Division of Labor: Tools for Growth and Scalability of Grids

K. Keahey[1,2], I. Foster[1,2], T. Freeman[1], A. Rana[3], B. Sotomayor[1], F. Würthwein[3]
[1]*The University of Chicago, Chicago, IL*
[2]*Argonne National Laboratory, Argonne, IL*
[3]*University of California, San Diego, CA*
{foster,freeman,keahey,borja}@mcs.anl.gov
{fkw,rana}@ucsd.edu

Abstract

To enable Grid scalability and growth, a usage model has evolved where resource providers make resources available not to individual users directly, but rather to larger units, called virtual organizations (VOs). This enables the resource provider to focus on the dynamics of providing resources to the VOs while VOs specialize to provide resources to their users. Achieving such division of labor requires tools and mechanisms that would allow a resource provider to reliably delegate the usage of a specific resource quantum in such a way that it is unimpacted by other activities that the resource provider participates in. In this paper, we argue that the virtual workspace abstraction provides mechanisms needed to create and manage such environments. Next, we present extensions to the Workspace Service based on the Globus Toolkit 4, and describe an implementation of workspace enforcement using the Xen virtual machine and Linux networking tools. Finally, we use this implementation to demonstrate how workspaces can be used by the resource provider to allocate resources to VO-specific infrastructure services called Edge Services.

1. Introduction

Over the last decade of successful Grid usage a model has evolved where a number of resources federated under a large *resource provider* such as Grid3 [1], Open Science Grid (OSG) [2], or TeraGrid [3] make resources available not to individual users directly but rather to larger units, called *virtual organizations* (VOs) [4]. The VO then enables its users to use the resources according to VO-specific policies. This interaction model allows Grids to scale – a fundamental condition of growth -- since instead of directly providing for the needs of each of many thousands of users, a resource provider interacts with only tens of VOs. To function correctly, this model requires the development of tools that will ensure that resources for each VO are provisioned in a controlled manner and used fairly, that the work of different VOs does not impact each other, and that each VO's usage is properly accounted for. Furthermore, it is frequently necessary for the isolation of a VO from the resource provider to extend to the software stack: a VO should be able to carry out its work regardless of the software supported by the resource provider (and vice versa) or by other VOs. In short, the growth and scalability of Grids requires the development of mechanisms that would allow for a clear separation of concerns between resource providers and virtual organizations, in other words enable the *division of labor* [5].

We argue that in order to provide an effective solution to the management issues that arise from this "division of labor", we need to develop abstractions and tools that allow VOs to dynamically configure, deploy, and manage the required environments, as well as negotiate enforceable resource allocations for their execution. We discuss the requirements and properties of such tools in the context of Edge Services – VO-specific infrastructure services particularly sensitive to issues of resource sharing – to illustrate the different aspects where a separation of concerns would be useful.

In [6] we defined an abstraction that meets many of our requirements: *virtual workspaces.* Workspaces can be implemented by a variety of mechanisms, including configuration of dedicated physical resources as well as the use of virtual machines. In this paper, we refine these abstractions to address the resource allocation and fair-sharing issues as well as security processing required to provide a tool satisfying the separation of concerns and fair sharing requirements.

Furthermore, we demonstrate how the virtual machine implementation of workspaces can provide a useful solution to our requirements. We do this by experimenting with their use in the *Edge Service Framework* (ESF) [7], illustrating how it deals with the challenging scenarios that occur in the context of running Edge Services on a production testbed.

In summary, our contributions are as follows:

1) We extend the workspace service abstraction to provide mechanisms for dynamically negotiated resource usage.
2) We present an implementation approach using Globus Toolkit 4 [8], the Workspace Service based on the Xen virtual machine [9] and Linux networking tools.
3) We describe the application of this architecture and implementation in the use case of Edge Services Framework.
4) We present an experimental evaluation of the methods we developed and show how they can be used to solve problems in the current Edge Service deployments.

2. Related Work

The need for developing management tools separating the resource provider's enforcement from a VO's enforcement has been described in [10]. Here, we argue for tools to manage a resource slot rather than just computation, and develop methods to achieve such separation in a particularly demanding case – the Edge Services – which require addressing more than one resource allocation aspects in conjunction. Further, the management methods we propose are finer-grained and open to negotiation by the client.

The concept of a resource allocation is similar to the PlanetLab abstraction of a resource *slice* [11]. We propose here methods for negotiating such slices dynamically and evaluate a specific implementation of the concept in the Grid context.

Many projects have used various implementations of virtual machines (VMs) in Grid computing to leverage their isolation properties [12-14]. Our approach here is focused specifically on negotiating and implementing fair share management between VMs constraining resource usage for infrastructure services administered by different VOs.

The Virtuoso Project also uses VMs for resource management [15], however the interface they propose focuses primarily on the CPU and, using a different implementation, have developed different methods. Work described in [16] focuses on evaluation of enforcement capabilities of different VMs; our work is different in that we develop methods using those capabilities to enforce negotiated resource allocations.

Ideas of combining virtual machines and distributed computing have also been proposed in the context of the Xenoserver Project [17] but on a more coarse-grained level.

Finally, the interfaces we propose are informed by the standards work at the Global Grid Forum, specifically the work on WS-Agreement [18] and the Job Submission Definition Language (JSDL) [19]; our effort is less broad in scope, focused on test-driving practical applications and the implementation implications of Grid abstractions.

3. Requirements and Focus

Our argument for enabling "division of labor" between resource providers and VOs is driven by the need to provide mechanisms for flexible and scalable behavior in the Grid and thus provide a basis for its growth. It is impossible for a resource provider to provide every single bit of configuration for a VO, much less to arbitrate between different VOs and their users; they need to be able to focus on providing resources and keeping them running. In general, we want to enable a model where a *provider* (e.g., a resource owner) can delegate the usage of a well-constrained resource quantum to a *consumer* (e.g., a VO) such that this consumer can turn around and further distribute those resources among its customers (e.g., VO users). As we explained in [6] this situation can in general involve many different layers and employ different workspace implementations to achieve the desired fairness and granularity of sharing.

A compelling illustration of the issues that arise where division of labor between resource providers and VOs is not recognized is provided by *Edge Services*. Edge Services are Grid middleware services enabling access to site resources (the name derives from the fact that they typically configured to execute on the edge of private/site and public network). Examples include infrastructure built around job management services (such as GRAM or Condor), storage brokers (such as SRM), and caching databases. In addition to its primary function, the implementation of an Edge Service often also includes multiple privileged and unprivileged actions such as data staging and registration, security processing, monitoring, resource procurement, and others. Edge Services are thus complex, and exercise many aspects of sharing between a VO and a resource provider as well as between multiple VOs sharing the same resource provider.

Edge Services are often required to be VO-specific: their configuration is determined by a VO to reflect the needs of its users. Different VOs upgrade these services on a different schedule and may use conflicting versions of such services. Further, each VO works with an often large and dynamically changing pool of users and has to mold its policies not only in response to its fluid membership, but also to potentially changing objectives (e.g. research vs. development). In addition, since all requests for site use come through Edge Services, they easily become a bottleneck as request rates increase. Because of their variety and complexity (combination of differently owned processes and threads, network and disk traffic, and memory demands) it is hard for a resource provider to track, account, and enforce resource usage and thus ensure quality of service for any particular VO. This leads to situations where some users cannot use a site at all due to excessive traffic from others. Last but not least, the relationship between an organization and resource provider evolves constantly reflecting the need for potentially frequent and dynamic change in the configuration and policy assigned to Edge Services.

Without a mechanism enabling a resource provider to effectively delegate bulk resource usage to a VO, the provider takes on too large a burden affecting its ability to scale – and to prevent any one VO from impacting another. Based on the previous discussion, in a general case, such mechanism should provide separation between the VO and the resource provider along the following dimensions:

1) Environment and configuration: a VO should be able to provide the configuration it needs independently of the resource provider.
2) Isolation: the provider needs to be able to delegate resource usage to a VO in such a way that the VO's activities cannot impact the resource provider -- and therefore don't need to be under its control.
3) Resource usage enforcement and accounting: a provider needs to be able to grant, enforce, and account for VO resource usage in a way that is independent of how the resource is consumed.

Addressing concerns (1) and (2) is the subject of our future and ongoing research [6, 20]. In this paper we want to focus on the third issue and propose extensions to the virtual workspace abstraction that provide mechanisms for the negotiation and enforcement of resource usage.

4. Allocating Resources to Workspaces

Virtual Workspaces [20] allow an authorized Grid client to dynamically deploy a customized and isolated Grid execution environment. The environment is deployed based on workspace meta-data, provided by the client, which contains all the information necessary for deployment (i.e., in the VM case, VM image and configuration information). In addition, the client provides a resource allocation request that describes resources bound to the workspace at deployment time. In addition to deployment capabilities, the workspace service provides other management interfaces based on the Web Services Resource Framework (WSRF) [21] such as inspection and lifetime management. Workspaces can be implemented through various means, including using imaging software on physical resources (similar as in [22]) as well as virtual machines.

In this section, we explain how the workspace service can be used to delegate resources to activities contained in the workspace. Specifically, we define a resource allocation element and discuss its implementation. We emphasize that our current interface enables the client to only negotiate resource allocation policy for a specified workspace and describe methods allowing a resource provider to apply such policy. The policy is applied at the resource provider's discretion. We do not at present implement methods that would enable the client to either enter into incentive-based agreements [18] or implement the monitoring systems and other infrastructure services that such architectures imply.

4.1. Negotiating Resource Allocation Policy

The Workspace Service interface is based on WSRF and thus includes operations supporting the creation, monitoring and lifetime management methods. Building on this model we added methods allowing a VO to dynamically determine, inspect, as well as renegotiate, resource assignment policy relevant to a specific workspace.

Shaping resource assignment policy for a specific workspace has four stages: (1) a client defines a requested resource allocation, (2) the resource allocation is negotiated, resulting in an assigned resource allocation, (3) the assigned resource allocation is published, and (4) the resource allocation is potentially renegotiated.

Our current implementation uses a simple all-or-nothing negotiation strategy; requested resource allocation is sent as part of the Workspace Service's

create operation and is either accepted or rejected based on resource availability. If accepted, the assigned resource allocation (which concretizes the requested resource allocation values as -- see below) is published as a WSRF resource property. Renegotiation is achieved by updating the resource property values. This can be done either by sending a complete new resource property description or by requesting the adjustment of a specific value (e.g., CPU percentage) – in the latter case, the request is interpreted as if the existing resource allocation with adjusted CPU value was sent. As with workspace creation, the result of this operation is subject to the same all-or-nothing strategy. If the request cannot be satisfied the workspace deployment is not disrupted – if it can, new resources are assigned to the workspace.

The assigned resource allocation is depicted in Figure 1. Time is specified as start time (only current time is accepted at present) and duration of deployment. Memory size is also specified as single value. The CPU is specified as a list of architecture/percentage pairs to accommodate workspaces with multiple CPUs. Similarly disk and networking are also specified as potentially multiple resource slots with salient characteristics; for example, in the Edge Services example the two networking slots are used for private and public connection respectively. The values of some of the qualities in the picture, including duration, CPU percentage, size, read/write speed and bandwidth can be specified with an "at least" option which is interpreted to mean "the assigned value or more" (given our assumptions, we did not find the "at most" option to be practical).

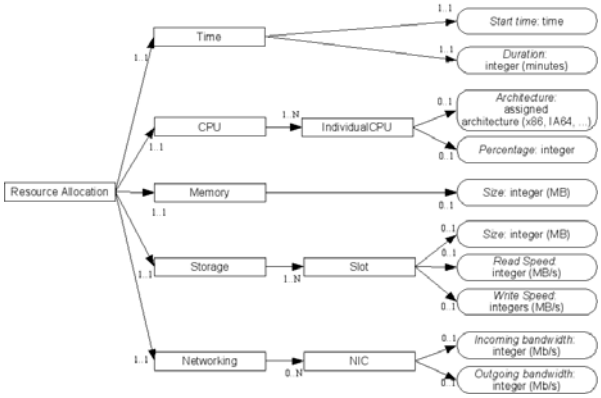

Figure 1: Assigned Resource Allocation

The requested resource allocation differs from the assigned resource allocation in two major respects. First, the disk, CPU and networking elements allow for the description of choices (e.g., a list of acceptable CPU architectures to choose from) and second, the requested values are specified in terms of ranges (e.g., 50-60% of CPU).

A client may be interested in only specific aspects of resource allocation; for example, it may specify only memory and CPU. In such cases, default values will be assigned by the workspace service. We extended the Workspace service Factory resource properties to publish the default policy on such cases; while the current implementation provides only the "best effort" policy we are experimenting with more controllable defaults, e.g. preventing service starvation through overbooking of memory or other qualities.

4.2. Enforcing the Resource Allocation

In its VM-based implementation, the Workspace Service interacts with the Xen hypervisor [9] to provide secure VM deployment and management for Grid clients. Hypervisor interaction takes place in Xen "domain 0" which, in addition to being a standard Xen virtual machine, allows a client to create and manage other virtual machines (called "user domains" in Xen). In this section, we describe how we use a variety of tools, including some Xen-supported features, in order to implement fine-grain resource usage enforcement.

To enforce the CPU allocation we used the Xen Simple-Earliest Deadline First (SEDF) [23] scheduler which provides weighted (i.e., percentage based) CPU sharing between different domains over a set period of time. If an attempt is made to claim more CPU than is available such request will be declined by the scheduler. An extra_time flag tells the scheduler it can give a domain extra CPU cycles during the scheduling period if they are available (i.e., not used by another domain); if this flag is set to false the domains get only their assigned CPU shares whether additional resources are available or not. Both this policy and the assigned CPU share can be changed on the fly; different domains can take advantage of the extra_time scheduling policy independently of each other.

All domains are treated equally – there are no special priorities assigned to any domains. This raises the question of how many resources need to be assigned to domain 0 in order to ensure timely processing of I/O; more domain 0 CPU% does not necessarily mean faster throughput. In our implementation, we found that an assignment of 10% plus extra time for domain 0 gave a good balance for the workloads in our set of use cases (see [24] for a discussion of assignment trade-offs).

The physical memory size allocated to a Xen domain is specified when the domain is created. It is a hard limit: if the requested memory size goes beyond

the available physical memory size the startup operation will fail and the hypervisor will report a memory allocation error. A domain can adjust its memory size after the startup using the balloon driver included in Xen. The balloon driver is a mechanism in a guest OS that can allocate "fake" memory in an attempt to flush invalid or cached memory pages out of physical memory that the hypervisor can then give to other VMs.

There are two flavors of disk allocation that are needed: obtaining storage for disk partitions that form a part of a VM image, and providing extra writing space for the VM. The former is addressed by mounting VM partition as a loopback device (a physical partition on the local disk could also be mounted but those are already allocated), the latter by allocating space from network filesystems or by creating new, blank loopback images. For best access and write times, both read and write partitions should be mounted from whichever site disk can offer the best performance within the requested allocation. In order to accomplish this, the workspace service keeps track of available local disk space on various resources and uses the "size" element in partition meta-data to schedule workspace deployment. Further, since VMs can share readonly partitions, this aspect could be taken care of in scheduling. Disk allocations cannot be managed on the fly.

Xen by itself does not implement controlled bandwidth sharing so we rely on Linux network shaping tools [25] in order to implement it. We take advantage of the fact that the network interface of each domain is connected to a virtual network interface in domain 0 by a point to point link and the traffic on these virtual interfaces is handled in domain 0 using standard Linux mechanisms for bridging, routing and rate limiting. To implement bandwidth sharing (for both incoming and outgoing bandwidth) we limit the rate of network traffic going to and from the respective domains (the to and from bandwidth can be different) using the Hierarchical Token Bucket queuing discipline [26]. To achieve this we needed to recompile the domain 0 kernel, and developed an API to the Linux tools that allows us to set the bandwidth rates for created domains. Bandwidth shaping is currently the only

5. Case Study: Edge Services Framework

The Edge Services Framework (ESF) is being developed for the OSG in order to decouple the process of configuring and managing service nodes for VOs from providing resources, thus allowing for division of labor between a VO administrator and site administrator. To do this, ESF leverages the abstraction of workspaces to allow VO administrator to configure a deployment-ready Edge Service and deploy them based on need and resource availability.

ESF consists of a workspace image library, image transport and storage mechanisms, and the workspace service. The image library contains base images (intended to provide base configurations) and wafer images (with fully configured Edge Services). The base images represent a basic OS configuration and include at present Scientific Linux 3/4, CentOS 3/4, and Fedora Core 4. The Wafer images currently include the ATLAS DASH service [27] and CMS FroNtier [28]. Since wafers can be large in size (5 to ~10 GB) ESF uses compression and fast transport mechanisms (GridFTP [29]) as well as high-end Storage Elements (SEs) such as dCache [30].

The role of a VO administrator is to prepare, configure, and test an ES wafer. The image can then be shared within the VO, transported to deployment sites, and stored within the local site SE where it can be retrieved for deployment by any of the VO administrators. In the current deployment, images stored on a site are further configured with required IP addresses, and a pre-generated credential; we are working towards automating this process as part of workspace deployment [20].

The role of a site administrator is to provision hardware resources that can be used for Edge Services, ensure their proper configuration, and maintain them. In our current deployment this includes configuring them with Xen, and providing one deployment of GT4 and the Workspace Service per site. A Site administrator also provisions storage space in a local Storage Element for storage and retrieval of ES wafer images.

During site operation, Edge Service workspaces are dynamically retrieved, provisioned, and deployed by a VO administrator authorized using his or her VOMS credentials [31]. For example, when working with ATLAS analysis jobs requiring a database cache of a specific type, an ATLAS administrator deploys the DASH Edge Service. On deployment, the cache initializes using remote data repositories over its public network connection and is then available on the private network to the jobs submitted by ATLAS users to the site.

Current ESF deployment spans both integration-level testbed sites and production-level sites on OSG. The integration-level sites include ANL, FNAL, University of Chicago and UCSD. The production-level deployment is at the DISUN [32] at SDSC.

Our ESF deployment experiences to date are encouraging. Advantages for VOs include portability of Edge Services distributions, guaranteed use of dedicated resources at sites based on timed leases, ease of hosting various OS solutions to cater to specific sub-community needs, and an increased control to customize services configuration. Advantages for sites manifest themselves as flexibility in hardware provisioning (resources can be freed on expiration of Edge Services leases or reallocated dynamically), freedom from deployment of diverse sets of services resulting in an ability to support more VOs with less effort, and a relief from providing direct configuration support to VO-specific services that can be more efficiently handled by VOs themselves.

A potential disadvantage for sites is decreased control over services and interfaces exposed on the local infrastructure and consequent concern about their soundness and security. In our current infrastructure this is addressed by vetting workspace images by site personnel. We are working on reducing the impact of such procedures by fragmentation as well as on managing trust using digitally signed images [20].

6. Experimental Evaluation

We evaluated our abstraction and implementation in the context of an Edge Service known as *Compute Element (CE)* configured to enable job submission to a site. A common problem with CEs is that it is very hard to guarantee the quality of service they provide: submissions from all VOs suffer equally at times when a CE is subjected to heavy load independently of whether they significantly contribute to this load or not.

In order to evaluate the validity of our approach we recreated this situation on a service node recreated on an AMD Athlon(tm) MP 2200+ machine (dual processor, configured to work with one) with 2GB memory, configured with Linux 2.6.12. For the VM-based experiments we used Xen 3.0, with domain 0 always given 10% of CPU and 256 MB of memory. For this experiment, our simplified implementation of an OSG CE was configured with GT4 GRAM.

We considered two different scenarios in which two VOs: VO1 and VO2, share a service node. In the first scenario (physical machine scenario) the CE is deployed directly on the physical machine and no mechanisms controlling sharing between VOs are in place (reflecting the situation in most current deployments). In the second scenario (workspace scenario) each VO deploys the CE in a workspace, implemented as a Xen VM, and negotiates a resource allocation for this workspace. In our experiments, each VO requests at least 45% of the CPU (45% CPU with extra time available if unused by other activity on the node) and 896 MB of memory respectively.

To simulate heavy request load coming from VO1 we used a load client that submitted a request to the CE every 10 seconds (to further simulate the computational load for processing a complex request, the submitted job performed 2 million square root operations). In both the physical and workspace scenario we measured the end-to-end job throughput of two clients, VO1 client and VO2 client, submitting a non-staging GRAM job (/bin/date). Each job submission consisted of create+subscribe, state notifications, and a destroy exchange. The job submission throughput was calculated over a period starting after the CE was saturated with the load (starting 150 seconds into the experiment) to a period when VO2 client stopped submitting (450 seconds into the experiment). The results shown below are an average of 5 trials (all values were roughly within 10% of the average):

PHYSICAL SCENARIO:
VO1client: **7.82** jobs/minute
VO2client: **8.00** jobs/minute

WORKSPACE SCENARIO:
VO1client: **4.18** jobs/minute
VO2client: **22.36** jobs/minute

We observe that in the physical scenario each client has roughly the same (low) request throughput: both VOs are equally impacted by the pre-existing request load coming from VO1. In the workspace scenario on the other hand, all of the VO1 request load is confined to VM1. This results in a significantly worse throughput for VM1 which is roughly halved as all the load generated by VO1 is now allocated half the resources. The request throughput for VO2 client on the other hand shot up substantially since none of the VO1 traffic is now directed to VM2 resources.

To closer observe the behavior of the CEs overtime we summed the number of completed request for VO1 and VO2 clients respectively during regular intervals (every 30 seconds) and plotted this number against time. The graph in Figure 2 shows the comparison: VO2 client has consistently high throughput. VO1 client has low throughput up until the point when VO2 client stops sending requests and causing VM2 to temporarily stop consuming resources. VO1 is then able to take advantage of the “extra_time” policy (specified by requesting “at least” 45% of CPU) and obtain more processing power. At about 600 seconds

the load client ceases to submit resulting in dramatically better throuput rate for VO1.

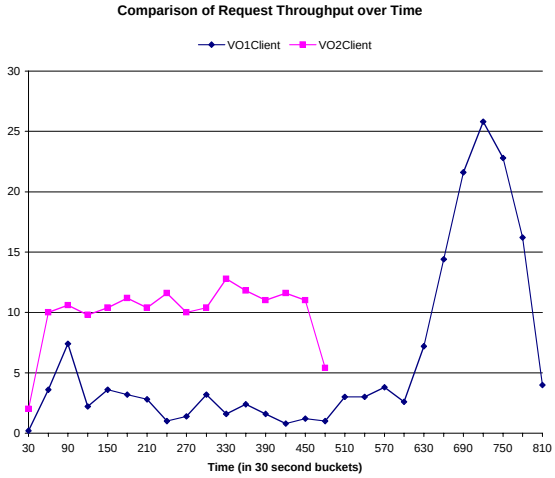

Figure 2: Request throughput for VO1 and VO2 clients; completed requests are totalled every 30 seconds

Varying the load coming from the load client (by doubling or tripling its operations) shows the same pattern for VO2 request throughput: the job throughput stays at the same level independently of the load which shows that VM2 is unimpacted by the varying load conditions in VM1 (Figure 3).

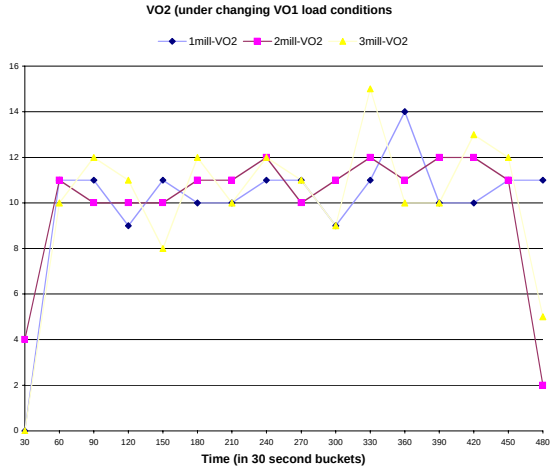

Figure 3: VO2 request throughput under increasing load conditions from VO1

The job throughput numbers suggest that due to the heavy load experienced by VO1 a resource allocation favoring VO1 might be more appropriate as long as that outcome would also be acceptable to VO2. We repeated the experiment with a 60% CPU assignment to VO1 and a 30% assignment to VO2 and saw a significant improvement in request throughput for VO1 accompanied by a drop in VO2 throughput (Figure 4).

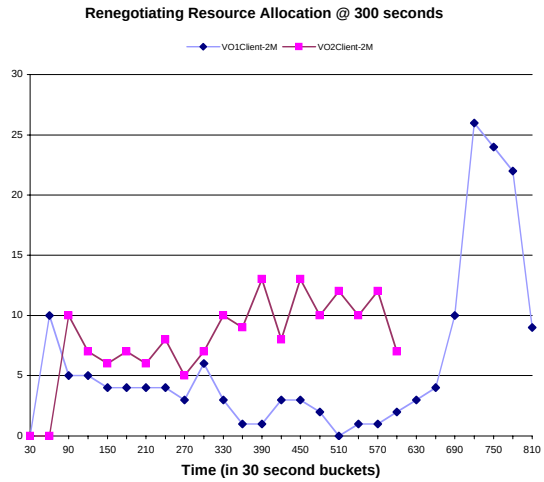

Figure 4: Request througput for VO1 and VO2: VO1 and VO2 start with 60% and 30% of CPU allocation respectively; at 300% the VO's 45%/45% share is renegotiated.

However, if VO2 is committed to providing a specific job request rate to its users this assignment may not be acceptable. In this case, VO2 may use the workspace service interact to renegotiate the allocation. Figure 4 shows this renegotiation happening at 300 seconds. As a result, VO1 and VO2 throughput quickly goes back to previous levels.

Our results demonstrate that using workspaces gives the resource provider a helpful tool allowing him or her to isolate and enforce resource usage between different communities independently of what methods are used by a community to consume resources. It also isolates the resource provider from negative effects of dealing with communities that are not able or willing to properly regulate their traffic – if one community should create unreasonable load, others are protected and can still support the negotiated quality of service. The benefits are similar from the perspective of communities, with the important addition that in case of service degradation a community has the option of negotiating a better resource assignment in discrete terms. Finally, these benefits come at a reasonable price; as demonstrated in [9] (and confirmed by our experiences on this project) using Xen results in only minimal performance degradation.

In practice, the principal cost of using workspaces comes from their memory demands. Each VM may have significant memory requirements (an typical ESF

VM is configured with about 1GB of memory) which certainly limit their usage on a per-user basis and sometimes even per-community basis. Also, much of the separation between VOs, both in our experiments and deployment was achieved effectively by replicating services (i.e., effectively using twice as many resources). While these concerns determine the level of granularity at which workspaces may be used in their current implementation, it is also probable that they may be alleviated in the future by copy on write memory sharing techniques used in other virtualization platforms. Finally, while we believe that the abstraction of a workspace is needed to provide division of labor in the general case, we also acknowledge that special cases exist where less formal enforcement methods and more casual sharing relationships are sufficient and less expensive.

7. Conclusions and Future Work

Our experimental as well as deployment experiences lead us to conclude that the workspaces constitute a promising "division of labor" tool for providers and consumers. This is especially true in situations, such as load management on OSG service nodes, where understanding the interdependency of various factors causing load is complex. The ability to negotiate and renegotiate resource allocations is particularly important, both to the client and the provider: it allows them to react to changing load conditions and optimize their provisioning to satisfy the targets.

Our future work in this area focuses on refining our workspace management methods and fine-tuning the enforcement implementation to better capture the interrelated resource management aspects. Once understood in the atomic case, we plan to extend them to use with aggregate workspaces. Also, while we believe that using workspaces will help Grid computing scale, it is likely that incentive-based models may be needed to further fuel its growth. For that, we will also need the invisible hand.

9. Acknowledgments

This work was supported in part by the NSF CSR program and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy under Contract W-31-109-ENG-38.

10. References

1. Foster, I. and others. The Grid2003 Production Grid: Principles and Practice. in IEEE International Symposium on High Performance Distributed Computing. 2004: IEEE Computer Science Press.
2. Open Science Grid (OSG). 2004: www.opensciencegrid.org.
3. The TeraGrid Project.
4. Foster, I., C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 2001. 15(3): p. 200-222.
5. Smith, A., The Wealth of Nations.
6. Keahey, K., I. Foster, T. Freeman, and X. Zhang, Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid. accepted for publication in the Scientific Progamming Journal, 2005.
7. Edge Services Framework (ESF): http://osg.ivdgl.org/twiki/bin/view/EdgeServices/WebHome.
8. Foster, I., Globus Toolkit version 4: Software for Service-Oriented Systems. IFIP International Conference on Network and Parallel Computing, 2005.
9. Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. in ACM Symposium on Operating Systems Principles (SOSP).
10. Foster, I., K. Keahey, C. Kesselman, E. Laure, M. Livny, S. Martin, M. Rynge, and G. Singh, Embedding Community-Specific Resource Managers in General-Purpose Grid Infrastructure. White Paper, 2005.
11. Bavier, A., M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Services. in 1st Symposium on Network Systems Design and Implementation. 2004.
12. Figueiredo, R., P. Dinda, and J. Fortes. A Case for Grid Computing on Virtual Machines. in 23rd International Conference on Distributed Computing Systems. 2003.
13. Adabala, S., V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, From Virtualized Resources to Virtual Computing Grids: The In-VIGO System. Future Generation Computer Systems, 2004.
14. Xu, M., Z. Hu, W. Long, and W. Liu, Service Virtualization: Infrastructure and Applications, in The

Grid: Blueprint for a New Computing Infrastructure. 2004, Morgan Kaufmann.
15. Lin, B. and P. Dinda, VSched: Mixing Batch And Interactive Machines Using Periodic Real-time Scheduling. Proceeings of the ACM/IEEE conference on Supercomputing, 2005.
16. Quetier, B., V. Neri, and F. Cappello, Scalability Comparison of 4 Host Virtualization Tools. White Paper, 2005.
17. Reed, D., I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: Accountable Execution of Untrusted Programs. in 7th Workshop on Hot Topics in Operating Systems. 1999. Rio Rico, AZ: IEEE Computer Society Press.
18. Andrieux, A., K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, Web Services Agreement Specification (WS-Agreement) Draft 20. 2004: https://forge.gridforum.org/projects/graap-wg/.
19. Andrieux, A., K. Czajkowski, J. Lam, C. Smith, and M. Xu, Standard Terms for Specifying Computational Jobs. http://www.epcc.ed.ac.uk/%7Eali/WORK/GGF/JSDL-WG/DOCS/WS-Agreement_job_terms_for_JSDL_print.pdf, 2003.
20. Lu, W., K. Keahey, T. Freeman, and F. Siebenlist, Making your workspace secure: establishing trust with VMs in the Grid. SC05 Poster Presentation, 2005.
21. Czajkowski, K., D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, The WS-Resource Framework. 2004: www.globus.org/wsrf.
22. Chase, J., L. Grit, D. Irwin, J. Moore, and S. Sprenkle, Dynamic Virtual Clusters in a Grid Site Manager. accepted to the 12th International Symposium on High Performance Distributed Computing (HPDC-12), 2003.
23. Xen Scheduling: http://wiki.xensource.com/xenwiki/Scheduling.
24. Gupta, D., R. Gardner, and L. Cherkasova, XenMon: QoS Monitoring and Performance Profiling Tool. Tech Report: HPL-2005-187, 2005.
25. Linux Advanced Routing and Traffic Control: http://lartc.org.
26. Devera, M., Hierarchical Token Bucket Queuing. 2005: http://luxik.cdi.cz/~devik/qos/htb/.
27. Vaniachine, A., DASH: Database Access for Secure Hyperinfrastructure: OSG document 307. http://osg-docdb.opensciencegrid.org/cgi-bin/ShowDocument?docid=307.
28. Lueking, L., FroNtier project: http://lynx.fnal.gov/ntier-wiki.
29. Allcock, W., J. Bester, J. Bresnahan, A.L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. in Mass Storage Conference. 2001.
30. The dCache Project: http://www.dcache.org.
31. The Virtual Organization Management System: http://infnforge.cnaf.infn.it/projects/voms.
32. Data Intensive Sciences University Network: http://disun.org.