Decentralized Scalable Fairshare Scheduling

Per-Olov Östberg, Daniel Espling, Erik Elmroth

Dept. Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden

Abstract

This work addresses Grid fairshare allocation policy enforcement and presents Aequus, a decentralized system for Grid-wide fairshare job prioritization. The main idea of fairshare scheduling is to prioritize users with regard to predefined resource allocation quotas. The presented system builds on three contributions; a flexible tree-based policy model that allows delegation of policy definition, a job prioritization algorithm based on local enforcement of distributed fairshare policies, and a decentralized architecture for non-intrusive integration with existing scheduling systems. The system supports organization of users in virtual organizations and divides usage policies into local and global policy components that are defined by resource owners and virtual organizations. The architecture realization is presented in detail along with an evaluation of the system behavior in an emulated environment. In the evaluation, convergence noise types (mechanisms counteracting policy allocation convergence) are characterized and quantified, and the system is demonstrated to meet scheduling objectives and perform scalably under realistic operating conditions.

Keywords: Grid scheduling, Fairshare scheduling, Fair share scheduling, Grid allocation policy enforcement

Email addresses: p-o@cs.umu.se (Per-Olov Östberg), espling@cs.umu.se (Daniel Espling), elmroth@cs.umu.se (Erik Elmroth)
1. Introduction

The core idea of fairshare scheduling is to schedule jobs with respect to what fraction of preallocated resource capacity job owners have consumed [14]. While a number of local resource site schedulers, e.g., Maui [13] and Simple Linux Utility for Resource Management (SLURM) [22], support fairshare-based job prioritization within resource centers, there is still a need for scalable distributed mechanisms that provide Virtual Organization (VO)-level fairshare support within and between resource centers in federated Grid environments. In this work we address that need and present Aequus, a decentralized fairshare job prioritization system that operates on global (Grid-wide) usage data and provides a scalable usage allocation model coupled with a distributed job prioritization mechanism.

Providing scalable fairshare support in federated Grid environments is a complex problem. Federated Grid environments are constructed through aggregation of volatile computational resources that span multiple administrative domains and utilize heterogeneous resource environments. In addition, a key requirement for Grid federation is that resource site autonomy is preserved and that local resource systems remain as unaltered as possible. Scheduling itself is in Grid environments a multi-dimensional problem where multiple components collaborate and operate on obsolete and incomplete information.

Much work on Grid infrastructure is directed towards virtualization of job and resource management, but many state of the art Grids still lack adaptability and flexibility in usage policy allocation and enforcement. This kind of rigidity can effectively restrict many of the main use cases for Grids and, e.g., force end-users to perform manual resource selection to meet quality of service requirements not supported by automated brokers. To facilitate end-user resource utilization and provide Grid environments with the flexibility required to effectively provide computational capacity as a service, policy enactment mechanisms that support Grid-level usage allocation while respecting resource site autonomy are required.

For scalability in system deployment and administration, Grid policy enactment systems should also allow delegation of policy administration, use scalable fairshare calculation algorithms, and support management of Grid-scale volumes of usage data.

The system proposed in this paper, Aequus, is designed to facilitate VO-level operation and can perform multi-scale prioritization of projects, users, and jobs simultaneously. At the core of the system lies a tree-based policy model for usage quota allocation. The system supports delegation of policy administration and dynamic mapping of VO policies at resource sites, which results in a flexible fairshare-based policy enactment mechanism suited for federated Grid environments.

Aequus defines a decentralized architecture with distributed components for hierarchical caching of usage records and local components for efficient fairshare prioritization calculations. The system is designed to allow local resource site schedulers to replace existing fairshare calculation mechanisms with call-outs to Aequus, minimizing the required changes to existing deployment environments. The main building blocks of Aequus are:

- A Grid usage policy model that supports delegated administration of resource capacity allocation (presented in Section 4).
- An algorithm for efficient calculation of job prioritization from usage data and allocation policies (presented in Section 5).
- A decentralized service-oriented architecture for dynamic fairshare job prioritization capable of non-intrusive integration with existing high-performance computing resource environments (presented in Section 6).

To characterize the proposed system, a performance evaluation of the system is performed. The evaluation utilizes an emulated system environment that assumes a general model of Grid environments built on High-Performance Computing (HPC) resource sites. In this model, jobs are fed from a batch system into a (cluster) scheduler. While this model is representative for many production Grid environments, the proposed system is not limited to use in HPC Grids. Aequus can be utilized in any system that performs execution order prioritization of jobs. The proposed system contains no functionality for advanced scheduling mechanisms, e.g., job preemption, and is to be viewed as an independent job prioritization component rather than a full policy enforcement or job scheduling mechanism.

The rest of the paper is structured as follows. Section 2 gives a brief introduction to the concepts of scheduling and fairshare job prioritization. Section 3 presents a differentiation of prior work and gives a brief survey of related work. The following sections present the building blocks of the Aequus system; a tree-based policy model (Section 4), an algorithm for efficient calculation of fairshare vectors (Section 5), and a decentralized architecture for scheduler-based Grid allocation policy enactment (Section 6). To characterize the
system. Section 7 presents a performance evaluation where the proposed system is evaluated in an emulated system environment. Finally, possible directions for future work are outlined in Section 8 and the paper is concluded in Section 9.

2. Scheduling and Scheduling Objectives

Scheduling in computing systems exists at multiple levels and have a number of goals (scheduling objectives), which may be formulated differently at each level of resource sharing. In general, scheduling problems are multi-dimensional and contain (often conflicting) scheduling objectives that are hard to meet as they originate from parties with different perspectives. For example, while resource owners are primarily interested in maintaining high resource utilization, system end-users are typically more interested in qualities of services such as short turn-around time for tasks or maximization of resource capacity shares. Here we provide a brief introduction to scheduling and the concept of fairness in scheduling.

2.1. Local and Meta Scheduling

In Grid systems, scheduling is normally done as a two-step process. Initially, a meta-scheduling decision identifies a suitable cluster to which the job is forwarded. The local, more informed, cluster scheduler then makes a decision on scheduling of the job to one or more local resource(s) for execution.

The meta-scheduling process can either be managed centrally by the Grid or in a distributed manner across several entry points to the infrastructure. Regardless of the meta-scheduling approach used, the local scheduler software is responsible for placing incoming jobs on local resources for execution (primarily) based on job priority. More information on local (cluster) scheduling can be found in, e.g., [12].

2.2. Fairness in Scheduling

There are many factors that can be used to determine the priority of available jobs, e.g., the age of the job (time spend in queue), the job size, or quality of service requirements for the job. Fairshare is another common factor, obtained by comparing historical usage of the user submitting the job to a predefined usage allocations. Jobs for users who have consumed more than their allocated resources receive a low fair-share factor, while jobs for users with remaining allocations receive a higher priority. The impact of different factors in schedulers can be configured and tuned to work towards different scheduling objectives. For a more comprehensive view of job scheduling, see e.g. Feitelson et al. [10].

Extensive use of fairshare to order jobs creates a job prioritization semantic of “least favored first”, which results in a self-balancing system where jobs are prioritized in proportion to the difference between the users target and actual usage.

2.3. Distributed Fairshare Scheduling

Cluster schedulers such as Maui [13] and SLURM [22] have built-in mechanisms for calculation of fairshare values, and means to combine this value with other factors. However, the fairshare processes in existing schedulers only consider historical usage on the same local cluster, which in the Grid case (comprising several sites) creates a different fairshare context for each site in the Grid. Usage allocations also have to be specified on a per-site basis.

Compared to local fairshare, global (Grid-wide) fairshare has additional challenges that include:

- Operation across administrative domains.
- Heterogeneity in technology, performance, availability, scheduling models, and allocation models.
- Greater usage data volumes.
- Usage and policy data updates has to be propagated to all participating sites, and each update may trigger a fairshare recalculation.
- Many different actors (site schedulers) depend on the same fairshare values simultaneously.

The system proposed in this paper is designed with the above challenges in mind.

3. Prior and Related Work

The contributions presented in this work build on earlier efforts [9], where preliminary versions of the policy model and simulations of the algorithm are presented. The main contributions of this work are a proposed architecture for realization of a decentralized system based on this algorithmic model, adaptations of the policy model and algorithm to facilitate distribution of the system, and a technical evaluation and analysis of the system. The architecture is designed for use in large scale environments, and focused on scalability through

---

1Schedulers may opt to disregard job priority to improve resource utilization using, e.g., backfilling techniques [20]
distribution and parallelization of data management and computations. Modifications of the preliminary results presented in [9] include:

- Extension of the fairshare algorithm with a framework for more fine-grained differentiation of resource consumption (fairshare operators).
- Reformulation of vector formulation and increased precision of vector elements to allow greater resolution in fairshare prioritization differentiation.
- Addition of fairshare distance measure operator to allow the configurable combination of relative and absolute calculations of fairshare distance.
- Reformulation of policy specification formats and interpretation algorithms to allow dynamic updates of allocation policies.
- Realization of the system. Prior work presented a simulation of the algorithm. This work presents a realization of a distributed system that is evaluated in an emulated environment.

A number of related work exist and brief descriptions of the most relevant ones follow. The term 'fairness' is not used consistently across the field which can lead to some confusion. The definition of fairness used in this paper was chosen to provide an intuitive mapping to the usage preallocation model - users that get allocated a certain percentage of resource capacity should also be able to consume a corresponding share. The short mention of different interpretations of fairness in related work are included to illustrate that there are more than one definition, and to reduce confusion on the topic.

The fair Share scheduler [14] introduces the concept of user-level fair resource allocation in uniprocessor sharing environments. The work addresses concepts such as fairness over time, support for different entitlements for different users, hierarchical policy structures, and sub-group isolation.

An evaluation of fair share in clusters or HPC systems is presented in [17]. Applicability of previous work on uniprocessor sharing [14] to Grids or HPC systems is analyzed and simulated using logged data for thousands of real jobs. Effects of fair share on job prioritization are found to be small in this case, partly because average system utilization is not high enough to cause enqueuing of jobs and partly because other factors (e.g. CPU requirements) are more deciding than differences in job priority.

Buyya et al. present a variation of the original Aequus resource allocation strategy in [15]. Sub-groups (such as a sub-VO) may have dedicated resource allocations that can be used in conjunction with allocation of ancestor nodes. Consumption cost is used to select which allocation to use if several suitable alternatives are available. The aim is to maximize resource utilization, and fair allocation of resources between siblings in a hierarchy is not taken into consideration. An extension that also provides fair resource sharing is presented in [16]. Node job arrival rates are assumed to be known for all nodes in the system and the problem is formulated as a waiting time minimization problem. Jobs that cannot be immediately scheduled are rejected, and as jobs arrive with an assumed Poisson distribution, minimizing waiting time affects job acceptance rate. In [16] fairness is measured by job acceptance rate for different users.

Fair Execution Time Estimation (FETE) scheduling [2] is another take at Grid fair scheduling, where jobs are scheduled according to expected completion time as if running on a time-sharing system instead of a space-sharing system. Focus of this approach is to minimize risk of missing deadlines for submitted jobs.

Another hierarchical model presented in [8, 6], is used to allot resources from different sites to VOs and from VOs to users. Each sub-allocation includes both a burst allocation and an epoch allocation to control resource consumption in short and long term. GRUBER [7], is an architecture that acts as a broker for resource usage Service Level Agreements (SLAs).

DI-GRUBER [5] extends GRUBER and adds support for distributed VO policy decision points. In these systems, VO polices are analogous to global policies in Aequus and manage suballocation of resources within a VO. In contrast to Aequus (where each site loads and enforces global policies), DI-GRUBER calls external decision points for VO policy decisions.

An evaluation of Grid resource allocation mechanisms is presented in [18]. Three different mechanisms considered are volunteer, agreement-based, and economic resource allocation. The agreement-based allocation mechanism used in the evaluation is based on earlier work ([9]), where the prototype version of the Aequus fairshare calculation algorithm was named FSGrid. The agreement based method was shown to have better overall resource utilization and suffer less degradation from high numbers of users compared to alternative approaches.

A comprehensive study on share scheduling mechanisms is presented in [3]. The study includes a thorough mathematical analysis of different strategies for share scheduling in uniprocessor, multiprocessor, and distributed systems.
Several algorithms for fair scheduling in Grids are presented in [4]. In this work, focusing on adhering to task deadlines, incoming tasks will receive a share of resources proportional to their specified needs and their priority (weight). Resource allocations are done iteratively until all resources are allocated. This work on fair scheduling operates on a lower level than the work presented in this paper, as the focus of our work is to accurately prioritize jobs rather than allocating the actual resources.

A game-theoretic approach to fair Grid resource management is presented in [21]. This work considers the case where local scheduling decisions may be taken to optimize the system from the local schedulers point of view, and evaluates consequences of different levels of local scheduler autonomy in terms of (fair) scheduling.

Fair decentralized scheduling for Desktop Grids is presented in [1]. Fairness in this case is defined as minimizing the overhead of running each task on a shared infrastructure compared to a dedicated one.

4. A Tree-Based Usage Policy Model

Grids are typically formed through joint collaborations of autonomous resource sites. The amount of resources contributed to a specific collaboration normally differs between sites, and may vary over time. Grid policy models, i.e. mechanisms for mapping user identities to resource allocations, must allow site administrators to specify resource allocations on multiple levels, e.g., between local and Grid jobs, or different Grid collaborations (e.g., VOs). As Grid user bases are usually formed as VOs, Grid policy mechanisms are required to adapt to dynamic changes in VO structure.

As illustrated in Figure 1, Aequus employs a model for specification of usage allocations in policy trees. Policy tree nodes contain tuples of VO identity strings, i.e. strings uniquely identifying a VO entity such as a user or a project, and usage share values. For example, in Figure 1 resource site RS local share policy contains preallocated usage shares for virtual organizations VO1 and VO2 and local job queue LQ. A usage share value expresses a relative usage preallocation of resource capacity within a policy group (a set of VO identities that are policy tree siblings). The user U4 allocation of 0.2 in Figure 1 is interpreted as U4 being allocated 20 percent of whatever resource capacity (e.g., monthly CPU hours) is allocated to project P1.

This model allows VOs to map internal structure directly onto policy trees, and express both organizational hierarchy (tree structure) and relationships between and within policy groups (node share values) in a single structure. There are no limitations on policy organization other than VO identities being unique within tree levels, i.e. within policy groups or projects.

Relative usage metrics (i.e. share percentages) in policy allocations have a number of benefits, including:

- Abstraction of differences in resource capacity between resources sites. Absolute metrics such as CPU hours depend on resource characteristics and result in imbalances in resource utilization that are hard to account for in fair resource scheduling. Using absolute metrics, the value of a CPU hour will vary between sites depending on machine capacity.

- Adaptability to situations where resource sites dynamically join and leave resource networks. As shares are expressed in percentages of available resource capacity, allocations are decoupled from the capacity and actual number of physical resources.

- Relative capacity allocations can be mapped to arbitrary metrics, allowing site networks to redefine the capacity metrics used dynamically (e.g., switching from CPU hours to combinations of CPU hours, network bandwidth, and storage). The actual performance metric used to measure resource consumption is irrelevant to the Aequus policy model (and also, as discussed in Section 5, irrelevant to the fairshare calculation algorithm).

- Relative usage metrics provide a direct semantic for (re)distribution of unused resource capacity.

As Aequus policy trees express relative share ratios and make no assumptions of tree structure, policy trees can be constructed from multiple sources by mounting subtrees onto leaf nodes in a policy tree (see Figure 1). Aequus makes a semantic distinction between local and global share policies. Local share policies are root policies defined by resource site administrators for individual resource clusters. Global share policies are independent policy trees defined by VOs, and are mounted onto local policy trees by resource site administrators (also illustrated in Figure 1). Local policies express what global policies to enact and relative resource allocations between them. Local share policies may have local queue components that allow site administrators to reserve resource capacity for local (non-Grid) jobs. Global policy trees express structure and allocations for VO components, e.g., groups, projects, and users.

Policy tree construction can be distributed and performed recursively. This allows resource site owners to subdivide and allocate resource site capacity shares to virtual organizations or specific projects, and these
policy actors can further subdivide and allocate their resource site capacity shares recursively within their organizations. Modifications to sub-policies of, e.g., specific projects can be done at a single location and updates are propagated to all sites without involving resource site administrators.

Mounting of policy components onto policy trees does not violate the node peer uniqueness criteria of the policy model as subtree root nodes are overwritten by policy tree nodes in the mounting process. Paths in policy trees uniquely qualify both VO identities (the bottom path node) and chains of relationships between VO identities and policy ancestors.

5. A Grid Fairshare Algorithm

Fairshare scheduling relies on prioritization of jobs with respect to consumption of resource capacity preallocations. In Aequus, job prioritization is performed through comparison of fairshare vectors, vectors of fairshare balance values calculated from paths in fairshare trees. Fairshare trees inherit structure from policy trees and are calculated from comparisons of policy trees and historical usage data. As paths in policy trees define ancestries of VO identities, comparison of fairshare vectors offer a computationally efficient way to simultaneously perform scheduling prioritization on multiple levels in policy trees.

The Aequus fairshare algorithm performs calculation of fairshare vectors in two steps. First, a fairshare tree is calculated (once per resource site, illustrated in Figure 3) from an Aequus policy tree and historical usage data. Second, fairshare vectors representing each VO identity in the system are calculated from the fairshare tree (once per VO identity, illustrated in Figure 4), and associated to jobs.

5.1. Fairshare Tree Calculation

Calculation of a fairshare tree is done in two steps. First, a usage tree is constructed by recursively (bottom-up) replacing all node values in a policy tree with a cumulative usage sum. This value is calculated as the sum of all usage data found in the usage time window for the node VO identity and the sum of all child node values. To facilitate comparison of usage and policy data, node values are normalized to [0, 1]. Normalization is performed by replacing each node value with the node’s relative share of the sum of all node values on the tree level. If no usage data is found (i.e. all sibling nodes have value zero), all sibling nodes in the tree level receive equal shares. Like in policy trees, all tree level node values sum to 1 after normalization. Normalization of the data in the usage tree effectively insulates the fairshare calculation algorithm from site-dependent absolute resource capacity metrics such as CPU hours. Usage tree construction is illustrated in Figure 2.

Second, a fairshare tree is calculated by node-wise application of a fairshare operator on the policy and usage trees (illustrated in Figure 3). Fairshare operators compare share values from policy and usage trees and quantify a distance from the current to the ideal system fairshare balance state (where all users utilize resource capacity according to policy capacity preallocations). The policy and usage trees are identical in structure, and have node values in [0, 1]. Node values in the resulting fairshare tree are in [−1, 1], and quantify a difference between policy usage preallocation and actual resource consumption (as defined by the fairshare operator used). Node value sign indicates direction (positive values underuse, negative overuse), and magnitude quantifies distance to policy-usage balance. All tree level node values in fairshare trees sum to 0. Like a policy tree contains all information required for policy enactment for a VO or a resource site, a fairshare tree contains all information required to perform fairshare prioritization of jobs on a resource site.

5.2. Fairshare Vector Calculation

Once a fairshare tree has been calculated, individual VO identity fairshare vectors are calculated (illustrated in Figure 4). As paths in fairshare trees uniquely define ancestries of VO identities, combining fairshare tree node values (top-down) along a tree path creates vec-
Figure 2: Construction of usage trees from usage data is done in two steps: 1. Raw usage trees are constructed from policy trees and historical usage data. 2. Usage trees are normalized to enable comparison to policy trees.

Figure 3: Aequus fairshare calculation. Fairshare trees are calculated by node-wise application of a fairshare distance measure operator on the policy tree and the usage tree, in the illustration the absolute fairshare operator $d_a = p - u$.

Figure 5: Aequus fairshare tree operators that contain fairshare information for hierarchies of VO identities. After vector calculation, node values ($x$) are transformed to integer elements ($y$) as

$$y = \text{floor}(\frac{x + 1}{2} \times c)$$

where

$x \in [-1, 1]$

$y \in [0, c]$

This results in non-negative integer vectors that can be serialized to strings and compared lexicographically. The value $c$ is a configurable upper limit constant determining the numerical resolution of individual vector element integer representations. The default value of this constant is set to 9999 to allow for fine-grained differentiation of vector elements.

For arithmetic comparison of vectors, where vectors are projected to one-dimensional value spaces, vectors are required to be of uniform length. Therefore, vectors are appended zero value elements until they reach maximum vector length. The maximum vector length is determined by the maximum depth of the tree and zero is chosen as pad value as it expresses policy-usage balance in fairshare trees. As illustrated in Figure 4, padding is performed prior to transformation to integer vectors in the vector extraction algorithm.

Prioritization of jobs based on fairshare vector comparison results in hierarchical ranking of VO identities. Vector comparisons express differences in policy-defined preallocations and actual resource capacity consumption on multiple policy levels. As comparison is done on job ownership VO identity level, all jobs owned by the same VO identity receive the same priority.

In this framework, fairshare scheduling can be viewed as an optimization problem where the distance from each VO identity’s usage state and the system balance axis are sought to be minimized simultaneously. By prioritizing jobs by fairshare distances, a scheduling policy of “least favored first” is enacted. The term convergence is in this context defined to refer to VO identities’ resource capacity consumptions approaching policy-defined usage preallocations over time. Conversely, any mechanism counteracting system convergence in this context is defined to be convergence noise.

5.3. Fairshare Distance Measure Operators

For comparison of individual nodes in policy and usage trees, a mechanism to quantify differences between policy preallocations and resource consumption is required. To construct a metric for comparison, Aequus defines a two-dimensional value space spanned by unit basis vectors for policy share preallocations ($p$) and resource capacity consumption ($u$). In this value space, fairshare distance operators $d = f(p, u)$ are defined to measure the distance between individual user’s policy allocations and resource consumption. As illustrated in Figure 5, fairshare distance operators form surfaces in the value space. The system balance state, where resource consumptions equal policy allocations, forms an
axis \((u = p)\) transecting the value space diametrically.

By ordering VO identities by their distance from the fairshare balance state, a prioritization order that can be used for job prioritization is established. For distance measurement, Aequus defines a fairshare operator \((d)\) constituted by an absolute \((d_a)\) and a relative \((d_r)\) component. Relative operator component influences are regulated by a configurable weight \((k)\).

\[
d = kd_a + (1 - k)d_r
\]

where

\[
d_a = p - u
\]

\[
d_r = \begin{cases} 
\left(\frac{p-u}{p}\right)^2 & \text{for } u < p \\
0 & \text{for } u = p \\
-\left(\frac{p-u}{u}\right)^2 & \text{for } u > p
\end{cases}
\]

and

\[
k, p, u \in [0, 1]
\]

While any arbitrary operator (with arbitrary value space) may be chosen for fairshare distance measurement, operator selection impacts complexity and design of the system. For example, uniform and symmetric value spaces make distance interpretation intuitive, zero distance balance points facilitates padding of fairshare vectors, and unit distance magnitude facilitates scaling of fairshare balance values. Conceptually the absolute fairshare operator can be seen as a geometrical measurement of the distance between resource consumption and policy allocations in usage credits. The relative fairshare operator expresses a ratio between resource capacity consumption and policy preallocations.

The requirement for a combined operator stems from the behavior of the individual operator components. In situations where a VO identity does not utilize allocated capacity, the absolute operator degenerates and divides
used allocations evenly among VO identity peers. In situations where no usage data is available (e.g., at start-up) the absolute operator favors users with large usage shares. In situations where zero policy allocations are assigned VO identities with reported usage, the relative operator yields a maximum distance regardless of differences in usage consumptions. The terms of the relative operator are squared to improve differentiation of users far from fairshare balance. Combining the two operator components allows Aequus to operate more robustly, and provides administrators the ability to customize the behavior of the fairshare operator.

5.4. Combining Job Prioritization Mechanisms

As defined here, fairshare scheduling implies only a prioritization order for jobs. Jobs with low fairshare values may be scheduled if there are resources available and no jobs with higher fairpriority prioritization value in queue. Jobs are by this mechanism not preempted or stalled, and fairshare scheduling is to be considered a soft scheduling mechanism. If policy fairness is more important than resource utilization, schedulers may combine fairshare prioritization with external mechanisms that, e.g., reject jobs with fairshare values below a certain threshold.

As mentioned in Section 2, schedulers may combine multiple scheduling factors to determine job priorities. In such cases, a scalar fairshare rank value computed by the Aequus algorithm can be used as a fairshare component in the linear combination. If so, the fairshare vector must be projected onto a limited value range to restrict the final prioritization value’s range, which may affect the numerical stability of fairshare prioritization. To avoid this, projection of the fairshare balance values (fairshare vector elements) to a more restricted value range may be replaced with an algorithm that assigns values to vector elements according to group-wise sort order. This will project the fairshare vector to a truncated value range, preserving vector sort order while truncating distances between vectors uniformly.

6. A Decentralized Grid Fairshare Architecture

The policy model and algorithm of sections 4 and 5 provide a mechanism for fairshare prioritization of jobs based on usage allocation and resource consumption. As usage allocation policies are constructed from distributed policy components, and the algorithm operates on usage data from multiple distributed resource sites, an architecture managing distribution of data and computations is required.

As illustrated in Figure 6, the architecture of Aequus is designed as a distributable Service-Oriented Architecture (SOA) where blocks of functionality in the Aequus fairshare algorithm are identified and exposed as services. The Aequus architecture contains three major blocks of functionality: policy administration, usage data monitoring, and fairshare vector calculation; which also constitute integration points between Aequus and the deployment environment.

To facilitate computational efficiency and reduce communication overhead of the system, a number of observations about the interaction patterns of the functionality blocks can be made. Fairshare vectors are required for job prioritization and should be recalculated whenever updated policy allocations or usage data are available. As schedulers require access to fairshare vectors whenever scheduling decisions are made, e.g., when job queues change or periodic scheduling cycle events occur, the fairshare vector calculation block should be located close to the scheduler. The policy administration and usage data monitoring blocks are by nature distributed, but should for reduction of communication overhead have a cache component close to the fairshare vector calculation block.

The computational complexities of computing fairshare trees and vectors are low, and both operations can be precomputed and results cached, making them well suited for implementation in Web Services. Calculation of the fairshare tree is performed once per resource site and scheduling step, and calculation of fairshare vectors is performed once per VO identity owning a job in the scheduling queue.

Note that the design of the system does not assume coordination of component actions, or synchronization of distributed state, but rather realizes a set of autonomous components that combined form a decentralized fairshare architecture. Global fairshare resource allocation is enacted through concurrent, asynchronous local computations on distributed data.

To minimize the system deployment footprint, all services are designed to integrate non-intrusively with existing infrastructure and minimize network traffic required by the system. Service deployment patterns are expected to vary from site to site, but are recommended to be based on the pattern illustrated in Figure 6 to minimize communication overhead.

6.1. Architecture Components

As illustrated in Figure 6, Aequus is constituted by five services and a set of plug-ins for scheduler prioritization, usage data submission, and identity resolution. To facilitate seamless integration into existing
HPC deployments, the architecture is implemented in Java and exposes service functionality through WSDL SOAP Web Services deployed in Apache Axis2 service containers. Integration with HPC cluster schedulers (currently Maui and SLURM) is done through injection of Aequus clients into scheduler exposed prioritization customization points.

6.1.1. Policy Distribution Service (PDS)

The Policy Distribution Service (PDS) provides a service interface to Aequus usage policy allocations. Internally, the PDS collates policy components from multiple sources, e.g., XML files, HTTP web resources, other PDSs; assembles a policy tree; and publishes policies through the service interface. As multiple PDS may be chained, and data read remotely, the PDS provides a flexible mechanism for delegating policy definition to VO and site administrators. To Aequus and Aequus clients, the PDS provides an easy to use interface for policy retrieval, and can be to, e.g., monitor updates in policy allocations.

6.1.2. Usage Statistics Service (USS)

The Usage Statistics Service (USS) is designed to provide time-resolved histograms of usage data on a per-user basis. To reduce the amount of data, the service interface accepts updates in a format semantically equivalent to summaries of Open Grid Forum (OGF) Usage Records [19], and exposes usage summaries for requested time windows. Internally, the USS stores usage histograms for known users in a database, and maintains a usage summary cache to minimize invocation response time. The USS is the only required part of Aequus that receives input data from the surrounding environment. As usage data constitutes the currency that drives Aequus fairshare, it is vital to Aequus system coherency that each job usage record is only reported to a single USS. As the USS provides a histogram-based view of historical usage data, it can be used by Aequus services and clients to assess usage statistics for individual VO identities on individual resource sites.

6.1.3. Usage Monitoring Service (UMS)

The main task of the Usage Monitoring Service (UMS) is to provide a service interface for computation of (normalized) usage trees from policy trees. Internally the UMS compiles data from a set of known USSs, maintains a database of USS usage summaries, a time-resolved per-user usage cache, a cache of previously known policy trees, and agents to monitor USSs and precompute usage trees. The UMS also maintains a customization point for moderation of usage data influence through a time window and usage decay function plug-in. The UMS provides an interface for summarizing usage records from multiple (USS) data sources and mapping these to (provided) usage policies, and can be used by Aequus services and clients to get normalized usage data views.

6.1.4. Identity Resolution Service (IRS)

Key to enabling fairshare scheduling of jobs in Aequus is to be able to access historical usage records for VO identities. As VO identities may be translated to local cluster or site users when jobs are dispatched to batch queues, schedulers may lack access to VO identities. The IRS exposes an interface for storing and accessing VO identity job associations, and is primarily used to resolve job ownerships. Use of the IRS in Aequus is optional. If a scheduler has access to VO identity job ownership data these may be used directly when requesting scheduling prioritization information, otherwise this data can be submitted to the IRS service at any time prior to invocation of the FCS. Submission is typically expected to be done by the system responsible for translation of VO identities to local resource site users, e.g., a batch system.
6.1.5. Fairshare Calculation Service (FCS)

The Fairshare Calculation Service (FCS) offers a flexible service interface that provides access to the Aequus policy-based fairshare tree, fairshare vectors for specified VO identities (or jobs), and preformatted fairshare tuples that contain VO identities, fairshare vectors, and scalar fairshare prioritization values. The rich interface of the FCS is designed to facilitate flexibility in implementation of scheduler integration plug-ins. Internally, the FCS maintains caches for job identifier to VO identity maps, policy, usage, and fairshare trees, as well as agents for monitoring services (PDSs and UMSs) and precomputing fairshare trees. The FCS allows configuration of UMS and PDS connections, PDS deployments, and monitoring scheduling intervals.

6.1.6. Integration Plug-Ins

In addition to the services of Aequus, a set of integration plug-ins is also considered part of the Aequus architecture. Depending on the Aequus deployment environment, integration plug-ins for scheduler job prioritization, usage data submission, and VO identity resolution may be required. Design of scheduler plug-ins depend on scheduler architecture, but typically consist of an FCS client implemented in the same language as the scheduler and possibly routines for calculation, transformation, and caching of fairshare prioritization data. Design of plug-ins for usage data submission and VO identity resolution depend on accounting system and scheduler architecture, and will typically consist of USS and IRS clients.

As many Grid computing environments build on existing HPC deployments, which typically are required to maintain HPC interfaces (e.g., batch systems) in coexistence with Grid interfaces, it is vital to design Grid systems to impose a minimum intrusion level when integrating Grid components with existing HPC deployments. The Aequus architecture is designed to have as few and simple integration points as possible while still maintaining compatibility with a general model for HPC deployment based Grid environments.

Typical Grid Aequus integrations include:

- Replacement of scheduler (fairshare) job prioritization mechanisms with FCS invocation clients.
- Injection of mechanisms for submission of usage data to a USS. This can be done in multiple ways, e.g., through a scheduler job monitoring plug-in, or a resource site or Grid accounting system.
- Optional injection of job ownership resolution components. If VO identity job ownership data is not available to the scheduler, a job ownership mapping between job and original VO identity can be stored in the IRS.

6.2. (Concurrency in) Data and Control Flow

As illustrated in Figure 7, the data and control flows of a typical Aequus deployment consist of five autonomous and concurrent processes:

1. A set of PDSs monitors a set of data sources and periodically compiles policy trees. Local policy component data sources are administrated by VO and project administrators that specify resource capacity allocations within their domains.
2. A set of USSs receives (summarized) usage reports for jobs and builds time-resolved usage histograms. Usage reports submitted to the USS contain usage cost metrics converted from site-specific job record measurements. Conversion of usage report data is the responsibility of accounting systems and external to the Aequus system.
3. A UMS monitors a set of (local or remote) USSs, periodically retrieves updates, and assembles usage summaries. The UMS precomputes usage trees for known policy trees, and on demand for unknown policy trees (which are added to the cache structure).
4. An IRS receives VO identity job ownership data and maintains a directory for ownership resolution.
5. An FCS monitors a PDS and periodically retrieves policy trees and calculates usage (via a UMS) and fairshare trees. The FCS maintains a cache of precomputed fairshare vectors (based on precomputed fairshare trees), and does not compute fairshare vectors for unknown VO identities.

The data required to drive the system, usage data and usage policy allocations, are provided by accounting systems and VO, project, and resource site administrators respectively. Should resource sites utilize Aequus to prioritize jobs without reporting usage data, resource consumption costs for jobs running on such sites do not contribute to fairshare calculation results and imbalances in global resource consumption may occur. Conversely, should resource sites report usage data without utilizing Aequus as a job prioritization mechanism, global fairshare convergence will suffer oscillations correlated in size to resource site capacity. As the core balancing mechanism of Aequus is self-adjusting, global fairshare balance will converge over time.

Specification of usage policies can be seen to be a largely manual process, while usage data submissions
Figure 7: An Aequus system deployment scenario. Local usage data is published to global USS services. Global share policies are distributed via PDS services and mounted onto local share policies.

Figure 8: Usage data histogram time window. Usage decay functions modulate influence of usage data.

are expected to be fully automated. Through these five processes, the Aequus system provides an automated, decentralized, and self-adjusting mechanism for Grid-wide fairshare enactment of usage policy allocations.

6.3. Time Window and Decay Function

As illustrated in Figure 8, Aequus defines a finite usage data time window (typically a configurable amount of days into the past) to restrict the influence of historical usage data on the fairshare mechanism. As also illustrated, Aequus employs a customizable usage decay function to modulate how usage statistics influence the fairshare mechanism. The time window width limits the scope of usage statistics influence (data outside the time window does not affect Aequus behavior). The granularity of the time window histogram slots affect the resolution of the fairshare mechanism. The usage decay function modulates usage statistics by, e.g., increasing or decreasing influence of more recent usage statistics on system behavior. In the Aequus architecture, both time window parameters are configurable, and the usage decay function is exposed as a customization point in the UMS. Further study of the impact of usage decay functions in this context is subject for future work.

7. Evaluation

To evaluate core system functionality and isolate noise sources (i.e. mechanisms counteracting system convergence), a number of tests designed to quantify aspects of Aequus’ technical performance are employed. Tests are run in an emulated system environment and designed to introduce and illustrate system mechanics. As the purpose of this evaluation is to evaluate system ability to enact policy allocations in a distributed environment rather than demonstrate system integration in a production deployment, use of an emulated system environment is sufficient. The following tests are performed:

- Noise characterization tests (Section 7.1). Investigate and characterize Aequus noise mechanisms.
- Noise interaction tests (Section 7.2). Investigate interaction between different noise types and illustrate impact of system deployment patterns on system performance.
- Scheduling objective tests (Section 7.3). Investigate Aequus ability to enact policy allocations in decentralized multi-site deployments employing multiple asynchronized concurrent schedulers. Quantify and evaluate Aequus ability to adapt to
dynamic changes in policy allocations and distributed system failures.

- Robustness and scalability tests (Section 7.4). Evaluate the robustness of Aequus convergence and investigate system ability to cope with realistically sized policy allocations in the presence of large amounts of usage data and updates.

- Dynamic job workload tests (Section 7.5). Illustrate the behavior of Aequus when subjected to the dynamic job arrival models typical of production systems.

Evaluation tests are performed on three sets of machines. The first set is comprised of four identical 1.8 GHz quad core AMD Opteron CPU, 4 GB RAM machines, interconnected using a Gigabit Ethernet network. The second set is comprised of four identical 2 GHz AMD Opteron CPU, 2 GB RAM machines, interconnected with a 100 Mbps Ethernet network. The third set is comprised of four identical 2x16 core 2.1 GHz AMD Opteron CPU, 56GB RAM machines, interconnected using a Gigabit Ethernet network. All machines are running Ubuntu or Debian Linux and Axis2 1.5. The Java version used in tests is 1.6, and Java memory allocation pools range from 512 MB to 1 GB in size.

Functionality tests are performed using a discrete-event simulator emulating an execution environment consisting of a batch system, a cluster scheduler, a cluster, and an accounting system. Each batch system registers (in the IRS) and feeds the scheduler sets of jobs. The scheduler invokes the FCS to prioritize jobs and allocates them to cluster hosts. The accounting system submits usage reports to the USS upon job completions. Job start and end timestamps are used to evaluate Aequus ability to enact resource capacity allocations.

For functionality tests (sections 7.1-7.4), batch system emulators saturate scheduling queues so that schedulers have access to at least one job for each usage policy VO identity at all times. Conceptually, this can be viewed as a job arrival model with infinite amounts of jobs for all users. For tests of realistic job arrival models (Section 7.5), the job arrival queues are based on workload traces from production Grid environments. In tests, single-site system emulations are run on a single host, multi-site system emulations as a set of non-communicating systems run concurrently on multiple hosts. Cross-site synchronization is performed exclusively by UMSs, which have access to USSs for all sites, emulating a distributed Grid configuration.

All tests are, unless stated otherwise, run using identical parameter sets and the policy tree illustrated in Figure 1. USS and UMS update intervals are set to 1 second, usage time windows are 10 slots wide and set to a granularity of 1 day (system wall clock time), all clusters have a single host, and job lengths are either fixed to 1 or stochastic and uniformly distributed between 1 and 5000 time units long. To eliminate them as parameters in measurements, absolute and relative fairshare operators are equally weighted (k = 0.5). Usage decay is disabled (i.e. usage decay function is constant y = 1), and the usage cost metric used is job length (CPU time).

Job failures do not affect Aequus convergence rates as failed jobs do not get reported to the accounting system and appear as diminished resource capacity.

7.1. Noise Characterization

As we refer to system ability to over time enact policy-defined resource capacity allocations as system convergence (to policies), we define any mechanism countering this process as convergence noise. To illustrate system convergence to policy allocations, we isolate policy (sub)groups, i.e. groups of nodes with a common parent, and render cumulative resource consumption for individual VO identities as a function of number of jobs run in the group. To maximize the influence of noise in measurements, we isolate the policy subgroup containing the VO identity with the lowest total usage share (P1 in Figure 1).

In Aequus, there are two primary mechanisms countering system convergence, variance in job usage costs and usage data update latencies. To isolate impact of variance in job usage cost, we emulate a single-site Aequus deployment with stochastic job usage costs drawn from a uniform [1,5000] probability distribution. To eliminate impact of usage data update latencies on system convergence, each scheduling step is delayed to allow usage data updates from prior jobs to propagate to the FCS between scheduling steps. As illustrated in Figure 9a, usage cost variance amplifies oscillations in system convergence. When compared to ideal convergence, differences in usage costs manifest as additive noise in convergence adjustments. In Figure 9a, job usage cost variance noise is illustrated as vertical offsets in convergence oscillations.

To isolate impact of usage data update latencies, we emulate a single-site deployment with uniform job usage cost (cost = 1) and UMS and FCS update delays designed to allow approximately 10 jobs to be scheduled between FCS usage data updates. As the Aequus job prioritization mechanism operates on usage data for completed jobs, i.e. has no memory for recent scheduling decisions or prediction mechanism for costs of running jobs, usage data update latencies result in multi-
ple subsequent scheduling decisions being taken on the same usage data. As illustrated in Figure 9b, this results in amplifying convergence oscillations and significantly lowering system convergence rate. When compared to ideal convergence, usage data update latencies manifest as multiplicative noise in convergence adjustments and a divisible reduction in convergence rate.

Prior work [9] suggests that including cost for scheduled and running jobs in prioritization calculations reduce impact of usage data update latency noise on system performance. Future work includes evaluation of different strategies for inclusion of this approach in multi-site Aequus deployments.

7.2. Noise Interaction

Under realistic Aequus operational settings both job usage cost variance and usage data update latencies are likely to be present. To investigate noise interaction, we emulate a single-site Aequus deployment with stochastic job lengths and usage update latencies. As illustrated in Figure 10a, usage data update latencies add a multiplicative component to usage cost variance noise. Impact of noise is amplified by lowered convergence rate.

To evaluate noise interaction in decentralized Grid environments, we emulate a four-site Aequus deployment with stochastic job lengths and usage data update latencies. As illustrated in Figure 10b, parallelism of concurrent scheduling amplifies update latency noise, in this experiment delaying system convergence by a factor of 10. As the number of jobs scheduled between usage data updates determine impact of update latency noise, large numbers of computational resources per scheduler skew system convergence at startup. As job lengths constitute lower bounds for usage data update latencies, excessive job lengths amplify update latency noise. For multi-site settings, concurrent scheduling with synchronized update schedules amplify update latency noise. Conversely, asynchronicity in multi-site update schedules allow parallel processing of usage updates to increase update frequencies and mediate impact of usage data update latency noise.

These experiments are run in an artificial environment, but outline a few interactions between mechanisms in the Aequus fairshare job prioritization system. Scheduling jobs after the principle of “least favored first” creates a self-adjusting system that over time distributes resource capacity after policy allocations. Noise from job usage cost variance and update latencies lower system rate of convergence by affecting the convergence adjustments (i.e. order in which jobs are run). Number of hosts, sites, job lengths, as well as frequency and synchronicity of usage data update schedules may serve to amplify convergence noise. Over time, relative impact of each noise source and type lessen, as more usage data affect scheduling prioritization. As long as usage data time windows are large enough to contain enough data for the system to converge, the system remains stable.

7.3. Scheduling Objectives

To evaluate system ability to enforce scheduling objectives (i.e. enforce policy allocations) in the presence of dynamic changes in site availability or policy specifications, we emulate an eight-site Aequus deployment with stochastic job lengths and usage data update latencies over an extended period of time. To study impact of site volatility, four sites are removed after approximately 25000 jobs are scheduled. After approximately 50000 jobs are scheduled, the local allocation policy RS is altered to transfer 10 percent from each of the allocations for VO1 and LQ to VO2.
(a) Interaction of usage cost variance and update latency noise. Impact of usage cost variance noise amplified by delayed convergence.

(b) Concurrent scheduler noise augmentation. Impact of usage data update latency noise amplified by parallelism in scheduling.

Figure 10: Noise interaction. Cumulative resource consumption as function of scheduled and run jobs. Convergence to usage policy allocations for VO identities designated in legend. Illustration capped to region of interest.

(a) Adaptation to site failures and updated usage policy allocations.

(b) Adaptation to site failures and subgroup isolation.

Figure 11: Policy enactment. Cumulative resource consumption as function of scheduled and run jobs. Convergence to usage policy allocations for VO identities designated in legend. Illustration capped to region of interest.

As illustrated in Figure 11a, eight concurrent schedulers cause significant initial convergence noise. At approximately 25000 jobs four schedulers are removed, and convergence noise is reduced (also visible at approximately 10000 jobs in Figure 11b). At approximately 50000 jobs the usage policies are updated and all schedulers adapt to new scheduling priorities. It takes approximately the same amount of jobs currently in the time window to reach the level of convergence achieved before the policy shift. As also illustrated in Figure 11a, convergence rate is a function of the relative share ratio, the VO identity with the lowest policy allocation \((LQ)\) converges slowest.

As illustrated in Figure 11b, which illustrates policy group \(P1\) of Figure 11a, altering the policy allocation of an individual policy group does not affect other groups in the same policy tree. Note that this simulation contains multiple shifts of the usage data time window, which do not visibly affect system convergence.

7.4. Robustness and Scalability

To evaluate Aequus ability to function in production environments, we run large scale tests to evaluate the robustness and scalability of the system mechanisms. System robustness is tested by repeating an experiment a large amount of times and statistically evaluating the system convergence rates, and scalability is evaluated in tests where the system is evaluated over longer periods of time using realistically sized policy allocations and system configurations. System convergence scalability is validated for tests using policy allocations with thousands of users running millions of jobs.

In robustness tests, an experiment using the policy tree of Figure 1 and the noise models of Figure 9 is repeated a large number of times and the convergence process is evaluated for robustness. In tests, jobs costs are drawn from the uniform rectangular distribution \([1, 5000]\), resulting in convergence adjustments that can vary in magnitude with more than three orders of magni-
Figure 12: Robustness and scalability of the Aequus fairshare system. Illustrations capped to regions of interest.

7.5. Dynamic Job Arrival Models

To illustrate the behavior and scalability of the fairshare calculation model, the prior tests have all assumed a job arrival model that saturates schedulers. In this section, the behavior of Aequus under realistic job arrival conditions is verified in two ways: by executing tests using a controlled dynamic job arrival model and by executing a set of production system workload traces from the Grid Workloads Archive [11].

In the controlled dynamic job arrival model, jobs from different policy groups are made available or unavailable to the scheduler at different stages of the execution. This case verifies the system behavior during periods where no jobs for one or more policy groups are available, and demonstrates the system’s ability to reach usage convergence once these policy groups start submitting jobs. Figure 13a shows the availability of jobs from each policy group during different phases of the test run. Also shown in Figure 13a are phases of adjustment during which the system converges to a balanced state in response to jobs from previously inactive policy groups being available. The total resource consumption of each group is shown in Figure13b. As before, the target allocations for $VO_1$, $LQ$, and $VO_2$ are 50%, 20% and 30%, respectively.

During execution of the first 500 jobs, only users from $LQ$ have jobs available and $LQ$ is therefore allowed to utilize all available resources. After 500 jobs, users from $VO_2$ start submitting jobs and the usage is balanced out between $LQ$ and $VO_2$ according to their respective shares. When 1500 jobs have been executed, users from $VO_1$ submit jobs until 3500 jobs have completed. During this time, the relative usage for $VO_1$ is drastically increased and the combined usage of all policy group converge to the target values. Between 3500 and 4500 jobs $LQ$ and $VO_2$ will get an increased share.
of resources as there are no jobs for VO1 available. After 4500 jobs have completed, jobs from VO1 are available for scheduling and the combined usage converges toward the target shares of each policy group.

The results and job arrival patterns of a production system workload (NorduGrid/GWA-T-3) are illustrated in Figure 14. The trace is comprised of 781,370 jobs submitted by 387 users divided into 107 groups, being executed across 2,000 resources. The figure shows the relative resource consumption of user groups at the top level of the tree. To improve readability, the user groups of the ten most active users are plotted separately and remaining users are bundled in a unified entry labeled others. The ten most active users represent eight user groups and 70.2% of the total resource consumption.

Even though workload traces are useful to verify system functionality, they are non-intuitive when interpreting the behavior of fairshare mechanisms. For example, it is not possible to determine if, and to what degree, fairshare was used when the workload was executed. Without this information it is not possible to accurately determine capacity allocations for users in the trace. Workloads also typically contain a wide variation in job arrival patterns for different users that may offset the convergence of the system. For example, when all jobs available for execution belong to users who have already consumed their allocated resource capacity (a realistic scenario), continued scheduling and execution of jobs will counteract system convergence. Factors such as these make it hard to isolate the specific behavior of the fairshare mechanism.

Figure 14a illustrates the relative usage of different user groups at different stages through execution of the workload. The job arrival pattern of the workload trace, as shown in Figure 14b, is bursty and contain phases of high activity for individual users. The selected workload trace contains periods of job starvation that constitute worst case scenarios for fairshare convergence. Comparison of the workload execution results and the job arrival patterns illustrate that:

- Initially in the workload trace, the combined others group is the only one submitting jobs, which causes this group to consume a large portion of its allocated resource capacity early on.
- Users in group G2 submit a large set of jobs (roughly 70,000) around the 130,000 job mark. The fairshare balances are stable until roughly 150,000 jobs are executed, at which point the only jobs to be scheduled belong to G2, which causes the balances to diverge.
- At roughly the 230,000 job mark users in group G68 start submitting jobs and are granted resources quickly until about 300,000 jobs are executed.
- The balances are stable until roughly 420,000 jobs are submitted, when users in group G69 submit a set of large jobs. These users are granted resources and group G69 quickly approaches its target usage share. From this point on, the relative usages converges towards the target shares.

As no fairshare policy data is available for workload traces, artificial policies are constructed from the actual resource consumption rates of the workload traces and used in tests. Note that policy allocations are constructed based on amounts of resource capacity consumed, which (due to differences in job lengths) not necessarily corresponds to number of jobs submitted. As the system converges over time, the relative resource allocations of the selected workload can be seen in the right-most points of the graph in Figure 14a.
While workload traces make interpretation of fairshare behavior non-intuitive, they represent actual system behaviors for production Grids and are as such important for validation of the Aequus system behavior. The tests illustrated in this section demonstrate that Aequus performs robustly even under the worst case job starvation scenarios that the selected workload contains. The self-balancing fairshare prioritization mechanism of the system recovers from job starvation and adapts to dynamic job arrival patterns even when the system convergence is skewed. As illustrated by the tests, the rate of convergence of the system is naturally limited by the amount of jobs available to schedule.

8. Future Work

The system evaluation of Section 7 demonstrates technical aspects of the proposed system and shows how Grid-wide fairshare job prioritization can be realized. While this evaluation is performed in an emulated environment, the evaluation demonstrates key aspects of system functionality, scalability of system mechanisms, and system ability to enact policy usage allocations. Full-scale testing and evaluation of the system in production environments is subject for future work.

Further investigation of trade-offs between Aequus convergence parameters is expected to increase understanding of system behavior. Evaluation of impact of update latencies, usage decay functions, and job scheduling patterns are likely to influence parameterization and further development of the system. Evaluation of experiences from integration of the system in production use Grid deployments are expected to be of interest for further development of the system. Incorporation of scheduled and running jobs in fairshare job prioritization is likely to reduce impact of usage data update latency noise. Integration of the Aequus job prioritization mechanism with additional cluster scheduling systems is expected to be of interest for system adoption.

9. Conclusion

In this work we present Aequus, a decentralized system for fairshare-based Grid usage policy enactment built on three main contributions: a flexible policy model, a scalable fairshare calculation algorithm, and a decentralized architecture for parallelized fairshare prioritization of jobs. The system design is presented in detail, along with a performance evaluation and a discussion of the system.

The policy model supports mapping of VO structures onto policies, delegation of policy specification, and virtualization of usage credits. The fairshare calculation algorithm is self-adjusting and noise-stable, virtualizes resource site capacity, provides subgroup isolation within policy allocations, and adapts to changes in usage data and policy allocations. The architecture of the system is designed to facilitate decentralization of system deployments, precomputation and caching of scheduling data, and integrates non-intrusively with existing scheduling systems. The presented system can be utilized for job prioritization and scheduler-based policy enactment in Grid and HPC environments.

The performance evaluation illustrates the Aequus policy allocation mechanism, demonstrates the feasibility of the approach, and identifies factors that manifest as noise in system convergence. The evaluation investigates trade-offs between convergence noise factors, and suggests how impact of these factors may be reduced. Grid Workloads are used to verify system functionality and also to study use of bursty and dynamic job arrival models. The discussion relates the proposed system to similar work and systems, and outlines the role of the system in Grid environments.
Acknowledgments

The authors extend their gratitude to Peter Gardfjäll for prior work, to Lars Karlsson and Lars Larsson for feedback and discussions, and to Raphaela Bieber-Bardt for work related to the project. The authors also acknowledge Tomas Ögren and Åke Sandgren for technical assistance to the project.

This work has been done in collaboration with the High Performance Computing Center North (HPC2N) and has been funded in part by the Swedish Research Council (VR) under Contract 621-2005-3667, the Swedish National Infrastructure for Computing (SNIC), and the Swedish Government’s strategic research project eSSENCE. The authors also acknowledge the Lawrence Berkeley National Laboratory (LBNL) for supporting the project under U.S. Department of Energy Contract DE-AC02-05CH11231.

References


