Design and Evaluation of a Decentralized System for Grid-wide Fairshare Scheduling

Erik Elmroth and Peter Gardfjäll
Dept. of Computing Science and HPC2N
Umeå University, SE-901 87 Umeå, Sweden
{elmroth, peterg}@cs.umu.se

Abstract

This contribution presents a decentralized architecture for a Grid-wide fairshare scheduling system and demonstrates its potential in a simulated environment. The system, which preserves local site autonomy, enforces locally and globally scoped share policies, allowing local resource capacity as well as global Grid capacity to be logically divided across different groups of users. The policy model is hierarchical and subpolicy definition can be delegated so that, e.g., a VO that has been granted a resource share can partition its share across its projects, which in turn can divide their shares between project members.

There is no need for a central coordinator as policies are enforced collectively by the resource schedulers. Each local scheduler adopts a Grid-wide view on utilization in order to steer local resource utilization to not only maintain local resource shares but also to contribute to maintaining global shares across the entire set of Grid resources. Share enforcement is addressed by an algorithm that calculates simple priority values, thus simplifying integration with local schedulers, which can remain unaware of the hierarchical share policy structure.

1. Introduction

In order to achieve fairness among Virtual Organization (VO) [6] members and efficient use of provisioned resources, it is important to be able to coordinate the utilization of aggregate Grid capacity so that each user group receives its expected Quality of Service (QoS) level.

We propose a decentralized system that enforces fairshare policies on a Grid-wide scale, without sacrificing the autonomy of participating sites, whose administrators retain ultimate control over their resources. The proposed system allows VO allocation authorities to declare share policies dividing the aggregate VO capacity between its members. These share policies are then collectively enforced by the VO resources to achieve fairness on a global scale.

The share policy model is hierarchical in that shares can be recursively subdivided to form a tree of shares and subshares. A policy may contain remote policy references, which allows "mounting" of remote policy subtrees, thereby enabling resource owners to delegate subpolicy definition to different allocation authorities. This mechanism also provides VO authorities with a means to distribute VO policies and hence coordinate VO resources to enforce the same subpolicy across the entire VO.

Typically, a resource owner divides the local resource capacity between local users and different VOs, granting each a resource share. This "resource slicing" essentially results in smaller (virtual) resources being reserved to different user groups. Further partitioning of a VO share is accomplished by referencing a remote VO subpolicy, establishing how aggregate VO capacity should be divided across VO projects and users. This globally scoped VO policy is then enforced jointly by all VO resources to achieve VO-wide fairness.

We employ an enforcement mechanism that we refer to as *decentralized Grid-wide fairshare scheduling*, where local resource schedulers collectively enforce hierarchical share policies by scheduling with a Grid-wide view on utilization in order to steer local resource utilization to not only maintain local resource shares but also to contribute to maintaining global shares across the VO's entire set of Grid resources.

Our framework constitutes a tool that gives both resource owners and allocation authorities fine-grained control over their sharing arrangements. We believe that this approach captures the essence of Grid computing – global resource sharing that is subject to local resource policies governing what resources are shared, between what parties resources are shared and to what extent resources are shared [6].

The rest of the paper is organized as follows: Section 2 describes how fairshare scheduling can be carried out on a Grid-wide scale. Section 3 presents a system framework

Proceedings of the First International Conference on e-Science and Grid Computing (e-Science'05)
0-7695-2448-6/05 $20.00 © 2005 IEEE

that enforces the share policies through a simple priority calculation algorithm, outlined in Section 4. Simulation results are presented in Section 5 and Section 6 describes some related work. Finally, Section 7 concludes the paper.

2. Grid-wide fairshare scheduling

The *fairshare scheduling* [9] technique logically divides the computing capacity of a resource among its users, granting each user a target share representing the portion of delivered resource utilization that the user is entitled to. Scheduling is then performed by adjusting job priorities according to job owners' historical resource usage, in order to steer resource utilization towards the target shares. Typically, the target shares are specified in fairshare policies by the local resource administrator. The scheduler takes the target share and the historical usage of the job submitter into account when calculating job priorities.

We extend on the fairshare scheduling technique and apply it on a Grid-wide scale. Conceptually, such an extension is quite natural since users tend to view a Grid as a single, although distributed, computing resource and usage is often accounted for on the Grid as a whole. Our extension adopts a hierarchical view on fairshare policies, effectively enabling both local and global capacity to be subdivided across an arbitrary number of policy levels. Hence, our technique provides local resource administrators as well as VO allocation authorities with greater control over resource utilization, allowing them to make strong claims regarding the QoS-level each group of users can expect.

Share policies can be applied with different scope:

- *local policy* shares only apply to local resource capacity. Such policies are stipulated by the resource owner and enforcement of these policies is carried out by the resource itself to achieve *local fairness*.

- *global policy* shares apply to the entire Grid/VO capacity. Such policies are typically provided by higher-level authorities (e.g., VO allocation authorities). The resource schedulers enforce these policies in a collective effort where each local resource schedules to achieve *global fairness* using global usage data.

Hence, global policy enforcement is carried out in a decentralized manner, without the need for a central coordinator.

Share policies are hierarchically structured into a share tree where shares may be recursively subdivided into a number of sub-shares, each granted a portion of the parent share. A policy may also reference remote subpolicies, which leads to a delegation-based policy definition model where resource owners typically define a VO share that references/mounts the policy subtree of the VO allocation authority. The VO authority could in turn delegate the definition of each project policy, further dividing the project share among its members, to the project's Principal Investigator.

As an example, consider a resource owner that defines a share policy as follows:

- VO-A is granted a 50% share of the local resource capacity. This policy entry references a global subpolicy provided by VO-A's allocation authority, which further divides the VO-A share so that each of VO-A's four projects P-A1, P-A2, P-A3, and P-A4 is granted a quarter of VO-A's share. The P-A1 project share is then split equally between project members U-A11 and U-A12.

- A 25% share of the resource is allocated to VO-B. The remote subpolicy of VO-B, which is mounted at this policy entry, further divides the VO-B share between three projects in a 60:30:10 ratio.

- 25% of the resource capacity is allocated to local (non-Grid) projects. Unlike the global subpolicies of VO-A and VO-B, which apply to the capacity of all VO-provisioned resources, the local projects are only allocated a share of this particular resource.

The resulting policy is illustrated in Figure 1, which shows how the local resource capacity is divided between VO-A, VO-B and the local projects. It also demonstrates how the VO-A and VO-B shares reference global subpolicies, that apply to the aggregate capacity of all VO resources.

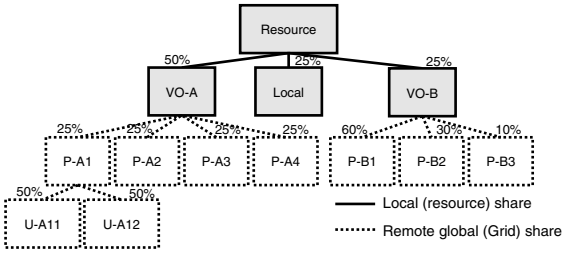

Figure 1. A local policy that references global subpolicies for two VOs.

In order to enforce this policy, the local resource scheduler strives to steer local resource utilization to:

- Maintain the *local shares* allocated to VO-A, VO-B, and local projects *on this resource*.

- Contribute to maintaining the *Grid-wide shares* allocated to the VO-A and VO-B sub-projects *across the entire set of resources available to VO-A and VO-B, respectively*.

Notably, share policies can be applied at different levels of granularity. In this example, user-level shares are only enforced within project P-A1 and there is no fairshare scheduling between local resource projects even though they, as a group, will be scheduled towards 25% of the local resource capacity.

In summary, local as well as aggregate Grid capacity is partitioned across groups of users by declaring locally and/or globally scoped share policies that are collectively enforced by the resources, which schedule with a Grid-wide view on utilization. The importance of Grid-wide usage data must be emphasized. Scheduling based on local usage data alone often fails to achieve global fairness. This is the case, e.g., when a user's workload is not evenly distributed across the Grid resources, as will be shown in the simulations.

3. System design

The share policy represents the ideal utilization of local and Grid-wide resource capacity. It defines target shares that the scheduling framework strives to maintain.

The hierarchical policy model is extensible, allowing an arbitrary number of policy levels as well as inclusion of new entity types. The root entry represents the total resource capacity, with an implicit 100% share. Each child entry represents the share allotted to a particular user group, specified as a percentage of the parent share. An entry contains the entity name (e.g. VO-A), type (e.g. VO), target share (e.g. 50%), usage data source, and a set of child policy entries.

The scope of a policy share is determined by its usage data source, which points to utilization data that the scheduler uses to determine the target share deviation of the entry. For global policies, the usage data source will refer to a service publishing Grid-wide utilization data, such as the logging service of the SweGrid Accounting System (SGAS) [14], whereas the usage data source of a local policy typically refers to a local usage database. The resource administrator may configure a protocol-specific collector for each type of usage data source. Note that the utilization metric does not need to be based on pure CPU-time but could, e.g, also account for the performance differences of different Grid resources.

Figure 2 shows the policy illustrated in Figure 1. Note, in particular, that the actual utilization corresponding to locally scoped policy entries will be determined from a local database, whereas the usage pertaining to VO policy entries is retrieved from global usage data services. Global subpolicies, such as those for VO-A and VO-B, are mounted in the local policy by specifying the addresses of policy provider services. The resource owner can configure policy fetchers that periodically follow policy references and incorporate them in the runtime policy.

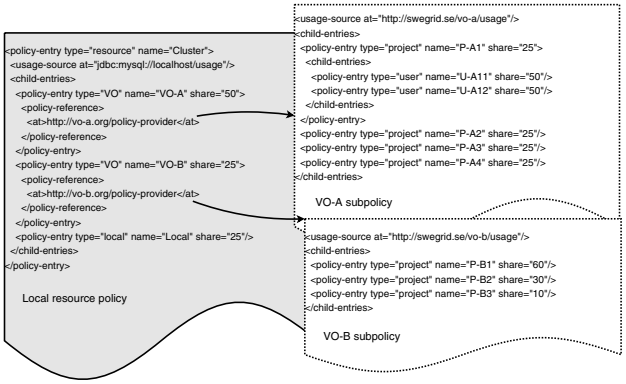

Figure 2. A resource policy that references global subpolicies for two VOs.

3.1. Scheduler integration

Figure 3 illustrates how our Grid-wide fairshare scheduling framework can be integrated with the local scheduler as a fairshare priority factor callout.

A job that is submitted to the resource is first received by the local workload manager, e.g., the Globus jobmanager [5] or the NorduGrid ARC Grid Manager [11], which may interact with an accounting system before granting the job access. A Grid accounting system [14, 15, 4] is a good complement to the soft share enforcement model of the scheduling framework, as it can provide hard usage limits by enforcing resource quotas.

The job is then handed over to the local scheduler, which calculates job priorities. During this process, the scheduler makes a callout to our scheduling framework in order to get the fairshare priority factor for the job. The Grid-wide fairshare scheduling framework calculates a job priority by comparing the actual usage of the submitter to the target share specified in the policy. The relative contribution of the fairshare factor on the overall job priority value can be controlled by adjusting a fairshare factor weight in the scheduler configuration.

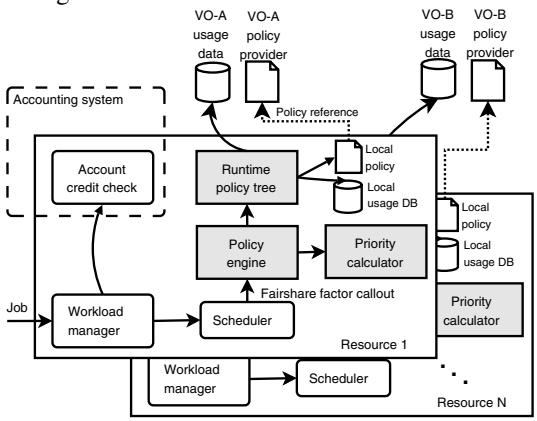

Figure 3. Framework components (shaded).

3.2. Framework components

The framework has four major responsibilities:

- Manage locally and globally scoped share policies.
- Map incoming jobs to the correct policy entry.
- Collect and manage usage histories for policy entries.
- Calculate job priorities based on target share deviation.

There are two main framework components – the *policy engine*, serving as the integration point with the local scheduler, and the *runtime policy tree*, which manages the target shares and collects usage data for policy entities.

Policy engine. The policy engine is a customizable component that calculates a fairshare priority factor for each job based on the target share and actual usage of the job submitter. The simple priority values produced by the policy engine (described in Section 4) allows the framework to be easily integrated with regular schedulers that are unaware of the hierarchical share policy structure. The policy engine can be replaced by custom implementations to integrate new schedulers or to meet the specific needs of a particular resource owner. The behavior of the engine is configurable and allows, like the fairshare parameters in Maui [10], the priority calculation to be customized in terms of usage history length (number of considered time-windows), history granularity (the duration of each window) and usage history aging (controlling the decaying impact of old usage data).

In order to calculate a fairshare priority, the policy engine needs information about the target share and actual usage of the job submitter. To acquire such information, the policy engine passes the job down the runtime policy tree, which maps job submitter credentials to their corresponding policy entities and gathers usage data about each entity.

Runtime policy tree. Each entity in the policy tree has a corresponding runtime policy entry, responsible for gathering historical usage data about its entity.

When a runtime policy entry receives a job, it collects information about its usage relative to the parent, using the usage data source of the parent. If a child entry is found that corresponds to the job submitter, the job is passed on to that child. The job then travels down the tree until it reaches a leaf node or until it cannot be mapped to a child entry.

The root-to-leaf ordered sequence of policy entities that a job passes on its way down the runtime policy tree builds a *target deviation tuple*, where each tuple element holds the difference in percentage between the target share and the actual share (i.e., the target deviation) for the policy entity it represents. The target deviation tuple serves as input to the share enforcement algorithm.

4. Share enforcement algorithm

The share enforcement algorithm, which may be replaced by custom implementations, is used by the policy engine to calculate a fairshare priority factor for each job. It strives to schedule jobs for the least favored policy entry, which can be found in a top-down policy tree traversal by selecting the entry that is farthest from receiving its target share at each tree level.

This procedure can be expressed in terms of the target deviation tuple, defined previously. By keeping the target deviation tuples of all jobs lexicographically ordered, a job tuple belonging to the least favored policy entity is located at the end of the list.

By adding 100 to each target deviation tuple element (which span from -100% to +100%), the tuple elements can be viewed as digits in a base 200 number. Hence, rather than handling tuples we can simplify the algorithm to work with decimal numbers by converting the base 200 number formed by a target deviation tuple to its corresponding decimal representation. This way, the underlying scheduler only needs to deal with simple decimal priority values.

The algorithm uses the following notation:

- $policy =$ the runtime policy tree
- $deviation =$ the job's target deviation tuple represented as a root-to-leaf ordered array
- $deviation[i] =$ the target share deviation (in percent) of the i-th tuple element

After receiving the target deviation tuple for a job, the job priority calculation is carried out as follows[1]:

$d := depth(policy)$
$deviation[i] := 0, \forall i \in \{length(deviation) + 1, \ldots, d\}$
$priority := \sum_{i=1}^{d} (deviation[i] + 100) \cdot 200^{d-i}$

The second step of the above algorithm simply pads the target deviation tuple to assure that all tuples contain the same number of elements. Picking the job with the highest priority value now corresponds to scheduling a job for the least favored policy entity.

The described algorithm has the following properties:

- It produces a single ("flat") priority value for each job.
- It performs top-down enforcement of policy shares.
- It isolates subgroups.

Subgroup isolation means that the target deviation of policy entries only affects the scheduling order among jobs

[1] For clarity, the presented algorithm is somewhat simplified, since multi-window usage histories need to be aggregated into a single utilization metric.

belonging to sibling entries in the policy tree. In our previous example, the target deviation of P-A1 only impacts the internal ordering of VO-A jobs, whereas it does not affect the scheduling order between VO-A and VO-B jobs.

5. Simulations

In this section we present a simulation-based evaluation of our framework. The simulations demonstrate the potential of Grid-wide fairshare scheduling, evaluate the accuracy of different usage history approaches and show basic properties of the hierarchical share enforcement algorithm.

5.1. Simulation setup

All simulations were carried out using GridSim [2], a Java-based, discrete-event Grid simulation toolkit. We have implemented the architecture outlined in Figure 3 by writing our own resource scheduler as an extension of the AllocPolicy GridSim class. Our simulation scheduler, which schedules in a manner similar to Maui [8], makes a callout to our scheduling framework policy engine for each queued job during a scheduling iteration. The policy engine then calculates a priority factor for the job based on the job submitter's deviation from its target share, as outlined in Section 4. Whenever a new job can be started, the scheduler selects the job with highest fairshare priority.

We simulate a SweGrid-like [17] Grid environment with six 100-CPU resources, all enforcing the same usage policy – illustrated in Figure 4. The first policy-level has local scope, granting VO-A 30% of the resource capacity and VO-B 70%. VO-A and VO-B both have global subpolicies to further divide their shares among projects (P-A1, P-A2, P-A3 and P-B1, P-B2, respectively). Project P-B1's share is also divided across its member users (U-B11, U-B12 and U-B13).

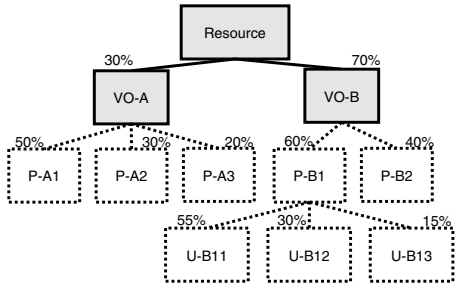

Figure 4. Simulation policy target shares.

Resource schedulers always have an up-to-date view on local usage data, whereas they maintain a locally cached view on global usage data that is refreshed once per minute. For the simulations, only a single usage history window is considered.

Each project/user contend for resources with a workload consisting of a stream of single-CPU jobs (which is the common case in Grid environments) that require an average wallclock runtime of one hour with a uniform variation of 40%. In order to maintain contention for resources users must always have a sufficient number of jobs eligible for scheduling. Hence, users submit a new job every 15 seconds. Each job is sent to a resource selected by a resource broker. Unless stated otherwise, users employ a random broker that sends jobs to each resource with equal probability. We also show what happens when users do not distribute their jobs evenly across the resources and when users do not attempt to utilize their full share.

We distinguish three different approaches for the scheduling framework to collect usage data. These are:

- *Historical usage* – only accounts for completed jobs.
- *Active usage* – includes both completed jobs and the elapsed time of currently running jobs.
- *Predictive usage* – includes completed jobs and the requested wallclock time for each currently running job.

All jobs are submitted with a requested wallclock time that is uniformly overestimated by 20%–40%, as is commonly the case according to [8]. For enforcing local policies, resources always consider local, active usage. For global policy enforcement, using global usage data, we evaluate the other approaches as well.

The effectiveness of the framework can be evaluated by comparing each entity's actual utilization share to its target share (shown in Figure 4). The actual utilization share, which we refer to as aggregated utilization, is the total, Grid-wide, utilization delivered to an entity since simulation start, represented as a percentage of the parent entity's utilization share.

5.2. Simulation results

Grid-wide fairshare scheduling correctness. Figure 5 shows the aggregated utilization of both VOs over time. This first level of the policy (local VO-A and VO-B shares of each resource) is only enforced using local usage data, and it shows close to identical numbers in all our simulations.

If we turn to global policies, Figure 6 shows how the Grid capacity utilized by VO-A is divided between VO-A projects over time, when resources schedule based on historical usage data. In Figure 7, we can see how VO-B utilization is spread over its sub-projects and, finally, Figure 8 illustrates how project P-B1's share is divided among its users.

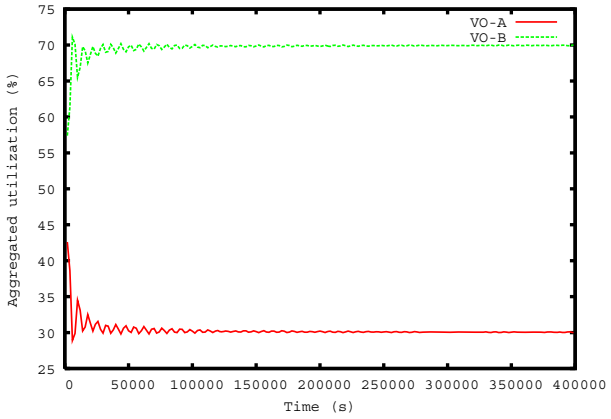

Figure 5. VO-A and VO-B actual shares of total capacity.

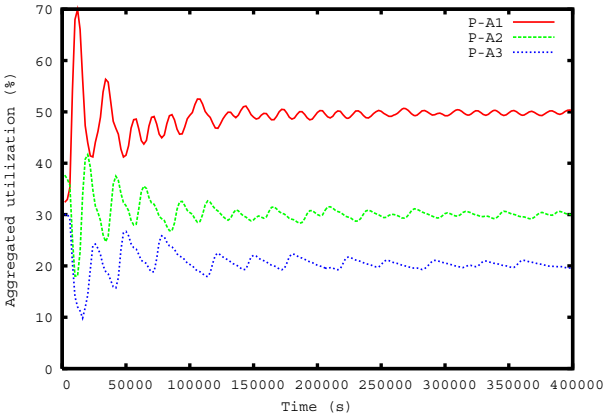

Figure 6. VO-A project shares considering historical usage data.

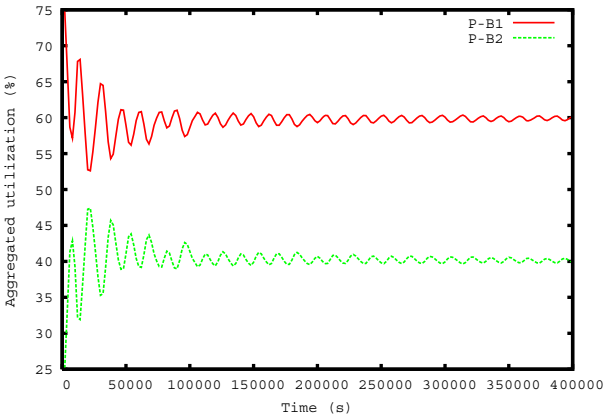

Figure 7. VO-B project shares considering historical usage data.

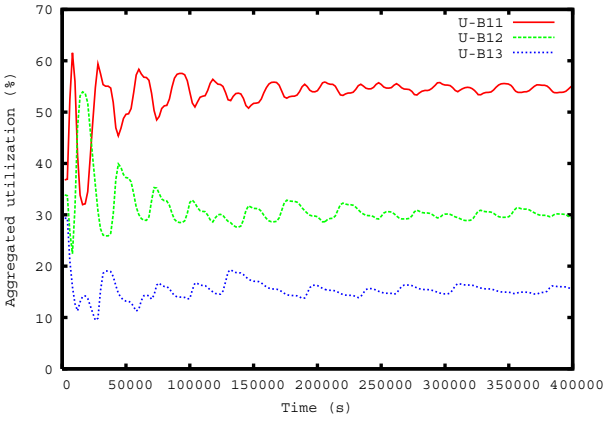

Figure 8. Project P-B1 user shares considering historical usage data.

It is clear from the figures, when comparing the actual utilization shares to those of the target policy (Figure 4), that the actual shares converge to the target shares at all policy levels in the long term. The actual utilization shares of the user groups fluctuate as the resources compensate lagging job submitters in an attempt to maintain the global shares. The target shares of deep-lying policy entries (e.g., P-B1's user shares) represent smaller absolute shares. This causes target share convergence to be slower for such entries since jobs belonging to these policy entries are scheduled less frequently and their target deviation is hence compensated for at a lower pace.

Comparing usage data approaches. The utilization of P-B1 during a similar scenario as in Figure 8, but this time scheduling based on active and predictive usage data can be seen in Figure 9 and Figure 10, respectively. In summary, the predictive usage data approach, which converges very quickly towards the target shares, produces the best scheduling accuracy.

Compared to active and predictive usage data scheduling, historical scheduling shows rather slow convergence towards the target shares. In particular, numbers tend to fluctuate a lot. This is an effect of *overcompensation*. What typically happens is that several new jobs are scheduled within a short period of time, which may cause too many jobs to be started for the least favored project. The active usage approach, and in particular the predictive usage approach, mitigate the overcompensation effect as they also account for the current and the (predicted) future usage of recently started jobs, respectively.

In summary, not limiting the view on utilization to historical data about completed jobs, but also taking the active usage of running jobs into account or even trying to predict the future usage of running jobs, can result in a significant improvement of the scheduling accuracy. Simulations show

that the predictive usage data approach performs well even when usage data is refreshed less frequently (e.g., every 5 minutes). To allow more than historical scheduling, information about active jobs, including their requested wall-clock times, and submitter credentials need to be published by the resources.

Fluctuations tend to be quite large early on in most simulations. One reason for this is that early jobs give a greater relative contribution to the total utilization. Another reason is the *first-come-first-served* treatment of arriving jobs on empty machines. Until the resource job queues fill up, all jobs will be started immediately as there is no contention for resources. Hence, projects with a small target share may receive a disproportionately large fraction of resources in the beginning of a simulation.

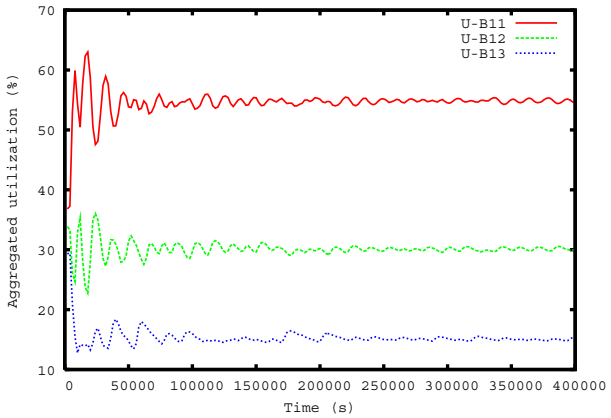

Figure 9. Project P-B1 user shares considering active usage data.

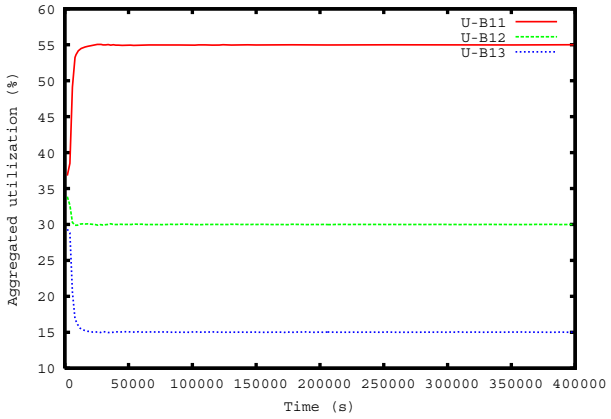

Figure 10. Project P-B1 user shares considering predictive usage data.

Imbalanced workloads. We will now consider a scenario in which project P-A2 and P-A3 only submit jobs to three of the six available resources. Such scenarios could, e.g., arise in situations where a user has very specific memory requirements that are only satisfied by a subset of resources. Of course, P-A2 and P-A3 will only reach half of their shares using the local usage data approach. This fact is illustrated in Figure 11, which is included for the sake of completeness and comparison. Notably, P-A1 is here given the shares not utilized by P-A2 and P-A3, as the schedulers strive to maintain the share allocated to VO-A. This is an effect of subgroup isolation and is further discussed below.

However, the workload imbalance is compensated for when we take global usage data into account. With a global view on utilization, the restricted entities are still able to reach their target shares, as illustrated in Figure 12. Here, it is obvious that P-A2 and P-A3 obtain their allotted 30% and 20 % shares, despite the fact that they are only submitting jobs to half of the available resources.

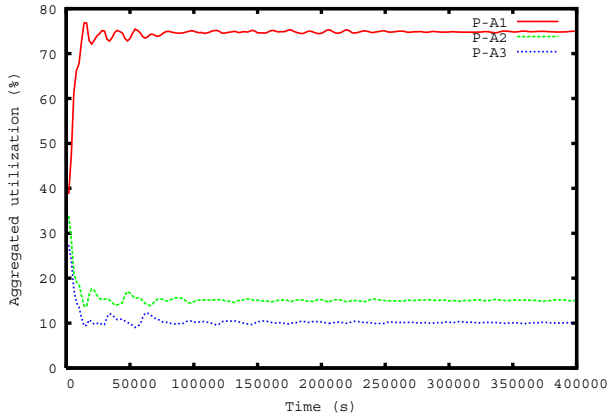

Figure 11. VO-A shares with local usage data and imbalanced P-A2 and P-A3 workloads.

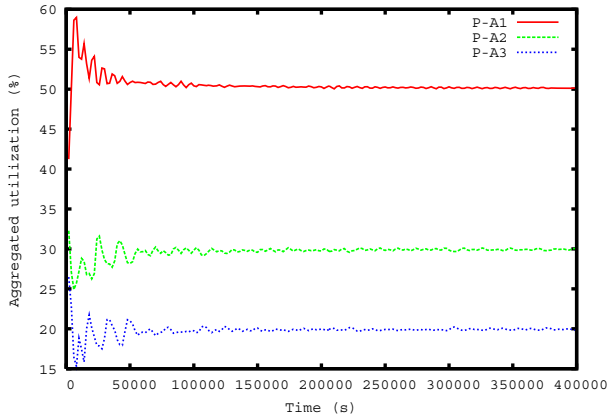

Figure 12. VO-A shares with global usage data and imbalanced P-A2 and P-A3 workloads.

To summarize, local scheduling (with local usage data only) does not cope well with imbalanced job distribution. If an entity only submits jobs to a fraction of the resources it will receive no more than that fraction of its share. Hence, local fairshare scheduling (only using local usage data) by itself does *not* achieve global fairness. By taking global resource utilization into account, the scheduling framework can compensate for job distribution imbalance.

Subgroup isolation. The algorithm presented in Section 4 performs subgroup isolation in the sense that policy subtrees are protected from each other. This means that the share of an idle entity becomes available to its active siblings. Other entities in the policy tree will not be affected.

Figure 13 shows how U-B12 runs out of work early in the simulation, causing its share to be evenly divided among its two active siblings, U-B11 and U-B13. By looking at the diagram it is clear that U-B12's share (30%) has been evenly divided across its two siblings, each given an additional 15% of the project share. As illustrated in Figure 14, P-B1 still receives its allotted share of Grid resources, despite U-B12's lack of activity. Note that these simulations make use of predictive usage data in order to make effects more easily observable, which also explains why the curves are smoother than in Figure 7.

6. Related work

Early work on fairshare scheduling was presented in [9], which mainly investigates fairshare scheduling in a single-CPU mainframe environment. It also introduces the notion of hierarchical shares.

Maui [10] is a policy-driven scheduler that determines when and where submitted jobs should be run on a computer cluster. Maui, which interfaces with several common resource managers, e.g. the Portable Batch System [12], can be customized along several policy dimensions to fine-tune its scheduling behavior. Different scheduling policies (such as fairshare, queue time and QoS-levels) are represented by priority factors and each factor is assigned a weight to control its impact on overall scheduling. Based on these factors and weights, the priority value for each job is calculated as a weighted sum. Maui performs fairshare scheduling, but it does not schedule to achieve global fairness and it is limited to single-level policies.

Sun Grid Engine (SGE) [16] is a resource manager that coordinates access to computers within an enterprise (referred to as a Campus Grid). Just like our framework, SGE enforces hierarchical share tree policies, but does this on a single-site basis, whereas our policies are Grid/VO-wide.

Grid accounting systems, such as SGAS [14], DGAS [7], Gold [13], and GridBank [1], provide a coarse-grained mechanism for achieving utilization fairness among Grid participants. Typically, each project is assigned a Grid-wide CPU-time allocation. Projects pay for their resource consumption with their allocations and may be denied resource access when their allocations have been spent. Our approach represents a softer mechanism that steers utilization towards pre-determined shares by means of adjusting job priorities, while the quota enforcement model of accounting systems can provide hard usage limits. These two mechanisms are, however, not mutually exclusive – they could be combined, as mentioned in Section 3. Furthermore, for purposes of maintaining an audit trail, most accounting systems provide some resource usage logging service, which constitutes a natural usage data source for our scheduling framework.

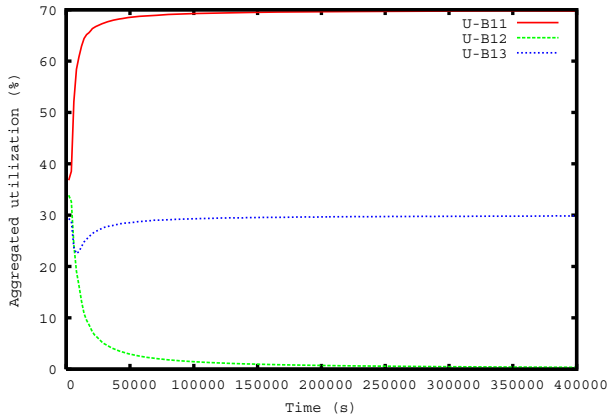

Figure 13. P-B1 user utilization when U-B12 is idle.

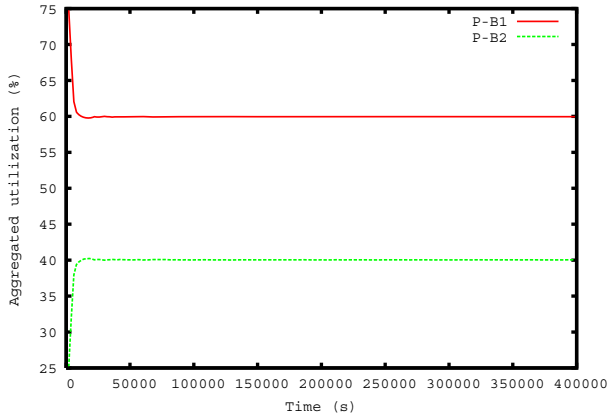

Figure 14. Total VO-B project utilization when U-B12 is idle.

CPU sharing in Grid environments is also addressed by [3] but they attack the problem using a slightly different approach. In essence, a two-level policy is established that is enforced in two stages. Resource owners grant shares of

local resources to VOs using local policies. In global policies, VOs can divide the aggregated VO resources among projects (groups) within the VO. These two-level policies are then enforced in two steps. First, job submissions are intercepted by a central VO policy enforcement point (PEP). The VO-PEP acts as a global scheduler that determines what job to run, when to run it, and what site to submit it to. The VO-PEP strives to achieve fairness among VO projects. The site then schedules jobs locally to achieve fairness between VOs. Our approaches mainly differ in that they enforce flat policies in a hierarchy of enforcement points (VO-PEPs and resources), whereas we enforce hierarchical policies in a flat enforcement structure (resources).

7 Conclusion and future work

We have presented a framework that performs fairshare scheduling on a Grid-wide scale and demonstrated its potential through simulations. The contribution of our proposed scheduling framework is fourfold:

(i) It enforces long-term shares of total Grid/VO capacity.

(ii) It enforces policies in a decentralized manner.

(iii) It manages and enforces hierarchical shares.

(iv) It includes a top-down policy enforcement algorithm that calculates simple priority values, allowing the framework to be plugged into regular "flat" schedulers.

Through simulations we were able to establish the following properties of our Grid-wide fairshare scheduling framework:

- Given sufficient user workload, Grid-wide fairshare scheduling delivers target shares in the long term.
- Scheduling with a Grid-wide view on usage data delivers target shares even if some users only utilize a subset of available resources.
- Not only using historical data of completed jobs, but also accounting for the usage of active jobs and/or trying to predict future usage drastically improves scheduling accuracy.
- The share enforcement algorithm performs subgroup isolation, so that the remaining part of an underutilized share is evenly divided among sibling policy entries.

As part of our future work we intend to implement a prototype of our Grid-wide fairshare scheduling framework that we will integrate with the Maui scheduler as a fairshare priority factor callout, as suggested in Figure 3. This would allow us to evaluate our framework in a more realistic environment, also taking the effects of other priority factors into account.

Acknowledgment

We are grateful to Åke Sandgren for technical support regarding the Maui scheduler and to Mats Nylén for sharing his experiences on fairshare (and non-fairshare) schedulers in traditional supercomputer systems. We also thank Johan Tordsson for fruitful discussions. Financial support has been received from the Swedish Research Council (VR) under contract 343-2003-953.

References

[1] A. Barmouta and R. Buyya. A Grid Accounting Services Architecture (GASA) for distributed systems sharing and integration. In *IPDPS'03*, France, 2003.

[2] R. Buyya and M. Murshed. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *ArXiv Computer Science e-prints*, pages 3019–+, Mar. 2002.

[3] C. Dumitrescu and I. Foster. Usage policy-based cpu sharing in virtual organizations. In *GRID '04*, pages 53–60, Washington, DC, USA, 2004. IEEE Computer Society.

[4] E. Elmroth, P. Gardfjäll, O. Mulmo, and T. Sandholm. An OGSA-based bank service for Grid accounting systems. In *State-of-the-art in Scientific Computing*, Lecture Notes in Computer Science, 10 pages, Berlin, 2005 (accepted). Springer-Verlag.

[5] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. J. Supercomput. Appl.*, 11(2):115–128, 1997.

[6] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200 – 222, 2001.

[7] Guarise, A. and Piro, R. and Werbrouck, A. DataGrid Accounting System - Architecture - v1.0. EU DataGrid, 2003.

[8] D. B. Jackson, Q. Snell, and M. J. Clement. Core Algorithms of the Maui Scheduler. In *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 87–102, London, UK, 2001. Springer-Verlag.

[9] J. Kay and P. Lauder. A fair share scheduler. *Commun. ACM*, 31(1):44–55, 1988.

[10] Maui Cluster Scheduler, April 2005. http://www.clusterresources.com/products/maui/.

[11] NorduGrid. http://www.nordugrid.org, April 2005.

[12] OpenPBS, April 2005. http://www.openpbs.org/.

[13] S. Jackson. The Gold Accounting and Allocation Manager, 2005. http://sss.scl.ameslab.gov/gold.shtml.

[14] T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O. Mulmo. An OGSA-based accounting system for allocation enforcement across HPC centers. In *ICSOC'04*, pages 279–288, USA, 2004. ACM.

[15] SGAS Project Page, April 2005. http://www.sgas.se.

[16] Sun Microsystems. Grid Engine Project Home, April 2005. http://gridengine.sunsource.net/.

[17] SweGrid, April 2005. http://www.swegrid.se.