs to be shared across owning VDs in secure and efficient ways, and at the least administrative costs. In this paper we identify the requirements for and propose a specific solution based on usage service level agreements (uSLAs) for this problem of VD sharing. Further, we propose an integrated architecture that provides uSLA-based access to resources, supports the recurrent delegation of usage rights, and provides fault-tolerant resource co-allocation.

Keywords: Resource Management, Virtual Domains, Usage Service Level Agreements

1. Introduction

E-Science, defined as large-scale, grand-challenge science carried out through distributed global collaborations enabled by the Internet and requiring access to very large scale computing resources [17], is starting to become a common research paradigm [11, 14]. For this vision to materialize, the already existing infrastructures and services need to be shared across existing virtual domains (VDs) in secure and efficient ways, and to be operated at the smallest costs.

Resource owners from multiple VDs, i.e., multi-clusters [6] or computational Grids [7, 10, 15], pool together more and more resources. VD providers may be virtual organizations (VOs [9]- if they also own resources), simple collaborations of companies providing outsourcing services, national Grid infrastructures, groups of scientific laboratories, or universities that provide access to their resources.

In this paper we address the problem of VD sharing. We first describe the characteristics of this problem, and the principles of our usage service level agreement (or uSLA) based approach. Secondly, we analyze the requirements of any Grid scheduling service to provide uSLAs-based support for intra- and inter-VD sharing. Further, we propose an integrated architecture that addresses the identified requirements, starting from an already existing Grid scheduling architecture, KOALA [13]. To make the architecture viable in real environments, we finally propose specific algorithms for resource brokering and scheduling, which we compare by means of simulations in several possible scenarios. We end the paper with our conclusions.

2. Virtual Domain Sharing

In this section we describe our envisaged VD sharing problem, the generic requirements and a uSLA-based solution for it. We start with a real scenario stemming from the european Grid community, which will serve as a guide to the reader's intuition throughout the rest of the paper.

2.1 Motivating Scenario

Consider the following scenario, in which two large-scale computing infrastructures, namely Grid'5000 [2] and the Distributed ASCI Supercomputer (DAS) [6], are combined into a $3,000$ CPU-strong Grid system:
→ **Infrastructure:** DAS is a wide-area computer system in the Netherlands that is used for research on parallel, distributed, and Grid computing. DAS has been built in three successive waves in the past 10 years, resulting in three independent sets of resources: the new DAS-3, the production-level DAS-2, and the somewhat outdated DAS-1. Grid'5000 is the counterpart of DAS in France, and is currently at its first building wave.

→ **Operation:** The resources composing DAS have been provided over time by more than seven different organizations, and are currently clustered into twelve sites. However, as one building wave occurs, the previous infrastructure is declared obsolete, and only the few users with very high computational demands continue accessing it. Within DAS, resources are shared equally among all the participating organizations, except for a few agreements: *Any application cannot run for more than 15 minutes from 08:00 to 20:00*, and *larger projects can reserve at most 50% of the resources in advance, and use them for periods not exceeding two weeks*. Similarly to DAS, Grid'5000 comprises over ten clusters, shared equally amongst more than ten organizations.
→ **VD Sharing:** DAS wants to share its newest component, DAS-3, with Grid'5000. The sharing mechanism will ensure that the Grid'5000 users do not get too many resources. In case of usage imbalances, automated actions specified as penalties, will be enforced until the penalty period expires, or the administrators cancel it. The Grid'5000 is made available to the DAS users, under similar restrictions.

Similar situations occur for other large-scale Grid communities that target collaboration, e.g., the LHC Computing Grid [7] (over 200 sites, over 40,000 CPUs, over 25,000 storage elements with 3 PB storage), NorduGrid [15] (over 20 sites, over 4,000 resources), and OSG/Grid3 [10] (over 3,000 resources). Also, due to the administrative constraints by allowing a resource allocation mechanism (i.e., including co-allocation) to operate unrestricted, the system is exposed to overload from the exterior. A particular problem is that of a large VD's load fraction overloading a small VD. Therefore, this work complements our previous work by introducing a needed mechanism to prevent such events. Without this extension, co-allocation would not be implementable in practice.

2.2 Problem Overview

The VD sharing problem is expressed as follows: *resource providers* (universities, national laboratories or VOs) give *resource consumers* (specific groups of interests, e.g., scientists from different domains) access to *resources of heterogeneous nature* (e.g., processors, disk space, but also software licenses, services, etc.) under specific *agreements*. We categorize the resource providers as domains, VDs and VOs. We further categorize the resource consumers as VOs, groups, and users. A *VD* (e.g., a Grid system or a Peer-to-Peer infrastructure) consists of several domains (e.g., institutions or universities). Each domain clusters resources of a heterogeneous nature; to avoid confusion with the VD, and to punctuate the single physical location of resources, from hereon we will use the term *site* to denote a domain. Within a VD, a multi-level hierarchy of *groups* and *VOs* exists. *Users* are members of a group within a certain VO, and may submit jobs to their own site or others.

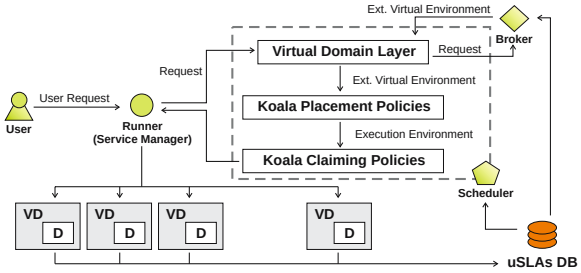

Figure 1. The uSLA-based Scheduling Architecture based on the KOALA Scheduler.

2.3 Requirements for Virtual Domain Sharing

In order to foster collaborations among VDs, specific mechanisms must be designed to allow the provisioning of resources based on pre-negotiated agreements and local preferences. Important challenges for inter-VD resource management can arise in practice from the lack of automated mechanisms for uSLA discovery, publication, from the complexity of the uSLA operations to be performed (to satisfy the transitive resource delegations), or from the sheer number of resource providers and consumers involved. To support controlled VD sharing by means of uSLAs, we identify as mandatory the following key requirements: (a) uSLA support for situations with and without contention and a semantic to ensure that both consumers and providers can establish well defined agreements upon which resources are used; (b) support for uSLA management: storage, location, enforcement, and translation of transitive uSLAs as needed by resource management; (c) support for enhanced scheduling algorithms to take advantage of the uSLAs made available through various means; and (d) tools to help the consumer make an informed resource selection through uSLA-aware brokering algorithms.

3. uSLA-based Scheduling and VD Sharing

In this section we present our uSLA-based architecture for inter-VD sharing. KOALA is implemented as a two layer co-allocation scheduler, and consumer requests are handled by means of a specialized component, the *service manager* or *Runner*, a controller that ensures the completion of the user's request.

3.1 Scheduling Architecture and Algorithms

The enhanced architecture is depicted in Figure 1. A runner sends a request to the KOALA's *engine* to instantiate on the consumer behalf an *execution environment*, in which the user's request can be run unto completion. The KOALA engine calls the *scheduling service*, which creates an *extended virtual*

environment, where the user is allowed to run, according to the associated uSLAs and previous accounting records. The scheduling service can sometimes rely on a specialized uSLA brokering services for building this extended virtual environment, like GRUBER [5]. Then, the scheduling service calls the *brokering service*, and the brokering service verifies the uSLAs, and recommends an *extended virtual environment*. After creating this environment, the scheduling service applies the scheduling policies, filtering the extended virtual environment until the final *execution environment* is identified (see Algorithm 1).

Algorithm 1 uSLA-BRKG-A: Brokering algorithm using decayed usage.

Input:
`Request` ← user request;
`Threshold` ← acceptable penalty threshold;
`Virtual Domains` ← list of virtual domains, their parameters, and uSLAs;
Output:
`Result` ← the list of the top-n resources (initially $\oslash$)
1: **for** each VD_i in `Virtual Domains` **do**
2: sort uSLAs by applicability range, into `uSLAs-Srt`
3: `isVDEligible` ← `TRUE`
4: **for** window w in 1..n **do**
5: compute consumer utilization on VD_i
6: **while** first rule from `uSLAs-Srt` ∩ window **do**
7: pop first rule from `uSLAs-Srt`, as `Rule`
8: **if** `Request` breaks `Rule` **and** `Rule.Penalty` > `Threshold` **then**
9: `isVDEligible` ← `FALSE`
10: **if** `isVDEligible` is `TRUE` **then**
11: `Score` ← `BrokerPolicy` (VD_i , `Request`)
12: `Result` ← `Result` ∪ (VD_i , `Score`)

3.2 The uDecay uSLA

The way udecay is defined is crucial for the behavior of KOALA [12]. In practice, the most encountered decay function is $F_j = k$, where k is a constant factor, e.g., 50%. For a busy system with interactive and batch jobs, a constant factor close to 1 will enforce lower usage shares for heavy users, effectively permitting the interactive job users to work. For a lightly loaded system, setting the decay factor close to 0 will help decrease the decayed utilization rapidly, which allows the users a new complete allocation of resources. Clearly, a constant decay factor cannot accommodate systems with high variations in demand, and various types of users [8].

$$DU = U_0 + U_1 \cdot F_1(S_1) + ... + U_n \cdot \prod_{j=1}^{n} F_j(S_j) \quad (1)$$

Our operator set is a set of per-window decay functions which map from the system state to a decay factor for the given window, $O = \{F_i\}$, with F_i the decay function for the ith window. We assume that system utilizations (S_i for the ith window), and consumer utilizations (U_i for the ith window) are available, as well as the maximum number n of historic usages (windows). With these notations, the decayed usage is given by Equation 1.

4. Validation Approach, Results and Recommendations

Because the integration work for DAS – Grid'5000 environment is still in progress, we perform our analysis by means of simulations using GangSim [4].

4.1 Scenarios and uSLAs

The experimental setup follows a common workload in e-Science settings: the execution of many instances of BLAST, a bio-informatics application. We consider a slighlty larger environment than the one exemplified in Section 2. Three consumers each submit a workload to five VDs. The four uSLAs considered for comparison are [3, 8]:
→ **no-limit uSLA (no-limit)** is a statement that specifies no limit. Resources are acquired on a first come first executed basis [3];
→**commitment-limit uSLA (commitment)**: specifies two upper limits, an epoch limit R_{epoch} and a burst limit R_{burst}, and requires intervals, T_{epoch} and T_{burst}. A job is admitted if (a) the average resource utilization for its VO is less than the corresponding R_{epoch} over T_{epoch}, and (b) there are idle nodes and the average resource utilization for the VO is less than R_{burst} over T_{burst} [3];
→ **time-decay uSLA (decay)** is a statement that specifies a single limit instead and a decay function for each time interval in the past [12].
→ **usage-decay uSLA (udecay)** is the uSLA introduced in Section 3.1.

4.2 Workloads

The employed workloads arrive at the external schedulers under a Poisson distribution; the job lengths are sampled from a Gaussian distribution with an average of 300s; the input files have size between 1kb and 5kb. In each scenario, we use two types of aggregated workloads. The first type is *synchronous*: all consumers submit their jobs in the same time. For the second type, *unsynchronous*, consumers submit their jobs at different time moments. The simulation interval is 1h in all cases, while the scheduling strategies at both the VO and site levels are FCFS. The VO workloads and allocations are:
→ **Balanced Workloads and Equal Allocations Scenario:** workloads are composed of 400, 600 and 500 jobs. The rules under which domains share their resources are 30%, 30%, and 30%, with burst limits of 60%, 60%, and 50% in the commitment uSLA case. The udecay parameters are 1, 0.5, 0.2, 0.1,

and 0.0 for 100%, 50%, 20%, 10%, and 0% utilizations, while the time decay parameters are set to 1, 0.5, 0.2, and 0.1;
→ **Un-Balanced Workloads and Equal Allocations Scenario:** For the second scenario, we use three workloads composed of 160, 800, and 400 jobs; the uSLAs and decay parameters are similar to the previous scenario;
→ **Balanced Workloads and Un-Equal Allocations Scenario:** For the last scenario, workloads and decay parameters are as for the first scenario, with 400, 600, and 500 jobs per workload, but resources are shared under different allocations, namely, 20%, 40%, and 30% and burst allocations of 30%, 60%, and 50% for the commitment uSLA.

4.3 Performance Metrics

The performance metrics considered for analysis are [3–4]:
→ **Aggregated resource utilization** (*Util*): represents the ratio of the CPU time actually consumed by the N jobs executed during the period considered to the total CPU time available.
→ **Total job completion per site, VO or overall** (*Comp*): measures the total number of jobs from a given set that are completed.
→ **Average Grid response time** (*Response*): is computed as the average time per job that elapses from job submission to an external queue until startup;
→ **Average starvation factor** (*Starv*): represents the ratio of the resources requested and available, but not provided to a user, to the resources consumed by the user (ET_i), where i represents a site index. Its equation is:

$$Starv = \sum_{i=1}^{N} min(ST_i, RT_i) / \sum_{i=1}^{N} ET_i \tag{2}$$

→ **uSLA violation ratio** (*Violation*): represents the ratio of CPU consumed by users (BET_i) to the total CPU power. The formula for this quantity is:

$$Violation = \sum_{i=1}^{N} BET_i / (\#\_of\_cpus * \Delta t) \tag{3}$$

4.4 Simulation Results

In this section we present our simulation results for four uSLA, two workload types, and three parameter variations.

Balanced Workloads and Equal Allocations Scenario: Tables 1 and 2 capture the five performance metric values for the four uSLAs. The udecay uSLA offers the best overall performance. It is the second best in terms of total system utilization, but the difference between *no-limit* and *udecay* is minimal. This difference is explained by the balancing introduced by *udecay* compared

uSLA / Metric	*Util (%)*	*Comp (%)*	*Response*	*Starv (%)*	*Violation (%)*
no-limit	90.16	93.57	252.49	13.14	–
commitment	88.22	91.71	233.89	11.83	24.00
decay	79.62	84.28	263.81	15.15	26.63
udecay	89.55	92.07	218.16	10.65	22.60

Table 1. Results for Equal Allocations and Balanced Synchronized Workloads

uSLA / Metric	*Util (%)*	*Comp (%)*	*Response*	*Starv (%)*	*Violation (%)*
no-limit	87.62	90.71	226.91	13.08	–
commitment	83.75	87.78	248.33	12.17	24.27
decay	79.56	82.14	256.94	14.18	25.79
udecay	85.61	88.21	238.94	11.09	22.70

Table 2. Results for Equal Allocations and Balanced Un-Synchronized Workloads

uSLA / Metric	*Util (%)*	*Comp (%)*	*Response*	*Starv (%)*	*Violation (%)*
no-limit	86.26	93.14	225.10	11.03	–
commitment	70.65	76.5	244.54	17.66	27.33
decay	73.06	80.34	246.17	14.34	25.09
udecay	79.64	89.35	226.03	9.66	20.97

Table 3. Results for Equal Allocations and Un-Balanced Synchronized Workloads

to the *no-limit* one. We must also note in the *udecay* case the low value for the *Starv* factor, which indicates that jobs entitled to run acquire fast enough allocated resources.

Un-Balanced Workloads and Equal Allocations Scenario: For this scenario, Tables 3 and 4 capture the five metric values for the four uSLAs. The *udecay* outperforms the *commitment* and *decay* uSLAs in terms of all metrics. It is the second best in terms of the total system utilization, while the difference with the *no-limit* uSLA is again minimal.

Balanced Workloads and Un-Equal Allocations Scenario: Tables 5 and 6 capture the five metric values. The *udecay* does not perform as well as before,

uSLA / Metric	*Util (%)*	*Comp (%)*	*Response*	*Starv (%)*	*Violation (%)*
no-limit	81.90	92.5	196.35	10.83	–
commitment	66.47	82.07	314.08	19.28	28.57
decay	79.90	89.92	215.60	9.44	20.28
udecay	78.95	91.42	226.37	8.74	19.74

Table 4. Results for Equal Allocations and Un-Balanced Un-Synchronized Workloads

uSLA / Metric	*Util (%)*	*Comp (%)*	*Response*	*Starv (%)*	*Violation (%)*
no-limit	90.16	93.57	252.49	13.14	–
commitment	79.58	79.04	298.56	18.50	29.00
decay	72.98	74.16	241.14	16.87	27.96
udecay	84.82	86.21	239.06	10.97	22.39

Table 5. Results for Un-Equal Allocations and Balanced Synchronized Workloads

uSLA / Metric	*Util (%)*	*Comp (%)*	*Response*	*Starv (%)*	*Violation (%)*
no-limit	87.62	90.71	226.91	13.08	–
commitment	74.84	78.28	300.56	19.61	29.91
decay	73.62	74.25	239.18	15.39	11.17
udecay	84.32	89.12	235.49	11.17	22.73

Table 6. Results for Un-Equal Allocations and Balanced Un-Synchronized Workloads

Param. / Metric	*Util (%)*	*Comp (%)*	*Response*	*Starv (%)*	*Violation (%)*
Slow (.8, .5, .2, .1)	82.51	85.78	285.12	5.09	15.79
Medium (.5, .2, .1)	85.61	88.21	238.94	11.09	22.70
Fast (.2,.1)	87.67	89.5	193.99	12.05	23.51

Table 7. Results for Different Decay Parameters

but still offers the best performance in terms of *Starv* and *Violation* metrics. The *udecay* performs better in terms of *Response* metric and the motivation is the un-balance of workloads which makes the historical share to be forgotten before a new wave of jobs start.

The Influence of Decay Parameters: The last set of simulations compares the performance of different decay functions for the udecay uSLA. Our results are captured in Table 7. As can be observed, for the set of high decay values fewer resources are obtained by all consumers, thus the lowest values (first columns). However, the *Starv* and *Violation* metrics are lower due to the free resources always available for the slow starters.

4.5 Lessons and Recommendations

Controlled resource sharing within very large environments is difficult in practice, due to the number and the complexity of participants, their local preferences and software. We believe that uSLA-based resource sharing provides a strong starting point for building environments in which resources are shared under owner preferences. While the uSLAs proposed in this paper are comprehensive, we expect that in the near future new semantics will be required for

other integration efforst. Economy-based sharing models represent an alternative to be considered for augmenting or replacing uSLA-based sharing.

5. Related Work

Current solutions for controlling resource access in large scale distributed systems focus extensively on enabling resource sharing among a virtual environment participants [3, 16].

Scheduling in Parallel for Heterogeneous Independent NetworXs (SPHINX) [16] is our first example of a framework for policy-based scheduling of Grid-enabled only resources. The framework has three main features. First, the scheduling strategy can control the request assignment to Grid resources by adjusting resource usage accounts or request priorities. Second, resource usage management is achieved by assigning usage quotas to intended users. Third, the scheduling method supports reservation based resource allocation and QoS.

Grid Service Broker [1], a part of GridBus project, mediates instead 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, and (e) accessing data from remote source during execution.

The last work we mention here is GRUBER [5], a uSLA-based broker, aimed at addressing the challenging issues that can arise within VDs that integrate participants and resources spanning multiple administrative domains. GRUBER represents the closest work to our proposed architecture.

6. Summary and Conclusions

Our intra- and inter-domain sharing mechanisms are based on uSLAs that permit consumers to use resources up to specified levels, for specified periods of time. Based on resource usage patterns encountered in real large-scale environments, we employ a generic, load-dependent mechanism for accounting resource consumption, i.e., the decay-based mechanism. Our proposed uSLA-based architecture manages the definition, storage, location, and enforcement of uSLAs, and offers support for the recurrent delegation of resource usage rights amongst parties. Being based on a proved Grid scheduling infrastructure, the KOALA Grid scheduler, our architecture provides fault-tolerant resource co-allocation. The architecture includes two uSLA-based components for resource management and for user decision support, which also employ the udecay-based mechanism: a scheduler and a broker.

References

[1] R. Buyya and S. Venugopal. The GridBus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. In *Proceedings of the 1st IEEE International*

Workshop on Grid Economics and Business Models (GECON'04), 2004.

[2] F. Cappello et al. Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *Proceedings of the 6^{th} IEEE/ACM International Workshop on Grid Computing (GRID'05)*, 2005.

[3] C. Dumitrescu and I. Foster. Usage Policy based Scheduling in Virtual Organizations. In *GRID '04: Proceedings of the 5^{th} IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 289–296, Pittsburgh, PA, USA, 2004. IEEE Computer Society.

[4] C. Dumitrescu and I. Foster. GangSim: A Simulator for Grid Scheduling Studies. In *Cluster Computing and Grid (CCGrid'05)*, Cardiff, UK, 2005.

[5] C. Dumitrescu and I. Foster. GRUBER: A Grid Resource Usage SLA BrokER. In *Proc. of 11th International Euro-Par Conference (Euro-Par'05), Portugal*, 2005.

[6] Dutch University Backbone. The distributed ASCI supercomputer (DAS-2), http://www.cs.vu.nl/das2, 2006.

[7] EGEE Team. LCG (URL: http://lcg.web.cern.ch/LCG/), 2004.

[8] D. H. J. Epema. Decay-usage scheduling in multiprocessors. *ACM Transactions on Computing Systems*, 16(4):367–415, 1998.

[9] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150:200–222, 2001.

[10] I. Foster et al. The Grid2003 Production Grid: Principles and Practice. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC-13 '04)*, Hawai, 2004.

[11] A. J. G. Hey and G. Fox. Special Issue: Grids and Web Services for e-Science. *Concurrency - Practice and Experience*, 17(2-4):317–322, 2005.

[12] MAUI Scheduler, http://www.clusterresources.com/pages/products, Last accessed: 2006.

[13] H. Mohamed and D. Epema. The Design and Implementation of the KOALA Co-Allocating Grid Scheduler. In *Proceedings of the European Grid Conference, Amsterdam*, volume 3470 of *LNCS*, pages 640–650, 2005.

[14] H. Newman, M. H. Ellisman, and J. A. Orcutt. Data-intensive e-Science Frontier Research. *Commun. ACM*, 46(11):68–77, 2003.

[15] NorduGrid Collaboration. Solution for Wide Area Computing and Data Handling, 2006.

[16] J. uk In, P. Avery, R. Cavanaugh, L. Chitnis, M. Kulkarni, and S. Ranka. SPHINX: A Fault-Tolerant System for Scheduling in Dynamic Grid Environments. *International Parallel and Distributed Processing Symposium (IPDPS)*, 01:12b, 2005.

[17] United Kingdom Research Councils. (URL: http://www.rcuk.ac.uk/escience/), 2007.