Division of Labor: Tools for Growing and Scaling Grids

T. Freeman[2], K. Keahey[2,3], I. Foster[1,2,3],
A. Rana[4], B. Sotomoayor[1], and F. Wuerthwein[4]

[1] Department of Computer Science, University of Chicago, Chicago, IL, USA
[2] Computation Institute, University of Chicago & Argonne National Lab, Chicago, IL, USA
[3] Math & Computer Science Division, Argonne National Lab, Argonne, IL, USA
[4] Department of Physics, University of California, San Diego, CA, USA
{foster, freeman, keahey, borja}@mcs.anl.gov,
{rana, fkw}@ucsd.edu

Abstract. To enable Grid scalability and growth, a usage model has evolved whereby resource providers make resources available not to individual users directly, but rather to larger units, called virtual organizations. In this paper, we describe abstractions that allow resource providers to delegate the usage of remote resources dynamically to virtual organizations in application-independent ways, and present and evaluate an implementation of this abstraction using the Xen virtual machine and Linux networking tools. We also describe how our implementation is being used in a specific context, namely the enforcement of resource allocations in the Edge Services Framework, currently deployed in the Open Science Grid.

Keywords: virtualization, grid computing, resource management, distributed computing.

1 Introduction

Over the last decade of successful Grid usage, a model has evolved whereby a number of resources federated by a large *resource provider* such as Grid3 [1] (and its successor Open Science Grid (OSG) [2]) and TeraGrid [3] make resources available not to individual users directly but rather to larger communities, called *virtual organizations* (VOs) [4]. Each 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 providing for the needs of each of many thousands of users directly, a resource provider need interact only with a smaller number of VOs.

To function correctly, this VO-based scheduling model requires the development of tools that can ensure that the resources provided to each client (i.e., VO) are delivered in a controlled manner, so that clients obtain the resources they need, and one client cannot interfere with others, for example by acquiring excess resources or damaging data. In addition, it is frequently important that individual clients be able to deploy specialized software not supported by the resource provider. In short, the growth and scalability of Grids requires the development of mechanisms that allow

A. Dan and W. Lamersdorf (Eds.): ICSOC 2006, LNCS 4294, pp. 40 – 51, 2006.

for a clear separation of concerns between resource providers and the clients that consume resources, and thus enable a *division of labor* [5].

We argue that to address management issues arising from this "division of labor," we must develop abstractions and tools that allow clients to dynamically configure, deploy, and manage required execution environments in application-independent ways, as well as to negotiate enforceable resource allocations for the execution of these environments. We have previously defined the *virtual workspace* abstraction [6] that meets many of these requirements. In this paper, we refine this abstraction to address dynamic resource allocation and management issues. In so doing, we provide a tool satisfying the separation of concerns requirements between resource provider and a client.

More specifically, in this paper we show how a workspace implementation based on virtual machines (VMs) can be extended to respond to negotiated resource allocations. We also quantify via experimental studies how well such allocations can be enforced. Finally, we report on the use of our workspace mechanisms to provide a platform for Edge Services [7]—VO-specific infrastructure services particularly sensitive to resource sharing—and discuss how workspaces are being used to support Edge Services on the OSG production grid.

In summary, our contributions in this paper are as follows:

- We describe an abstraction for providing mechanisms for dynamically negotiated resource usage as seen from the client's perspective.
- We show an implementation of this abstraction using the Globus Toolkit [8], the Xen virtual machine tools [9], and Linux networking tools.
- We describe how this abstraction and implementation has been used to realize Edge Services on the Open Science Grid.
- We evaluate our abstraction and implementation experimentally.

2 Related Work

The need for abstractions and mechanisms that enable a separation of resource provider and VO enforcement functions has been argued elsewhere [10, 11]. Our work here focuses on the management of such resource "slots," with a particular focus on issues that arise when managing more than one slot attribute at a time.

Several projects have explored the of VMs in distributed computing [12-15]. In particular, the Cluster-On-Demand (COD) [16] and VIOLIN [17] projects are relevant to our effort as they address resource management issues arising when overlaying virtual machines over physical resources. However, while these projects focus on simulating coarse-grain management of large numbers of VMs, we develop fine-grain abstractions and enforcement methods that provide detailed low-level management. We also model virtual resources in terms of allocations directly available to the resource user, requiring that overhead be accounted for separately. The Virtuoso Project [18] builds a VM scheduler and their approach is complementary to our work.

The interfaces we propose are informed by the standards work at the Open Grid Forum, specifically the work on WS-Agreement [19] and the Job Submission Definition Language (JSDL) [20]. We focus on the implementation of such interfaces to negotiating Grid abstractions and on demonstrating their relevance.

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. It is impossible for a resource provider to provide every bit of configuration for a VO, much less to arbitrate between competing demands from different VOs for different configurations. Instead, we want to allow resource providers to focus on providing and maintaining resources. In general, we want to enable a *provider* (e.g., a resource owner) to delegate the usage of a well-constrained resource quantum to a *consumer* (e.g., a VO) such that this consumer in turn can further distribute those resources among its customers (e.g., VO users). As we have explained elsewhere [6], this situation may involve many resource layers and employ different workspace implementations to achieve the desired fairness and granularity of sharing.

A compelling illustration of the need for division of labor between resource providers and VOs is provided by *Edge Services*, Grid middleware services that run at a site to enable remote access to site resources. Examples of Edge Services include job management services, storage brokers, and database caches. Edge Service implementations must often perform multiple privileged and unprivileged actions, such as data staging and registration, security processing, monitoring, and resource procurement. The variety of possible implementations means that Edge Services are often VO-specific. Different VOs upgrade them on a different schedule and may use conflicting versions of the software.

Further, each VO works with an often large and dynamically changing pool of users and has to mold its policies to 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 a significant burden affecting its ability to scale – and to prevent any one VO from impacting another. In a general case, such mechanism should provide separation along the following dimensions:

1) *Environment and configuration*: a VO should be able to obtain the configuration it needs independently of the resource provider.
2) *Isolation*: a VO's internal activities should not impact the resource provider, and thus should do not need to be under the provider's control.
3) *Resource usage and accounting*: a provider needs to be able to provide a resource in a way that is independent of how the resource is consumed.

We focus here on the third concern; the other two concerns are the subject of our ongoing research [6, 21].

4 Allocating Resources to Workspaces

The term Virtual Workspaces [21] a customized and isolated execution environment that can be dynamically deployed in the Grid. This environment is implemented by a workspace image, typically provided by the client, that contains all information necessary for its deployment. In addition, the client provides a resource allocation request that describes resources bound to the workspace at deployment time. The workspace service also provides management interfaces based on the Web Services Resource Framework (WSRF) [22], such as inspection and lifetime management. Workspaces can be implemented by various means, such as software imaging on physical resources as well as virtual machines. Here, we explain how the workspace service is used to assign resources to activities contained in the workspace.

4.1 Negotiating Resource Allocations

We define a workspace's *resource allocation* as those resources *directly* available to the workspace. This allocation does not include additional (overhead) resources that the resource provider requires to support the workspace's execution.

We model a resource allocation as a `ResourceAllocation` element, based on the JSDL [20] `Resources` element, with extensions as highlighted in Figure 1. Time is specified as start time and duration of deployment (in practice, only current start time is supported). Memory size is also specified as single value. CPU type is specified as a list of architectures (e.g., x86, IA64, x86_64) and percentage pairs to accommodate workspaces with multiple CPUs. Storage 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 `Storage` element may describe one or more partitions in terms of size and read/write speeds. (This information may determine if partitions need to reside in local storage or may be remote.) The networking element specifies the incoming and outgoing bandwidth, as measured by the iperf program for a virtual NIC. The values of duration, CPU percentage, size, read/write speed and bandwidth can be specified as JSDL RangeValue term. Currently, only exact values (JSDL Exact) and lower-bound open-ended ranges (JSDL LowerBoundedRange, interpreted to mean "the assigned value or more") are supported.

All values specified as part of `ResourceAllocation` can be specified as a ranges of acceptable values in a request (e.g., 50-60% of CPU), thus allowing a client to pose open-ended resource requests to be concretized by the workspace service on acceptance. Feasible resource allocation ranges are published by the workspace factory service, much like WS-Agreement agreement templates [19]. Information about feasible ranges takes into account resource coordination; for example, the current workspace service policy does not allow clients to specify resource allocations that cannot be realized as in the case where not enough CPU is requested for a certain network bandwidth.

Shaping resource assignment policy for a specific workspace has four stages: (1) a client defines a requested resource allocation, (2) the resource allocation is assigned based on negotiation with the resource provider, (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: a *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 value ranges of the requested resource allocation) is published as a WSRF resource property of the workspace.

Renegotiation is achieved by updating the resource property values. This updating can be performed either by sending a completely 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 value was sent with the adjusted value. 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.

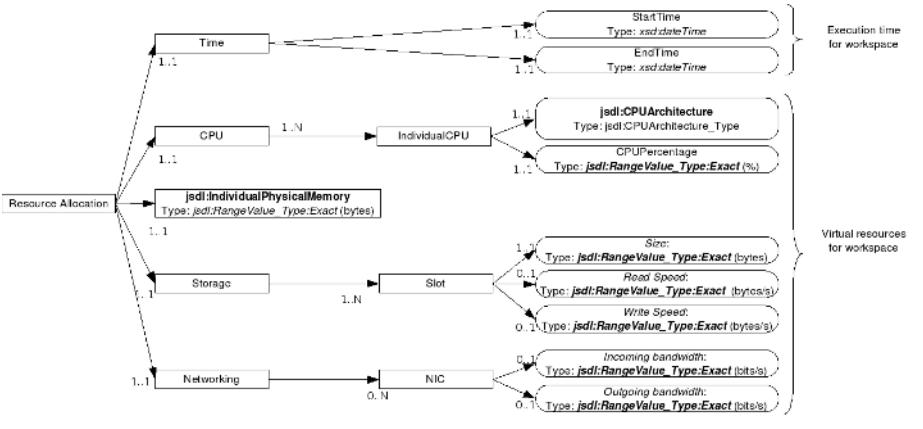

Fig. 1. The Resource Allocation Element

A client request may specify requirements for only some of the resources: for example, only memory and CPU. In such cases, default values for other resources are assigned by the workspace service. These default policies are published as resource properties by the workspace factory service. The current implementation provides only a "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

Our current VM-based workspace implementation deploys workspaces on a pool of resources configured with a hypervisor, and interacts with those workspaces through a configurable back-end. We focus in this paper on an implementation based on the Xen hypervisor [9]. In Xen, privileged hypervisor interaction usually takes place in domain 0, which allows a client to create and manage other virtual machines (called user or guest domains).

The physical memory size allocated to a Xen domain is specified when the domain is created. This memory size can be adjusted after startup using Xen's balloon driver. Two flavors of disk allocation are needed: obtaining storage for disk partitions that

form a part of a VM image, and providing extra writing space for the VM. We address the former requirement by mounting the VM partition as a loopback device. (A physical partition on the local disk could also be mounted, but those are already allocated.) We address the latter requirement 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 goal of best performance, 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.

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 domains. Weights can be specified exactly, or alternatively the scheduler can be allowed to give a domain extra CPU cycles if they are available, in which case it provides *at least* the specified weight. In practice, the latter policy results in better overall performance and was used throughout our experiments. The assigned CPU share as well as the policy can be changed dynamically. For CPU-intensive domains, we found that an assignment of 5-10% to domain 0 gave a good balance.

Effective resource allocation frequently requires coordinating more than one resource dimension. For example, when requesting a bandwidth allocation to a workspace, we have to ensure not only that the incoming/outgoing message rate is limited but also that the CPU share assigned to both the workspace and hypervisor overhead is sufficient to process messages. Since domain 0 in Xen acts like a switch, switching traffic to user domains, we have to maintain a balance between weights allocated to domain 0 and the guest domains: while domain 0 needs to process all incoming messages, giving it more weight at the cost of guest domains may result in poor performance as the guest domains are unable to process traffic in a timely fashion. In addition, while scheduling a domain more often improves the latency of this domain (as it is able to receive messages faster), it also results in higher context switching and thus CPU cost.

To develop rough guidelines for managing these tradeoffs, we conducted a parameter sweep over different CPU allocations for up to eight deployed domains, with the goal of obtaining an aggregate bandwidth to those domains within 1% of the maximum bandwidth achievable for our hardware (described in Section 6). We found that while CPU allocations for both the hypervisor and guest domains can vary due to various factors, it was possible to obtain acceptable bandwidth (within 1% of maximum for our hardware) by using bounds on those settings. Specifically, we adopted a bound of 20% of CPU for the domain 0 setting and guest bounds dependent on the number of guest domains. (For example, for the two domains case described in Section 6, the aggregate bound was 20%.)

The workspace service then bases its allocation decisions on those bounds and admits only allocations that can be supported within them. We find in our experiments that there is often a substantial difference between actual CPU utilization and the CPU weight assigned (typically less CPU gets used). However, weights are important to obtain the desired bandwidth. In addition, we find that while our bounds on CPU weights simplify resource management, they also allow further inefficiency to burden the system. Thus, our present SEDF-based resource management strategy

for CPU allocations is approximate and sacrifices much utilization to compensate for the relative lack of control that SEDF provides over CPU allocations for I/O intensive operations. Work on alternative schedulers that address the SEDF shortcomings leading to this inefficiency is underway [24, 25] and we plan to explore such alternatives in the future.

Xen does not implement controlled bandwidth sharing itself, so we rely on Linux network shaping tools [26] for that purpose. We again take advantage of the facts that the network interface of each domain is connected to a virtual network interface in domain 0 by a point-to-point link and that 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 using the Hierarchical Token Bucket queuing discipline [27]. To implement this behavior, we needed to recompile the domain 0 kernel. We developed an API to the Linux tools that allows us to set the bandwidth rates for created domains. We also increase the domain 0 CPU allocation by 2% to manage bandwidth splitting for user domains.

5 Case Study: Edge Services Framework

The Edge Services Framework (ESF) has been developed to decouple the tasks of (a) configuring and managing VO service nodes and (b) providing resources, thus allowing for division of labor between VO administrator and site administrator. ESF achieves this decoupling by leveraging the workspace abstraction to allow a VO administrator to configure an Edge Service image and deploy it 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 (basic OS configuration, at present including Scientific Linux 3/4, CentOS 3/4, and Fedora Core 4) and fully configured Edge Services images, currently including the ATLAS DASH service [28] and CMS FroNtier [29]. Since Edge Service images can be large in size (5 to ~10 GB), ESF uses compression and fast transport mechanisms (GridFTP [30]) as well as high-end Storage Elements (SEs) such as dCache [31].

The role of a VO administrator is to prepare, configure, and test an ES image. 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 VO administrator. In the current deployment, images stored on a site are further manually configured with required IP addresses and a pre-generated credential. We are working to automating those latter tasks as part of workspace deployment [21].

The role of a site administrator is to provision hardware resources that can be used for Edge Services and to ensure the proper configuration and maintenance of those hardware resources. In our testbed these tasks include configuring hardware resources with Xen, and providing one deployment of the workspace service per site. A site administrator also provisions storage space in a local Storage Element for storage and retrieval of ES images.

During site operation, Edge Service workspaces are dynamically retrieved, provisioned, and deployed by a VO administrator authorized using their VOMS

credentials [32]. 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.

The current ESF deployment spans both integration-level testbed sites and production-level sites on OSG. Integration-level sites include Argonne National Laboratory, Fermilab, University of Chicago, and UCSD. The production-level deployment is at DISUN [33] at SDSC.

6 Experimental Evaluation

To evaluate the usefulness of our approach, we performed experiments in a configuration comprising two VOs, VO1 and VO2. Each VO deployed an Edge Service on the same physical node, an AMD Athlon MP 2200 (booted in single CPU mode) with 2GB memory and a GB NIC over a 100 Mbps switch. The machine was configured with Linux 2.6.12 and Xen 3.0. The Edge Services were managed using our workspace service deployed on a different node than the services themselves and receiving requests from VO clients running on separate nodes.

6.1 Managing Execution Request Throughput

We first examined to what extent running within workspaces helps implement policy-regulated sharing between VOs. For this experiment, we modeled the work of a Compute Element (CE) by configuring an Edge Service with the GT4.0 GRAM service receiving non-staging requests (/bin/date) from the two VO clients. To simulate high load on VO1, we used an additional load client that submitted to VO1 a job performing 2 million square root operations every 10 seconds.

We considered a scenario in which one VO receives high request load in two different settings: (1) a physical machine setting in which the CE is deployed directly on the physical machine and used by both VOs (reflecting the situation in most current deployments), and (2) the workspace setting in which each VO deploys the CE in a Xen workspace, and each negotiates a resource allocation of at least 45% CPU weight and 896 MB of memory (domain 0 is set to 10% of CPU and 256 MB).

In both the physical and workspace setting, we measured the end-to-end job throughput of VO1 and VO2 clients. The job submission throughput was calculated over a period starting after the CE was saturated with load to a period when the VO2 client stopped submitting. As expected, the physical scenario resulted in roughly the same (low) request throughput for both VO clients (7.83 jobs/minute for VO1 and 8.0 jobs/minute for VO2), as the pre-existing VO1 load impacted both equally. In the workspace scenario, all VO1 request load is confined to VM1. Thus, we see worse throughput for VM1—4.18 jobs/minute—but observe that the request throughput for VO2 is now unimpacted by VO1 load and increases to 22.36 jobs/minute.

To observe CE behavior more closely over time, we summed the number of completed requests for VO1 and VO2 clients respectively during regular intervals (every 30 seconds) and plotted this number against time. (The values shown are an average of 5 trials and all fall within 10% of the average.) Figure 2 (left) shows the

comparison: VO2 has consistently high throughput while VO1 has low throughput until VO2 client stops sending requests, causing VM2 to temporarily stop consuming resources. VO1 is then able to claim a larger CPU weight and obtain more processing power (at 600 seconds the load client ceases to submit and the throughput improves further). Even when we varied the load coming from the load client by doubling or tripling its operations, the VO2 request throughput was unimpacted (right side of Figure 2) and stayed at the same level independently of the load on VO2.

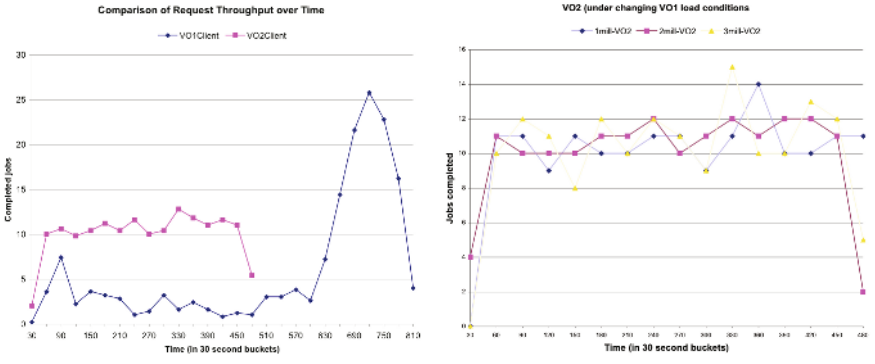

Fig. 2. Request throughput for VO1 and VO2. The graph on the left compares request throughput for VO1 and VO2 over time. The graph on the right shows that the VO2 request throughput remains unimpacted in the presence of increasing load on the physical machine.

6.2 Managing Network Traffic

In this experiment we model a situation that might be experienced during the operation of a Storage Element (SE), in which two VOs run services requiring heavy network traffic. As in the previous example, running the SE inside one workspace does not result in fair sharing. Unlike our previous example, effectively managing such traffic requires simultaneous management of two qualities: the bandwidth leased to each workspace assigned to the respective VOs, and the CPU share required to support the processing of message arrival rates.

Our experimental scenario is the same as in the previous experiment. To isolate network performance, we modeled the work of an SE with a GridFTP 4.0.1 performing memory-to-memory transfers. In our scenario both VO1 and VO2 have each negotiated a resource allocation of 128 MB memory, 6% CPU weight and 4.1 MB/s raw bandwidth (the CPU weight is the minimum needed to process this bandwidth as per our baseline). The workspace service allocates a weight of 22% to domain 0; all remaining CPU share is always taken up by an additional CPU-intensive domain to ensure enforcement.

Figure 3 shows a time trace of the behavior of the two workspaces. VO2 processes a steady stream of requests and saturates its bandwidth from the beginning of the trace. VO1 performs transfers of various sizes for a while and then it too begins to saturate its bandwidth (~150 seconds of the trace). Note that neither domain achieves the raw 4.1 MB/s bandwidth due to application (GridFTP) overhead. After about 240

seconds (first arrow mark on the graph), the VO1 renegotiates its allocation to request a higher bandwidth of 8.2 MB/s requiring at least 14% of CPU (as per our baseline transfer utility measurements). As a result the workspace service adjusts the CPU settings (from 6 to 14) as well as bandwidth settings (from 4.1 to 8.2) on the hypervisor node and the bandwidth for VO1 goes up. To compensate for application overhead, the VO then further renegotiates just the CPU weight from 14 to 34 which brings performance closer to the desired bandwidth.

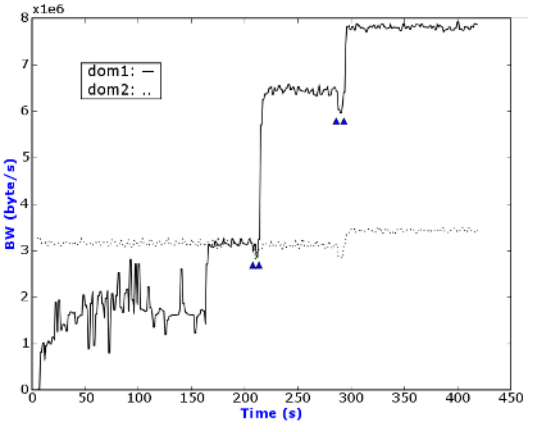

Fig. 3. Isolating SE for VO1 and VO2; the arrows indicate renegotiation points

Renegotiation (Figure 3) takes about 7 seconds. Most of this time (~5 seconds) is taken by SEDF weight adjustments, a factor that is responsible for the indentation in the graph during negotiation time. Increasing the domain 0 allocation speeds up this processing, at a cost of lower guarantees offered to user domains. The times taken by bandwidth adjustment and end-to-end request processing are ~400 ms and ~300ms (WS overhead), respectively.

7 Conclusions

Our deployment and experimental experiences lead us to conclude that workspaces are a promising "division of labor" tool for providers and consumers. We believe that this conclusion is especially true in situations in which the task of understanding interdependencies among various load-causing factors is complex, and indeed potentially impossible to manage at the application level. Load management on OSG service nodes is an example of such a situation, due to the complex and time-varying workloads that may be generated by different clients.

The ability to negotiate and renegotiate resource allocations is particularly important, both to the client and the provider. This ability allows both client and provider to react to changing load conditions and optimize their provisioning to satisfy targets, for example by requesting more CPU to account for application overhead in the GridFTP example. Additional flexibility can be obtained by using migration, as demonstrated by Clark et al. [34] for an application similar to ours.

Our results indicate that it is useful to support "aggregate" resource allocations that combine, for example, bandwidth and CPU allocations, for the reason that an allocation of one resource (e.g., bandwidth) may be wasteful unless matched by at another (e.g., at least a minimal CPU allocation to match a bandwidth allocation). It is a useful aid, and potentially a useful policy, for the resource provider to publish offers of such "bundled" allocations.

Finally, while we managed to obtain reasonable results in bandwidth-related allocations with the SEDF scheduler, we also found that tractable allocations with this scheduler are hard to determine for management purposes, enforce, and account for, both in terms of hypervisor overhead and actual guest domain usage. Ongoing work on credit-based schedulers [24, 25] may overcome some of those issues. Our approach was to use bounds within which allocations could be supported. However, this approach can introduce inefficiencies, which may decrease utilization for the resource provider.

Acknowledgements

This work was supported by NSF CSR award #527448 and, in part, by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, SciDAC Program, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

References

1. Foster, I. and others. The Grid2003 Production Grid: Principles and Practice. in HPDC 2004: IEEE Computer Science Press.
2. Open Science Grid (OSG). 2004: www.opensciencegrid.org.
3. The TeraGrid Project 2005: www.taragrid.org
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. 1776
6. Keahey, K., I. Foster, T. Freeman, and X. Zhang, Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid. Scientific Progamming Journal, 2005.
7. ESF: http://osg.ivdgl.org/twiki/bin/view/EdgeServices/WebHome.
8. Foster, I., Globus Toolkit version 4: Software for Service-Oriented Systems. 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 SOSP 2003
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. 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, The Grid: Blueprint for a New Computing Infrastructure. 2004, Morgan Kaufmann.
15. 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.
16. Irwin, D., J. Chase, L. Grit, A. Yunerefendi, D. Decker, and K. Yocum, Sharing Networked Resources with Brokered Leases. 2006: in submission, available at http://issg.cs.duke.edu/publications/sisyphus.pdf.
17. Ruth, P., J. Rhee, D. Xu, S. Kennell, and S. Goasguen. Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure. in ICAC. 2006.
18. Lin, B. and P. Dinda, VSched: Mixing Batch And Interactive Machines Using Periodic Real-time Scheduling. Supercomputing, 2005.
19. 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) 2004: https://forge.gridforum.org/projects/graap-wg/.
20. 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.
21. Lu, W., T. Freeman, K. Keahey, and F. Siebenlist, Making your workspace secure: establishing trust with VMs in the Grid. SC05 Posters, 2005.
22. 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.
23. Xen Scheduling: http://wiki.xensource.com/xenwiki/Scheduling.
24. Gupta, D., L. Cherkasova, R. Gardner, and A. Vahadat, Enforcing Performance Isolation Across Virtual Machines in Xen. HP-2006-77, 2006.
25. Xen CPU Scheduler w/SMP Load Balancer: http://lists.xensource.com/archives/html/xen-devel/2006-05/msg01315.html.
26. Linux Advanced Routing and Traffic Control: http://lartc.org.
27. Devera, M., Hierarchical Token Bucket Queuing. 2005: http://luxik.cdi.cz/~devik/qos/htb/.
28. Vaniachine, A., DASH: Database Access for Secure Hyperinfrastructure: OSG document 307. http://osg-docdb.opensciencegrid.org/cgi-bin/ShowDocument?docid=307.
29. Lueking, L., FroNtier project: http://lynx.fnal.gov/ntier-wiki.
30. 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.
31. The dCache Project: http://www.dcache.org.
32. The Virtual Organization Management System: http://infnforge.cnaf.infn.it/projects/voms.
33. Data Intensive Sciences University Network: http://disun.org.
34. Clark, C., K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, Live Migration of Virtual Machines. NSDI, 2005.