A Model for Usage Policy-based Resource Allocation in Grids

Catalin L. Dumitrescu
Dept. of Computer Science
The University of Chicago
catalind@cs.uchicago.edu

Michael Wilde
Math and CS Division
Argonne Natl. Laboratory
& University of Chicago

Ian Foster
Math and CS Division
Argonne Natl. Laboratory
& University of Chicago

Abstract

Challenging usage policy issues can arise within virtual organizations (VOs) that integrate participants and resources spanning multiple physical institutions. Participants may wish to delegate to one or more VOs the right to use certain resources subject to local policy and service level agreements; each VO then wishes to use those resources subject to VO policy. How are such local and VO policies to be expressed, discovered, interpreted, and enforced? As a first step to addressing these questions, we develop and evaluate policy management solutions within a specialized context, namely scientific data grids within which the resources to be shared are computers and storage. We propose an architecture and recursive policy model, and define roles and functions, for scheduling resources in grid environments while satisfying resource owner and VO policies.

1. Introduction

We consider scenarios in which *providers* wish to grant to *consumers* the right to use certain *resources* for some agreed-upon time period. Providers might be companies providing outsourcing services, or scientific laboratories that provide different collaborations with access to their computing resources. Providers and consumers may be nested: a provider may function as a middleman, providing access to resources to which the provider has itself been granted access by some other provider. Usage policy issues can arise at multiple levels in such scenarios. Providers want to express (and enforce) the policies under which resources are made available to consumers. Consumers want to access and interpret policy statements published by providers, in order to monitor their agreements and guide their activities. Both providers and consumers want to verify that policies are applied correctly. In summary, we are interested in the expression, publication, discovery, enforcement, and verification of policies, at both resource provider and consumer levels.

We report here on work that addresses these issues within a specific problem domain, namely the distributed analysis of large quantities of scientific data [1]. In so-called "data grids," we have a three-level structure in which individual scientists and sites provide resources (computers, storage, and networks) to scientific collaborations that in turn provide resources to their members. Providers and consumers negotiate service level agreements (SLAs) to establish what resources providers make available for consumer use. VOs must then allocate aggregate resources provided by different owners to different VO purposes, and orchestrate distributed data analyses to use those aggregated resources efficiently.

This problem encompasses challenging and interrelated policy, scheduling, and security issues. We focus here on policy issues, although from a scheduling perspective. Specifically, we seek to address the following questions: *"How usage policies are enforced at the resource and VO level?", "What strategies must a VO deploy to ensure usage policy enforcement?", "How are usage policies distributed to enforcement points?",* and *"How usage policies are made available to VO job and data planners?"* In addressing these questions, we build on previous work concerning the specification and enforcement of local resource scheduling policies [2,3,4,5,6]; the negotiation of SLAs with remote resource sites [7,8]; and expressing and managing VO usage policies [9]. We extend this work to Grid environments, such as Grid3 [10], which are composed of sites, VOs, VO groups and other entities.

2. Problem Statement

The grids that we target in this work may comprise hundreds of institutions and thousands of individual investigators that collectively control tens or hundreds of thousands of computers and associated storage systems [11,12]. Each individual investigator and

institution may participate in, and contribute resources to, multiple collaborative projects that can vary widely in scale, lifetime, and formality. At one end of the spectrum, two collaborating scientists may want to pool resources for the purposes of a single analysis. At the other extreme, the major physics collaborations associated with the Large Hadron Collider encompass thousands of physicists at hundreds of institutions, and need to manage workloads comprising dynamic mixes of varying priority work, requiring the efficient aggregation of computing and storage elements [14].

Our approach considers two classes of entities: resource providers and resource consumers. A physical site is a resource provider; a VO can be both a resource consumer (consuming resources provided by a site) and a provider (providing resources to users or user groups). We assume that each provider-consumer relationship is governed by an appropriate SLA, but do not address the nature of these SLAs [14].

We use a simple example to illustrate some issues that can arise. Assume that provider P has agreed to make R resources available to consumer C for a period of one month. How is this agreement to be interpreted? These resources might be dedicated to C or, alternatively, P might make them available to others when C is not using them. In the latter case, P might commit to preempt other users as soon as C requests them, or might commit to preempt within a certain time period. If C is allowed to acquire more than R resources when they are not used, then this may or may not result in C's allocation being reduced later in the month. C may or may not allow reservations.

A VO in its role as both resource consumer (from sites or other VOs) and provider (to its consumers: users or groups within the VO) acts as a *broker* for a set of resources. These brokering functions can be implemented in at least three different ways. The VO could determine a partitioning of the resources to which it has negotiated access, and then work to establish SLAs directly between its consumers and its providers that reflect this partitioning. Or, if its providers support the necessary mechanisms, the VO could hand its consumers some form of digital ticket that the VO could present to any of the VO's resources [15]. In both cases, the VO need not have any further involvement in the consumer-provider relationship.

3. Specification Syntax and Semantic

In the experiments described in this paper we represent a policy for the allocation of resources by a provider to a consumer as a set of *resource allocations* of the form:

```
< resource-type, provider, consumer,
  epoch-allocation, burst-allocation >
```

where:

```
resource-type ::= [ CPU | NET | STORAGE ]
provider ::= [ site-name | vo-name ]
consumer ::= [vo-name|(vo-name,group-name)]
epoch-allocation ::= (interval, percentage)
burst-allocation ::= (interval, percentage)
```

In the grid environment that we consider here, allocations can be made for processor time, storage, or network bandwidth. An allocation specifies a maximum percentage of a resource type managed by the provider, which it allocates to the designated consumer for a specified period of time. Note that this definition of policy yields a multi-level hierarchy of resource assignments: to a VO, by a resource owner; to a VO user or group, by a VO; and so on.

Limits are specified for both long-term scheduling "epochs" and for "burst periods." Both periods are modeled here as recurring within fixed time slots. A provider may grant requests above the epochal allocation if sufficient resources are available, but these resources can be preempted if other parties with appropriate allocations request those resources at a later stage. Our starting point for this approach is the Maui [4] semantic in specifying resource utilizations. In the Maui context, "fair share" is the use of historical CPU data to influence job priority with a goal of allocating certain percentages of available CPU cycles to particular users or groups.

For MAUI, a fair share percentage value would lower or raise the priority of currently user queued jobs in an attempt to bring the CPU utilization in line with the fair share target. For example, if a group has used less than its share of the available cycles over a period of time, then fair share would increase the priority of the same group next queued jobs, attempting to bring its utilization up to the share target. Another important aspect of fair share is how historical CPU utilization is determined. Utilization is calculated by breaking up the scheduled time into intervals. The length of each fair share interval is specified by two parameters, FSINTERVAL, and FSDEPTH. FSINTERVAL specifies the time interval, while FSDEPTH specifies how many intervals must be considered for the fair share computation. A third parameter, FSDECAY, determines the weight of each interval used in the overall CPU usage calculation [4].

For simplification, we consider in this paper only "average limits", and use (FSINTERVAL, FSDEPTH, FSDECAY) = (3600, 1, 1) for both epoch and burst utilizations. Examples of usage policy specifications in our model are: $Site_1$ gives VO_0 20% of its CPU resources over a 1-hour period using this policy tuple:

```
[CPU, Site1, VO0, (3600s,20),(5s,60)]
```

and VO_0 gives its own $Group_0$ 40% of its CPU resources over a 1-hour period using:

[CPU, VO_0, $Group_0$, (3600s,40),(5s,90)]

4. Usage Policy Enforcement Prototype

The main questions we seek to answer with the prototyping and simulation efforts we describe here are the following: "*What strategies must a VO deploy to ensure grid-wide resource policy enforcement in a decentralized manner which we believe is required for practical grid deployment?*", and, at a site level, "*Can we implement VO-based site policy enforcement in a simple and portable manner for the popular local schedulers that are deployed in our application environments, as typified by PBS and Condor?*" These questions try to capture the main problems that can be faced in practice. A grid may be composed of a large number of entities, and in the same time be composed of heterogeneous components that run at different levels. We consider important to address these questions, otherwise our solution is too specific to be accepted for today existing grids.

In order to gain practical experience with these issues we have developed a prototype system that implements the usage policy management model introduced before and used it to control the allocation of computing resources. In our prototype, usage policies are entered through a web interface and disseminated to associated enforcement points. The prototype environment which we used for our evaluation and experimentation is depicted in Figure 1. Most experiments were conducted on a simple grid consisting of two execution sites, two VOs, and two submission sites [1].

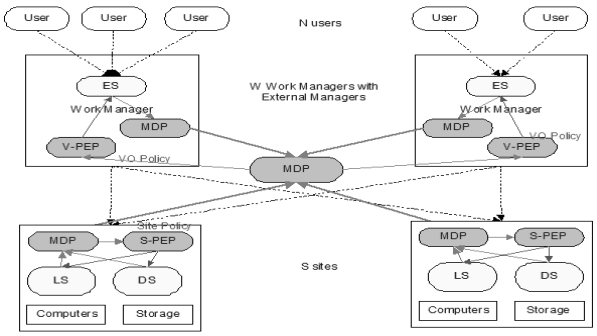

Figure 1: VO-Level Architecture

Policy enforcement points (PEPs) are responsible for executing policies. They gather monitoring metrics and other information relevant to their operations, and then use this information to steer resource allocations as specified by the usage policies [16,17].

We distinguish between two types of PEPs. *Site policy enforcement points* (S-PEPs) reside at all sites and enforce site-specific policies. S-PEPs operate in a continuous manner, in the sense that jobs are immediately preempted or removed when policy requirements are no longer met. Jobs are not, however, necessarily restricted from entering site queues just because policy would prevent them from running. In sub-section 4.1 we present a further refinement of our S-PEP in a concrete context, the Grid3 environment.

VO policy enforcement points (V-PEPs), labeled V and associated with VOs, operate in a similar way to S-PEPs. They make decisions on a per-job basis to enforce policy regarding VO specifications for resource allocations to VO groups or to types of work executed by the VO. V-PEPs are invoked when VO planners make job planning and scheduling decisions to select which jobs to run, when to send them to a site scheduler, and which sites to run them at. V-PEPs interact with S-PEPs and schedulers to enforce VO-level policy specifications.

4.1. Site Usage Policy Enforcement

We describe here two solutions for site usage policy enforcement, and in Section 6 compare them in terms of the capacity to achieve resource sharing according to policy.

Solution 1 (Stand-alone S-PEP): Our first solution does not require a usage policy-cognizant cluster resource manager. It works with any primitive batch system that has at least the following capabilities: provide accurate usage and state information about all scheduled jobs, job start/stop/held/remove capabilities, and running job increase/decrease priority capabilities.

The S-PEP sits at the level of the local scheduler(s), checks continuously the status of jobs in all queues and invokes management operations on the cluster resource manager when required to enforce policy. In more detail, the S-PEP gathers site usage policy from the policy enforcement distribution module, collects monitoring information from the local schedulers about cluster usage, computes CPU-usage parameters, and sends commands to schedulers to start, stop, restart, hold, and prioritize jobs. Our approach provides both priority-based and feasibility-based enforcements.

Priority-based enforcement involves only job's priority modifications without preempting or removing jobs from execution. Practically, all jobs are allowed to run, but resource utilization is controlled by means of what share are allocated during the execution. Feasibility-based enforcement assumes that all jobs run with the same priority, and the only approach to enforce usage policy is to hold/start or pre-empt jobs from execution. Note that the S-PEP does not have its own queue.

The processing logic of our prototype S-PEP is based on the algorithms presented below:

```
1. foreach VOi with EPi
2.    # Case 1: fill BPi
3.    if Σ(BAj)==0 &
         BAi < BPi & Qi has jobs then
4.      release job i from some Qi
5.    # Case 2: available and BAi < BPi
6.    else if  Σ(BAk)<TOTAL &
        BAi<BPi & Qi has jobs then
7.      release job i from some Qi
8. # Case 3: res. contention: fill EPi
9. else if  Σ(BAk)==TOTAL &
              BAi<EPi & Qi has jobs then
10.    if j exists & BAj >= EPj then
11.       suspend an over-quota job Qj
12.    release job i from some Qi
13. foreach VOi with EPi
14.    if EAi>EPi then
15.     suspend jobs for VOi from all Qi
```

where:

```
EPi = Epoch allocation policy for VOi
BPi = Burst allocation policy for VOi
Qi  = set of queues with jobs from VOi
BAi = Burst Resource Allocation for VOi
EAi = Epoch Resource Allocation for VOi
TOTAL = possible allocation on the site
Over-quota job = job of VOj
```

As a further clarification, BA or EA represents the share actually utilized by a VO, computed based on the terms defined in Section 3. BP or EP represents upper values for these utilized shares. When BP or EP increases for example, the VO is entitled to more shares starting with the moment of the change. However, we avoided variations in our experiments.

An important novelty of our S-PEP over a cluster resource manager is its capability to keep track of jobs under several resource managers and to allow the specification of more complex usage policies without the need to change the actual cluster resource manager implementation. The results are captured in section 6.

Solution 2 (Policy-Aware Scheduler): Our second solution was developed and implemented with success in the context of Grid3 environment. We decouple the functionalities of the S-PEP in two major components and map to existing solutions: a standalone site policy observation point (S-POP) and the cluster resource manager modules for resource allocation control. In this case, we assume that the cluster resource manager is able to enforce by itself the desired usage policies, which are provided by means of our S-POP module. Examples of such cluster resource managers are Condor [18], Portable Batch System [3], and Load Sharing Facility [5], widely used on Grid3 [10]. This solution is similar to MAUI's approach in working in conjunction with various RMs.

The S-POP's main functions are to optionally provide and translate to/from the RM understanding what usage policies have to be enforced, and to monitor the actual resource utilization. An advantage of this solution is that site administrators do not have to use an additional grid component in managing their clusters. Again, our implementation of the S-POP has the advantage of interfacing with more than one resource manager at a time, but with slight MAUI modification the same goals could be achieved.

4.2. VO Usage Policy Enforcement

At the VO level, the V-PEP is cognizant of VO policies. Our V-PEP modules operate at the submission host queues. The V-PEP executes its logic to determine whether new jobs should be submitted to sites according to the VO usage policies. We have modeled a version of the logic presented in this sub-section using a "first-come-first-served" scheme with opportunistic scheduling mechanisms to utilize effectively resources. Opportunistic scheduling allocates free resources whenever VOs which are entitled to schedule jobs do not have any work to run.

The V-PEP provides answers to two questions: "*What jobs should be scheduled next?*", and "*When job j should start?*". The third question important here, "*Where job j should run?*", is also addressed by our V-PEP prototype, but we consider this question beyond the size and purpose of this paper. Also, the V-PEP has different functions than S-PEP presented before. The difference is driven by the large amount of information that must be collected. The logic of our V-PEP is described by the algorithm below, which can be seen to be symmetric to that of the S-PEP:

```
1. foreach (Gi with EPi, BPi, BEi)
2.    # Case 1: fill BPi + BEi
3.    if Σ(Baj)==0 &
         BAi<Bpi & Qi has jobs then
4.      schedule a job from some Qi
          to the least loaded site
5.    # Case2: res. available &BAi<BPi
6.    else if (Σ(BAk) < TOTAL)
          & BAi<BPi & Qi has jobs then
7.     schedule a job from some Qi
         to the least loaded site
8.  # Case 3: rs. contention: fill EPi
9.  else if (Σ(Bak)==TOTAL)&(BAi<EPi)
            &(Qi exists) then
10.    if (j exists & BAj>=EPj) then
11.       stop scheduling jobs for VOj
12.     # Need to fill with extra jobs?
13.     if (BAi < EPi + BEi) then
14.       schedule a job from some Qi
            to the least loaded site
15. if (EAi < EPi) & (Qi has jobs) then
16.    schedule additional backfill jobs
```

4.3. Verifying Monitoring Infrastructure

Accurate monitoring is important if we are to understand how our prototype actually performs in different situations. As a first step towards this goal, we develop mechanisms for measuring how resources are used by each VO and by the grid, overall.

As monitoring infrastructure, we build upon the Ganglia Cluster Monitoring toolkit [19], a distributed solution for collecting cluster monitoring information. The system is composed of a *host sensor collector,* a *summation meta-daemon* and a *cluster/host web-interface*. We enhanced this monitoring tool by transforming the summation meta-daemon into a *monitoring distribution point* (MDP), able to exchange data with other MDPs and to collect information about hardware usage, VO-related usage, and policy restrictions [14]. Information exchanged by MDPs is encoded in XML format and it is composed of different metrics for different entities. In order to avoid large amount of information gathering at the root MDPs and to distribute the process of maintaining monitoring information, we introduce various summation operations for different metrics. This approach provides scalability by ensuring that the same amount of information is exchanged among MDPs at different levels in the monitoring tree. The frequency with which information is exchanged also influences monitoring system capabilities and accuracy. For GangSim, both monitoring information collection and distribution are performed at pre-specified time intervals. In addition, MDPs monitor local host load, and if this increases over a pre-configured value, the exchange interval is increased, thus reducing the accuracy of monitoring information.

4.4. Usage Policy Management

While previous sections answered three of the questions stated in introduction, the *"How are usage policies distributed to enforcement points?"* question has remained yet unanswered. In this sub-section we provide a simple solution we tested with our prototype over Grid3 [10].

In our approach, the monitoring system (MDPs) provides in addition support for usage policy specification and distribution. The usage policies are specified through a web interface that connects to a selectable MDP. Further, the MDP distribute these usage policies to all site and submission host MDP that are associated with it. Even further, in a multi-layer deployment (cascading MDPs), each MDP multicasts periodically its usage policies to the other MDPs. While this solution is not very scalable (thousands of MDPs), we assume that for a grid ten times larger than today Grid3 is sufficient. As an additional note, usage policies are associated with the MDP that distributed it and with the console that provided it and they can be deleted only by the same point of decision. While this is an important point of our solution, we analyze in more detail this problem somewhere else [23].

5. Architecture Simulations

We describe here simulation studies that we conducted to evaluate the grid-wide resource allocation model that we presented in the previous section. In particular, we wanted to determine whether CPU resources could be allocated in a fair manner across multiple VOs, and multiple groups within a VO, without requiring the centralized control that is impractical in a grid environment.

We conducted our grid-wide simulations by discrete simulation mechanisms that generated performance data which was fed directly into the Ganglia collectors. The simulators ran in "real time" rather than using a simulation clock. The simulations were conducted using a module that managed all internal structures for our VO-Ganglia simulation tool [14], based on predefined clusters, workloads, and usage and scheduling policies components.

We ran our tests on a simulated grid consisting of two sites with a total of 22 CPU resources, 7 in one and 15 in the other, and two VOs. The scheduling policies used were based on FCFS (at both the VO and the site level), without preemption, and with opportunistic strategies to increase overall utilization.

The complexity of the environment used in this first set of simulations is specifically low in order to provide case scenarios that can be followed easily.

Workload: We used a composite workload that overlays work for two VOs, each consisting of two groups. The workload consisted of the 440 jobs shown in Table 1. All workloads are synthetic being composed of jobs, each corresponding to a certain amount of work and without any precedence constraints.

Table 1: Grid-wide workload summary

VO	Group	#jobs	Mean Job Duration
0	0	80	150 sec
0	1	100	250 sec
1	0	120	200 sec
1	1	140	300 sec

Jobs arrive, are executed, and leave the system according to a Poisson distribution. Because in our simulations we consider an environment with several VOs, an important factor is synchronization among

workloads. Here, we all workloads start at the same moment in time (synchronized), very appropriate to a scientific community that work for gathering results before a conference deadline.

The simulation period was one hour (one scheduling epoch) and the measurement interval was five seconds. Job durations and inter-arrival times were generated randomly using Poisson and Gaussian distributions, respectively.

***Policy*:** The two sites had different local policies, as specified by the following tuples. All burst intervals were five seconds, while epoch periods were one hour.

Allocating grid resources from 2 sites to 2 VOs:

```
(1) [CPU, Site1, VO0, (3600,20),(5,60)]
(2) [CPU, Site2, VO0, (3600,20),(5,60)]
(3) [CPU, Site1, VO1, (3600,80),(5,90)]
(4) [CPU, Site2, VO1, (3600,80),(5,90)]
```

Allocating VO resources from 2 VOs to 4 groups:

```
(5) [CPU, VO0, Group0, (3600,40),(5,90)]
(6) [CPU, VO0, Group1, (3600,60),(5,90)]
(7) [CPU, VO1, Group0,(3600,50),(5,100)]
(8) [CPU, VO1, Group1,(3600,50),(5,100)]
```

***Job states*:** The graphs in this section are based on a model where jobs pass through four states: submitted by a user to a submission host; submitted by a submission host to a site, but queued or held; running at a site; and completed.

Each view of VO or group resource allocation and utilization, below, is shown as a pair of graphs. Even numbered figures show the workload for the VO or group, broken down by jobs in different states of grid scheduling, as follows. Dark gray lines represent jobs executing at a site; white lines represent jobs queued or hold at sites; and black lines represent the total number of jobs at the site (the sum of the two previous states). Jobs at the job submission site (or planner) and finished jobs are not shown. The gray area is the total number of CPUs (22) available for job execution. The X axis for all graphs represents seconds, while the Y axis represents percentages in terms of the total available computing power (22 CPUs in this section).

Odd numbered figures show policy enforcement actions by overlaying CPU utilization on top of policy limits. The straight policy line shows percentage of total grid resources allocated; it reflects percentage of grid to VO and then percentage of VO to group. The straight policy line shows the cumulative percentage of epoch allocation used at each point in time. The black curve shows the burst (or instantaneous) CPU allocation, sampled in these tests every five seconds.

Allocation of Grid resources to VOs (S-PEP in conjunction with V-PEP): We first look at the effectiveness of policies 1 through 4, above, at allocating the resources of both grid sites to two different virtual organizations, VO_0 and VO_1. Figure 2 shows the effect on the jobs of VO_0 of policies 1 through 4. illustrates the policy levels for VO_0 and the effects of enforcement.

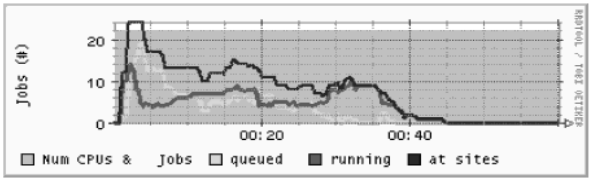

Figure 2: VO_0 job execution on two sites

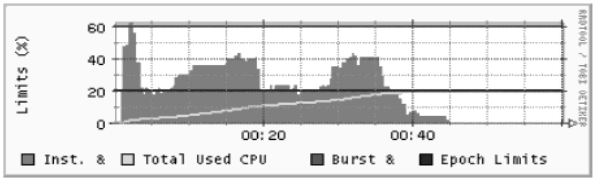

Figure 3: Usage Policy for VO_0 on two sites

We see in that VO_0 is allocated 20% of the overall grid resources, and is permitted short-term bursts in which it can use up to 60% of all grid resources. At about t=00:02 into the test, we observe that VO_0's workload hit its maximum burst limit, and was throttled back by the policy enforcement logic.

Immediately after that point, at about t=00:04, as VO_1's workload begins to build, we see () that VO_0's allocation is throttled back to its epochal limit (20%) until about t=00:07, when VO_1's contention for resources drops back to a level that once again permits VO_0 to gain resources above its epochal allocation. The same usage policy enforcement action takes place from about t=00:19 to t=00:28.

Finally, at t=00:37, we see VO_0's total CPU consumption for this epoch (rising light gray line) hit its epoch allocation limit (black line), after which the PEP ensures no new jobs are initiated for VO_0 until the end of the current scheduling epoch (at t=01:00).

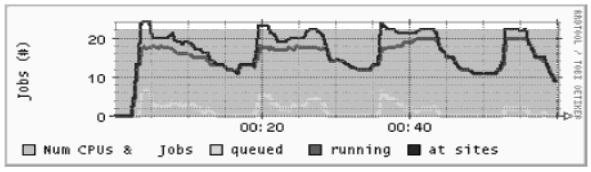

Figure 4: VO_1 Jobs execution on two grid sites

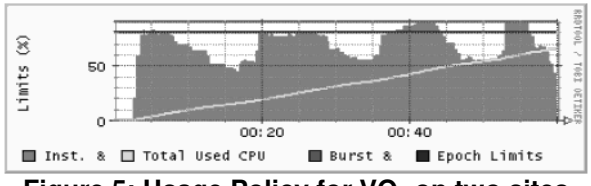

Figure 5: Usage Policy for VO_1 on two sites

Observing the grid-wide resource utilization of VO_1 in , we can see that its allocation of grid resources is limited to its 80% epochal allocation due to contention from VO_0 from t=00:03 to t=00:08, and t=00:20 to t=00:29. At t=00:40 and t=00:53 contention

from VO_0 has diminished to the point where the S-PEP permits VO_1 to reach its burst allocation of 100%. Note that within the observed epoch, VO_1 comes close to, but does not exceed, its allocation.

Allocation of VO resources to Groups (V-PEP): In a manner analogous to enforcing policy for the allocation of site resources to VOs, our distributed PEP model can simulate the allocation of VO resources to groups within a VO and measure the effectiveness of policy enforcement of such sub-allocations. We describe here the results of simulating the enforcement of policy rules 5 through 8 above, which specify that the CPU resources allocated from the grid sites to VO_0 are to be sub-allocated to VO_0's $Group_0$ and $Group_1$ on a 40% / 60% basis.

We see in Figure 7 that VO_0 $Group_0$ is throttled back to its epochal limit at two different points (t=00:0 and t=00:22), and then exceeds its epochal allocation at t=00:32. Similar policy enforcement of $Group_1$ can be seen at several points in Figure 9. $Group_1$, too, exceeds its epochal limit somewhat later, at t=00:40, after which jobs running at that time complete but no new jobs are scheduled. Both of these groups then carry their workload into the next scheduling epoch.

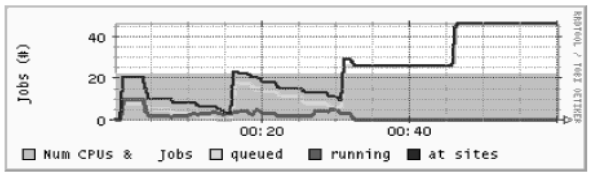

Figure 6: Workload for VO_0 $Group_0$

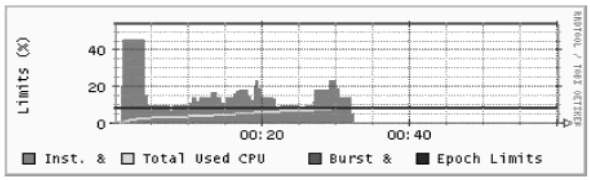

Figure 7: Policy enforcement for VO_0 $Group_0$

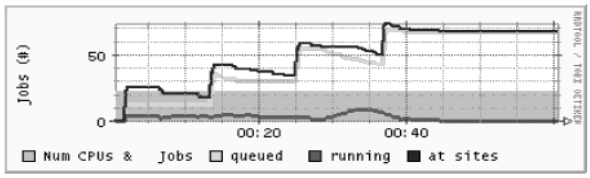

Figure 8: Workload for VO_0 $Group_1$

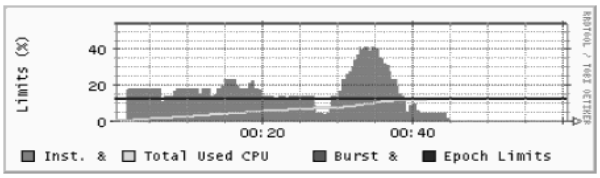

Figure 9: Policy enforcement for VO_0 $Group_1$

Overall Grid Utilization: Figures 10 and 11 show overall grid utilization by all VOs. We see from Figure 11 that CPU resources are well utilized whenever sufficient demand exists, despite the allocation-limiting actions of the PEPs.

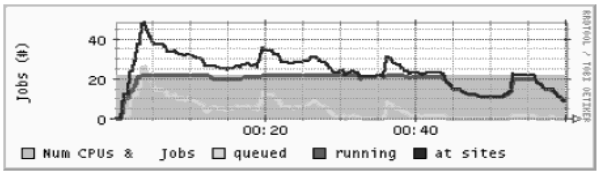

Figure 10: Overall grid workload

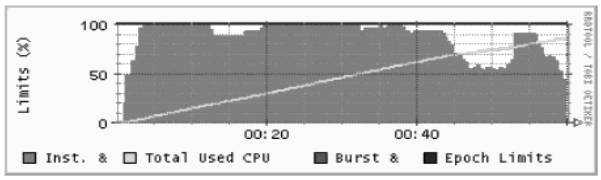

Figure 11: Overall grid CPU utilization

Testing generality: As an initial test of the generality of our approach to handle a larger grid, we ran a simulation test of the same workload and policies as were specified above for two VOs and four groups, but on a grid of 5 sites and 83 hosts.

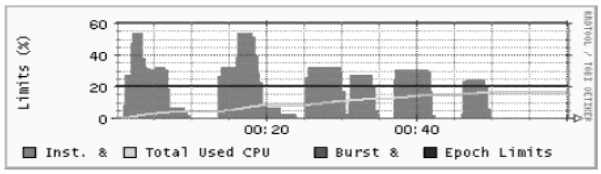

Figure 12: Policy for VO_0 on 5 sites

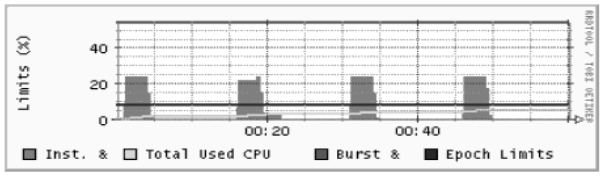

Figure 13: Policy for VO_0 $Group_0$ on 5 sites

Figure 12 captures CPU utilization on five sites (burst in middle gray and epoch in light gray) over a 1-hour scheduling epoch by two VOs, each composed of two groups. The grid-wide policies in effect are (epoch/burst) 20/60 for VO_0 and 80/90 for VO_1, with 40/60 for VO_0/G_0 and 60/80 for VO_0/G_1. These policies are represented with black/dark gray (epoch limits / burst limits). Figure 13 captures the resource sharing between the two groups as implemented by the VO policy mechanisms. Because the test grid for this example was large enough, there was no contention at the group level and jobs run without policy limitations.

6. Usage Policy Experimental Results

Here we report on experiments that compare the two S-PEP solutions introduced in Section 4.1. We measure how well local resource managers, in particular Condor and Maui in conjunction with Open-

PBS, are able to enforce our proposed extensible usage policy specification. In each case, two VOs submit workloads identical to those used in the previous simulations to a single site ($Site_0$), with one of our two S-PEP approaches used to enforce a usage policy in which CPU resources are allocated 20% to VO_0 and 80% to VO_1. The two VOs are allowed 30 second burst utilizations of 60% and 90% of the site's CPU resources, respectively:

```
[CPU, Site0, VO0, (3000s,20%),(30s,60%)]
[CPU, Site0, VO1, (3000s,80%),(30s,90%)]
```

Jobs were submitted via the Globus Toolkit® 2.0 [20]. The test site was monitored by collecting load information every 10 seconds. As a side note, we are not aware of other schemas for usage policy enforcement at the site level in the form proposed here. The closest match [15] does global enforcement for usage policies (grid-level).

6.1. S-PEP Usage Policy Enforcement

The first set of experiments involves our S-PEP implementation, which performs policy enforcement actions every 30 seconds. Figures 14-17 show instantaneous and total CPU utilization per VO as a function of time for different VOs and schedulers.

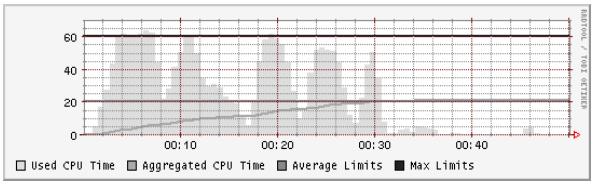

Figure 14: S-PEP with Condor (VO0)

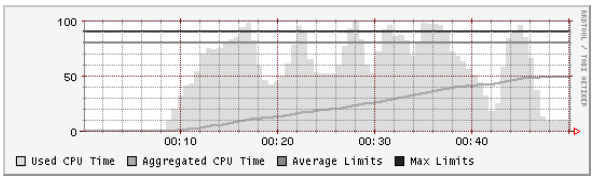

Figure 15: S-PEP with Condor (VO1)

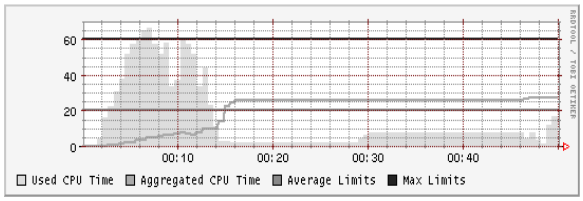

Figure 16: S-PEP with Maui/OpenPBS (VO0)

We observe that the policy enforcement module has similar effects for both schedulers. The ideal behavior cannot be achieved, due to the latencies incurred in submitting processes to site schedulers, and the subsequent scheduling delay. In addition, the monitoring sub-component achieves different behaviors, being influenced by the capacity (or in-capacity) of the tested scheduler to return job information in a timely fashion.

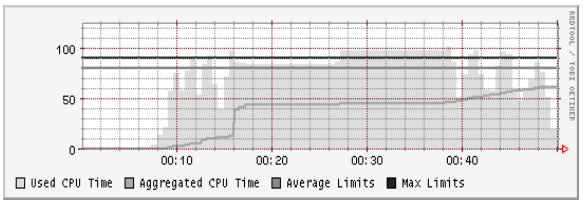

Figure 17: S-PEP with Maui/OpenPBS (VO1)

6.2. Local RM Usage Policy Enforcement

The second solution is entirely based on the local schedulers' capabilities to enforce various usage policies. Usage policies are specified as RM configuration rules; the S-POP collects and translates them to an abstract policy understandable at grid level.

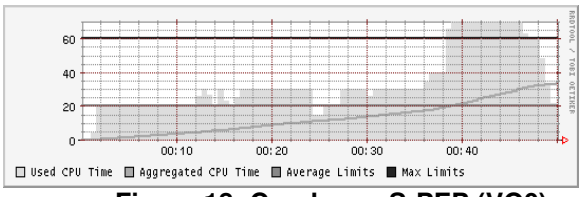

Figure 18: Condor as S-PEP (VO0)

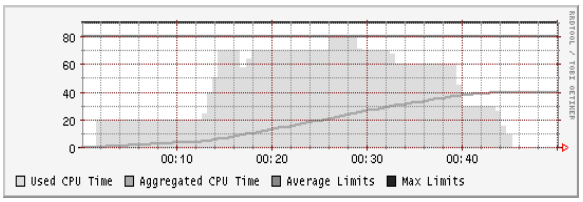

Figure 19: Condor as S-PEP (VO1)

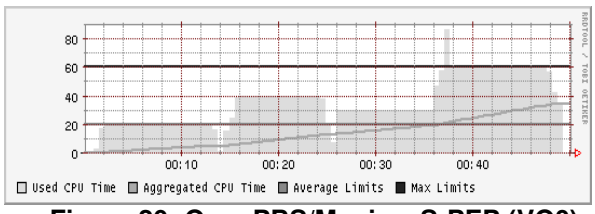

Figure 20: OpenPBS/Maui as S-PEP (VO0)

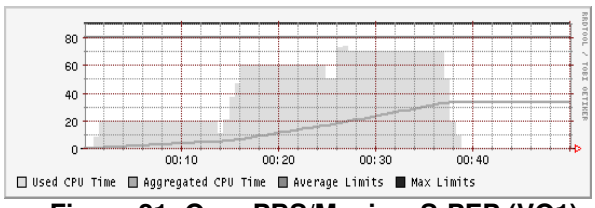

Figure 21: OpenPBS/Maui as S-PEP (VO1)

Figures 18-21 show instantaneous and total CPU utilization per VO as a function of time for the two VOs and the two different local schedulers. The usage policy is simpler in this case to accommodate Condor's capabilities in priority specifications.

6.3. Quantitative Comparison

Table 2 presents a quantitative comparison of our two enforcement solutions. We define *burst usage policy violation* (BUPV) as the ratio of the CPU-resource over-burst allocation consumed by users (BETi) to total CPUs available. We compute this quantity as follows:

$$BUPV = \Sigma (BET_i) / (\#_{cpus} * \Delta t)$$

We define the *epoch usage policy violation* (AUPV) as the ratio of the CPU-resource over-epoch allocation consumed by users (EETi) to total CPU resources available. We compute this quantity as follows:

$$EUPV = \Sigma (EET_i) / (\#_{cpus} * \Delta t)$$

We define *aggregated resource utilization* (ARU) as the ratio of the CPU-resource actually consumed by users (ETi) to the total CPU-resources available. We compute this quantity as follows:

$$ARU = \Sigma (ET_i) / (\#_{cpus} * \Delta t)$$

Table 2: Quantitative Comparison Results

Policy	BUPV	EUPV	ARU
S-PEP/Condor	0.12	0.01	70.2
S-PEP/PBS	0.10	0.08	88.5
RM/Condor	0.16	0.16	73.2
RM/PBS	0.03	0.17	67.6

These results require further validation using additional workloads. However, we see that for these particular scenarios, S-PEP achieves better enforcement of epoch usage policies than do local resource managers; for burst usage policies, there is no clear winner. We also see that PBS is slightly better at enforcing burst usage policies than is Condor.

6.4. Grid3 Usage Policy Enforcement Status

We test our assumptions in a real scenario by monitoring the site usage policies and actual resource usages over a period of two weeks at a single Grid3 site, U. Chicago. Results are captured in Figure 22. U. Chicago runs Condor, but we also see similar behaviours for other local resource managers (e.g., OpenPBS, LSF). The two VOs that compete for resources are USATLAS and Grid-Exerciser. The usage policies enforced were 35% for USATLAS and 0.1% for GRIDEX, based on Condor's extensible fair share. USATLAS's larger allocation means that whenever it has jobs queued at this site, they are immediately executed. But as soon as USTALAS load decreases (X=800, or X=1030), Grid-Exerciser's jobs take over and get all the resources they request. When the USATLAS jobs start again (X=950), Grid-Exerciser's jobs are throttled back.

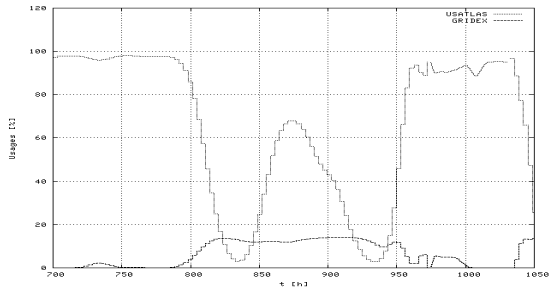

Figure 22: Allocations at U. Chicago

6.5. Grid3 Advantages

In order to evaluate the utility of taking into account usage policies, we conducted further tests on Grid3. We used a specialized resource SLA based broker [21] that uses policy specification and resource utilization information gathered by means of S-POPs. The broker creates and maintains an internal image of Grid3 and answers following queries: "*From the list L of sites, on which subset S can a specified VO run?*", and "*Which is the best site for VO's workload?*".

We define the *aggregated response time* (ART) as follows, with RTi being individual job time response:

$$ART = \bullet_{i=1..N} RT_i / N$$

We define the *aggregated job completion* (AJC) during a predefined time interval as follows:

```
AJC = Job Completed / Job Submitted
```

The results in Table 3 capture aggregated resource utilization, average response time, and aggregated job completion [14] for a workload consisting of 1000 jobs submitted to a subset of 16 sites. Each job ran on average 35 minutes, and the experiment 2.5 hours.In all cases, our broker identified feasible sites for execution. In the first two rows (NP), sites were selected based only on CPU information. In the last two rows (SC), sites were selected based on the usage policies as well. Further, we used random assignment (RA) and round robin (RR) for site selection.

Table 3: Results for the BLAST Workload

Policy	ART	ARU	AJC
RA /NP	97.01	0.27	0.54
RR /NP	114.99	0.27	0.54
RA /SC	126.43	0.35	0.69
RR /SC	130.70	0.33	0.65

We observe that the use of policy information leads to a better job completion rate, even though the response time seems to be higher. However, we consider further analysis beyond this paper's scope. We consider further analysis beyond the scope of this paper.

7. Related Work

In et al. [15] propose a system for policy based scheduling of grid-enabled resource allocations. It provides strategies for (a) controlling the request assignment to grid resources by adjusting resource usage accounts or request priorities; and (b) managing efficiently resources by assigning user quotas. The difference with our approach consists in that we do not assume a central point of policy enforcement.

Humphrey et al [22] consider a different means for usage policies. They focus on VO-wide resource distribution and how resources can be allocated based on various requirements. We look from both a consumer and provider point of view, targeting also provider goals and their resource management.

Ludwig et al. [24] propose a more generic system for SLA-based resource allocation, Cremona. Cremona is a framework for SLA specification and automated negotiations. The difference with our solution consists in the targeted level. We focus on providing an infrastructure for usage policy distributed management that can also be used by the Cremona framework for resource provisioning.

Firozabadi et al. [23] describe an alternative approach in usage policy specification. They define notions such as obligations and entitlements, and provide a theoretical foundation. Again, as for [26] comparison, our solution addresses a specific practical problem. We consider that Firozabadi et al. provide an alternative for the MAUI solution used as a starting point, but the environment conditioned our solution.

8. Conclusions and Future Work

In this paper we achieved results on several dimensions. Firstly, we were interested in providing a usage policy, which can be applied with success in real case scenarios. Secondly, we compared two cluster-wide enforcement prototypes, able to deal with several cluster management packages, and provided preliminary results comparing their effectiveness. Thirdly, we provided the design of an infrastructure for usage policy-based grid scheduling.

There are still problems not fully explored in this paper. Firstly, our analysis did not consider the case of resource *over-provisioning*, a policy that allocates 40% of CPU to VO_0 and 80% to VO_1. Secondly, we did not address here how our solution applies to other resources besides processor time, disk and network.

Acknowledgements: This work was supported in part by the NSF Information Technology Research GriPhyN project, under contract ITR-0086044. Graphs in this paper were created using the RRDtool [25].

References

1. Ranganathan, K. and I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications", in *11th International Symposium on High Performance Distributed Computing*, Edinburgh, Scotland.
2. Condor Project, *Condor-G*. 2002.
3. Altair Grid Technologies, LLC, *A Batching Queuing System,* Software Project, 2003.
4. Cluster Resources, Inc., *Maui Scheduler*, Software Project.
5. Platform Computing Corporation, *Administrator's Guide, Version 4.1*, February 2001.
6. Foster, I. et al., "End-to-End Quality of Service for High-end Applications", in *Computer Communications*, 2004.
7. S. Tuecke, et al., *Grid Service Specification*.
8. Dan, A. et al., "Connecting Client Objectives with Resource Capabilities: An Essential Component for Grid Service Management Infrastructures", in *International Conference on Service Oriented Computing*, NY, 2004.
9. Pearlman, L. et al., "A Community Authorization Service for Group Collaboration", in *3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.
10. Foster, I., et al., "The Grid2003 Production Grid: Principles and Practice", in *13th International Symposium on High Performance Distributed Computing*, 2004.
11. Avery, P. and I. Foster, "The GriPhyN Project: Towards Peta-scale Virtual Data Grids", NFS proposal, 2001.
12. Chervenak, A. et al., "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets", in *Network and Computer Applications*, 2001.
13. IBM, *WSLA Language Specification, Version 1.0.* 2003.
14. Dumitrescu, C. and I. Foster., "Usage Policy-based CPU Sharing in Virtual Organizations", in *5th International Workshop in Grid Computing*, 2004.
15. In, J., et al., "Policy Based Scheduling for Simple Quality of Service in Grid Computing", in *International Parallel & Distributed Processing Symposium (IPDPS)*. April '04.
16. RFC3060, *Policy Core Information Model, 1st Specification*
17. RFC3198, *Terminology for Policy - Based Management*.
18. Condor Project, *Condor User's Manual*, 2002.
19. Massie, M., B. Chun, and D. Culler, "The Ganglia Distributed Monitoring: Design, Implementation, and Experience", in *Parallel Computing*, May 2004.
20. Foster, I. and C. Kesselman, *Globus: A Toolkit-Based Grid Architecture*, GridBook chapter, p. 259-278.
21. Dumitrescu, C., Foster, I., Raicu, I., "A Scalability and Performance Measurements of a Usage SLA based Broker in Large Environments", *iVDGL/GriPhyN report*, 2005.
22. Firozabadi, B., et al., "A Framework for Contractual Sharing in Coalitions", in *5th International Workshop on Policies for Distributed Systems and Networks,* NY, 2004.
23. Wasson, G., Humphrey, M., "Policy and Enforcement in Virtual Organizations", in *4th International Workshop on Grid Computing,* Phoenix, 2003.
24. H. Ludwig, A. Dan, B. Kearney, "Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements", in *International Conference on Service Oriented Computing*, NY, 2004.
25. Oetiker, T., "RRDtool: A system to store and display time series", http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/.