Integration and Evaluation of Decentralized Fairshare Prioritization (Aequus)

Daniel Espling, Per-Olov Östberg, and Erik Elmroth
Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden

Abstract—Fairshare is commonly one of the factors used by cluster resource management systems to prioritize jobs during scheduling. Despite the grid vision of a transparent and unified infrastructure, fairshare is normally calculated and enforced at the local cluster level rather than at a grid-wide scale. Aequus is a self-contained decentralized system for grid-wide fairshare job prioritization. Using Aequus, detailed global share policies can be combined with local cluster policies to offer a unified grid fairshare prioritization system where local administrations retain control over their clusters. This work shows how Aequus can be integrated with local resource management systems such as SLURM and Maui with minimal intrusion. Early results from production use are presented, and the system is further tested and evaluated for use at a nation-wide scale. Statistical models are created based on historical national grid usage data, and synthetic traces based on these models are used to create a diverse input set used to exemplify system behavior. The system is shown to behave consistently despite great variations in job arrival patterns and partial participation of some of the collaborating installations.

Index Terms—Grid scheduling, Fairshare scheduling, Workload modeling

I. INTRODUCTION

Grid Computing [1] is an established paradigm to form large virtual platforms for distributed super computing. Grids are commonly formed atop existing High Performance Computing (HPC) [2] infrastructures that contribute resources for use in one or more associated Grids. As such, Grids are heterogeneous by nature and commonly comprised of resources with varying hardware capabilities, running in different geographical locations, and often with different underlying software stacks.

Grid management software, middleware, allocates and manages jobs submitted to the Grid to the different resource sites that form the collaboration. This is commonly referred to as Grid scheduling [3], [4]. Each individual resource site in turn perform scheduling of jobs to each local resource, using resource management systems such as Maui [5] or SLURM [7]. The order in which the available jobs are executed on the local resources is determined by the local resource scheduler, and the resulting job priority commonly depends on several different factors such as job size, job age, or the fairshare priority [5] of the user submitting the job.

Fairshare priority represents the difference between actual and expected usage of a particular user, project, and/or Virtual Organization (VO) [8]. How fairshare priority is calculated and weighted together with other factors can differ between different resource scheduling systems. Since grids may be comprised of several different resource scheduling systems, the same job may be prioritized differently depending on to which underlying site the job is submitted for execution. Usage history and policy management is also delegated to each local site, making each site an independent fairshare prioritization system where only local history is considered.

Aequus [9] is a decentralized independent system for global fairshare prioritization that can be integrated with a wide range of resource management systems. Instead of calculating the fairshare priority locally at each site, local resource management systems query Aequus for a global priority factor. Aequus takes the global usage allocation of the executing user and any historical jobs previously executed on any site into account when calculating fairshare priority values, ensuring that jobs receive a comparable ranking regardless of to what resource site the job is sent for execution.

In this work, we show how Aequus is integrated with local resource management systems (SLURM and Maui) and describe how the system is deployed for use at High Performance Computing Center North (HPC2N). We also evaluate the functionality and performance of the fully integrated system under simulated nation-wide conditions, in preparation of having the system deployed at a nation-wide scale in the future.

Simulation input data are sampled from statistical models based on historical nation-wide usage. Early statistics from running Aequus in a production environment are also presented.

The rest of this article is organized as follows. Section II presents an overview of previous work on the Aequus system and other related work. Section III shows how Aequus is integrated with local resource management systems. Section IV presents an evaluation and validation of the fully integrated system, covering both data obtained from running in production and data obtained from running synthetic workloads on an integrated test-bed. The statistical models used to generate the synthetic workloads are also presented. Finally, conclusions and future work can be found in Section V.

II. BACKGROUND AND RELATED WORK

The work presented in this paper extends on earlier work on grid-wide fairshare prioritization. The first publication [10] presents the fundamental algorithm used to calculate the fairshare factors for each user, including evaluations of the approach using simulation. The second publication [9] describes the internal architecture of the Aequus system and features an in-depth functional evaluation of the system in an emulated
The process is made up of three main constituents: usage and pre-defined usage policies. The fairshare calculation obtains a global fairshare value, calculated based on grid-wide resource management systems with a call-out to Aequus to easily replace existing fairshare calculation mechanisms in and calculating fairshare on a grid-wide scale. It is designed to provide the necessary context for this work, an overview of the key aspects covered in earlier papers is presented in Section II-A.

Each of the previous publications present other work on fairshare related to Aequus. As this work employs statistical modeling as a tool for input data generation, complementing related work on using statistical modeling for system evaluation and its ability to isolate subgroups and adapt to events such as changing policies during run-time. In this work we focus on evaluating the system in a fully integrated setting using statistically modeled input data based on historical usage in our target environment.

A. Aequus Overview

Aequus is a distributed stand-alone system for managing and calculating fairshare on a grid-wide scale. It is designed to easily replace existing fairshare calculation mechanisms in resource management systems with a call-out to Aequus to obtain a global fairshare value, calculated based on grid-wide usage and pre-defined usage policies. The fairshare calculation process is made up of three main constituents:

- Hierarchical, tree-based, usage policies defining the target usage share of each user, project, or VO in the system. The usage share of one entity can be recursively subdivided into subgroups and sub-allocations. Globally managed sub-polices can be dynamically mounted into a locally administered root node, which allows local site administrations to manage the coarse allocation of resources to, e.g., a grid without having to manage the subdivision of usage within the grid itself. This allows local administrators to assign parts of the resources to one or more grids while retain full control over the infrastructure.
- Usage data representing the per-user resource consumption across all sites contributing to the grid. Normally, the Aequus installation of each site participating in the grid manages the usage data for jobs executed on the local infrastructure. The different Aequus instances then exchange usage information for grid users in a compact form, relaying the combined usage on each user on each site while omitting the details of individual jobs.
- An algorithm that calculates per-node fairshare values based on the usage policy and global historical usage of each user. The algorithm is parameterized and can be configured with, e.g., different usage decay functions to control how the impact of previous usage is decreased over time, or different ways of quantifying the difference between policy and usage.

The relation between each constituent is illustrated in Figure 1. As explained in [9], Aequus supports both absolute and relative distance metrics when comparing usage to policy shares. A configurable weight can be used to regulate the influence of each distance metric, with a default weight of 0.5 indicating that the absolute and relative components have equal weight. In Figure 1, only the absolute distance (usage share subtracted from policy share) is used for simplicity.

The Aequus system is implemented as a set of Java Web services that each capture a distinct part of the required functionality. An overview of the Aequus services can be seen in the corresponding part of Figure 2. The Policy Distribution Service (PDS) is responsible for managing user policies both locally and globally by mounting sub-policies from other sources (which may be other PDS services). The Usage Statistics Service (USS) gathers per-job usage results of the local site, and produces per-user histograms for configurable time intervals. The Usage Monitoring Service (UMS) of each site gathers usage histograms from one or more USSs and pre-computes usage trees based on the site-specific policies. The Fairshare Calculation Service (FCS) fetches usage trees from the UMS and policy trees from the PDS periodically, and pre-calculates fairshare trees with the current fairshare values for all users. This way, no real-time calculations need to take place when new jobs arrive, as pre-calculated values already exist and can be assigned to the job based on the associated user identity. The Identity Resolution Service (IRS) is an auxiliary service that can be used to revert the site-specific mapping process from grid user identity to a system user account. For more information about user identity management, see Section III-B.

Previous work has shown the stability of the algorithm and its ability to isolate subgroups and adapt to events such as changing policies during run-time. In this work we focus on evaluating the system in a fully integrated setting using statistically modeled input data based on historical usage in our target environment.

B. Related Work for Workload Modeling

A comprehensive guide to workload modeling written and published online by D. Feitelson can be found in [11]. This work covers many aspects of workload modeling and is based on research results from a number of publications by Feitelson.
and associates, e.g., [12], [13], [14]. Another study on workload characterization in cluster environments is presented by Li et al. [15]. In this work the workload of DAS-2 a distributed multi-cluster environment has been characterized, evaluated, and modeled using statistical methods. Iosup et al. [16] present means for advanced grid workload modeling, considering co-allocation, data and network management, and failure modeling.

Workload modeling for Web applications and server evaluation have similarities to grid and cluster workload modeling, although Web workloads have a strong focus on user behavior and the associated stream of interactions that make up a user session. Early work on the subject has been done by e.g. Barford and Crovella [17] and Arlitt and Williamson [18]. A brief summary on generative workload modeling can be found in [19].

III. INTEGRATION

The integration between Aequus and local resource management systems can be divided into three main tasks; technical integration, user identity management, and combining the global fairshare value with other factors such as job age or Quality of Service level.

A. Technical Integration

The technical integration is performed by implementing a unified system library (libaequus) that can be linked with local resource management systems to obtain access to the global fairshare functionality offered by Aequus. The libaequus library provides a C/C++ based interface that underneath contains Web service clients that communicate with Aequus to retrieve fairshare values, usage identity mappings, and store usage records. Previously resolved fairshare values and identities are cached within the library (for a configurable amount of time), which considerably reduces the amount of network traffic and computations required when batches of jobs are submitted and processed at the same time.

Figure 2 illustrates the different components involved in the fully integrated Aequus deployment. As shown in the figure, libaequus has been integrated with two different local scheduling systems, SLURM and Maui.

- SLURM integration is done by implementing custom Aequus priority and job completion plugins for use in the SLURM plug-in system. The priority plug-in is based on the existing multifactor priority plugin, with the normal fairshare priority calculation code replaced with a call to libaequus. A job completion plug-in supplies usage information to Aequus by calling libaequus.
- Maui has no inherit plug-in system, and therefore the integration is done by applying patches to the Maui source code. Similarly to SLURM, the local calculation of the fairshare priority factor is replaced with a call to the libaequus system library, and another call for supplying usage information to Aequus is injected into Maui for execution when jobs are completed.

B. User Identity Management

When a job arrives in the local resource management system, the global grid user identity of the user owning the job must be mapped to a corresponding system user on the local cluster. Exact mapping between grid user identity and system user can differ between resource management systems, between different sites depending on configuration, or even between clusters at the same site depending on local administrative policies. Normally, the resource management system is concerned only with the system user executing the job, and is unaware (and unaffected of) whether or not the job is executed by a local system user or as a global grid user being mapped to a local account. For local fairshare concerning only a single site or cluster this is not an issue, but global fairshare prioritization requires that the global grid user identities are consistently associated to any grid job regardless of where (and on what resource management system) it is being executed. The global user identity is used by Aequus throughout the entire fairshare prioritization process ranging from parsing share policy definitions to associating newly arrived jobs with historical grid-wide usage.

As briefly mentioned in Section II-A, Aequus contains auxiliary identity resolution features (using the IRS) that offer means to revert the mapping process from system user to grid user identity as part of the fairshare calculation process.
The revert mapping can be obtained in two ways; either by actively making a call to IRS to store the reverse mapping in a look up table, or by implementing a small custom mapping resolution end point and configuring the IRS to call the end point with name resolution queries using a minimalist JSON based protocol.

C. Combining Fairshare with Other Factors

As described in previous work [9], [10], Aequus employs an internal tree-based representation of the global fairshare values organized top-down into groups, subgroups, and users. The fairshare value of a specific user is obtained by extracting a vector of values representing the path from the root node down to the leaf node where the user is located. Figure 3 illustrates the fairshare tree and a set of fairshare vectors extracted from the tree. The vectors contain elements with a configurable resolution (the example uses a value range between 0 and 9999), and each element represents one level in the fairshare tree hierarchy. If a path should end before reaching the bottom level of the tree (like /LQ does in the example), the vector is padded with elements representing the balance point, which is the center value of the value range. Representing the fairshare values of users as vectors offers the following advantages:

Arbitrary Depth

The vectors support an arbitrary depth in the hierarchy, since the number of elements is unlimited.

Unlimited Precision

The precision of the values are limited only by the numerical resolution of floating point representation.

Subgroup Isolation

The priority of each element in the vector, representing an entity subgroup in the identity hierarchy, is affected only by other elements in the same group. This way a perfect top-down enforcement of fairshare usage can be obtained.

Proportional

The difference between fairshare values of different users is proportionally reflected in the resulting numerical value. If non-proportional, the resulting fairshare number correctly indicates the sorting order, but the relative difference is lost.

Both SLURM and Maui employ a linear combination of several factors to prioritize jobs, of which fairshare may be one among several. Each factor is represented by a value in the [0,1] range, and configurable weights are applied to factors to calculate the combined fairshare factor for a particular job. To retain this functionality while still using globally determined fairshare, the fairshare tree structure used by Aequus and the associated vectors needs to be projected down to a floating point value within the desired range.

A projection of the vector into a floating point number can in practice not be done while still retaining all above mentioned properties of the fairshare vectors. Aequus currently supports three different algorithms that can be used to project the fairshare vector to a single numerical value in the desired range, and each algorithm makes different trade-offs in terms of depth, precision, subgroup isolation, and proportionality. Each approach is described below, and an overview is presented in Table I. The approach to use is configurable and can be changed during run-time.

Dictionary Ordering

The vectors are represented as Strings sorted according to each element starting with the leftmost element, resulting in a descending sort of fairshare vectors. Each vector value is assigned an evenly spaced value in the [0,1] range relative to their rank. For example, three vectors would result in the numerical values 0.75, 0.50, and 0.25, according to sorting order.

Bitwise Vector

Each vector element is awarded N bits of entropy representing the fairshare balance at that level. The bits are bitwise-merged into a double data primitive with the bits representing the top level at the most significant end. The resulting number is re-scaled to the [0,1] range to obtain the final fairshare value.

Percental

A total target share of each user is calculated by multiplying the user share with the share of each ancestor node. For example, a project share of 0.20 and a user share of 0.25 result in a share of 0.05. The total usage is computed in the same way, multiplying the usage of the user with the usage of each parent node. The fairshare value is obtained by subtracting the usage share from the total target share, and re-scaling this value to the [0,1] range. A similar approach is used in SLURM [7] prior to version 2.5.

In-depth evaluation, characterization, and fine tuning of the above mentioned algorithms is part of our planned future work. More work on finding alternative approaches is also ongoing, where one interesting alternative is to reverse the problem and instead investigate modeling other factors, such as job age, using a representation combinable with the fairshare vectors.

IV. Evaluation

A comprehensive evaluation of the core mechanisms and system convergence was performed in the previous paper highlighting the Aequus architecture [9]. In this paper we complement the previous emulated tests with data from running the fully integrated system, including resource management systems and run-time identity resolution.

Production system tests has been performed in collaboration with High Performance Computing Center North (HPC2N). Aequus has been deployed along side SLURM 2.4.3 at a
clusters comprised of 68 nodes, each with dual 2.66 GHz quad-core Xeon CPUs (for a total of 544 cores), and 16 GB of memory. The maximum flop rate of the cluster is 5.8 TFLOPS. A single installation of Aequus is sufficient to handle the traffic from the cluster, and a small name resolution endpoint has been deployed in the HPC2N system to allow Aequus to revert the mapping of grid user identities to system users.

Since the system was deployed at the start of 2013, about 40,000 jobs per month has been executed on the cluster. During this period the system has shown to be stable and the transition from using local fairshare to global fairshare as performed by Aequus has had no noticeable impact on the performance or the stability of the cluster.

The production level tests are important as they help assess the stability of Aequus also when running across a full-scale cluster, and also when running for several months without manual intervention. However, running in a production environment by definition only produces one set of output, and the nature of a production environment does not allow us to affect neither the input nor experiment with the setup in order to focus on isolated aspects of system testing. Also, for practical reasons, the initial production level tests are limited to a single cluster rather than several distributed clusters.

To prepare for future deployment across multiple production clusters we first need to verify and assess the system functionality in nation-wide conditions using an emulated environment. To that end, the 2012 annual usage statistics for the Swedish National grid are used as input for simulated testing, using a test environment that corresponds to the national resources but at a smaller scale. For each of the national computing centers, a corresponding miniature local cluster has been set up with the full Aequus stack and SLURM as local resource scheduler. The resources of each local cluster corresponds to 10% of the resources at each computing center, using virtual resources as computational nodes. The actual computations are replaced with idle wait jobs to allow for large amount of virtual resources being hosted on the (considerably smaller) available set of physical resources.

1) Workload Characterization: To ensure that the workloads used during testing are relevant, a trace containing jobs run in 2012 across all national clusters is used as a baseline work arrival model for the distributed tests. Statistical models for job arrival and job duration have been created based on this data, and synthetic workloads generated from the statistical model are used during testing. By varying a few parameters in the model, such as the job arrival rate of specific users, it is possible to generate diverse workloads that still retain key statistical properties of the original trace while diversifying the test data to evaluate several different situations.

As suggested by Feitelson in [11], jobs that are submitted and managed by system administrators or automated monitoring systems are not representative of the actual workload and are removed prior to modeling. In addition, jobs with zero duration (most likely due to being canceled or failed) are considered outliers and are also removed. In total, about 15% of the total number of jobs, representing 1.5% of the total usage of the system, were removed prior to modeling.

Upon analyzing the trace it became clear that the vast majority of jobs are submitted by three different user identities. In reality, a single user identity may represent a major research project comprised of thousands of users rather than a single user, but this is indistinguishable from the perspective of the grid system. The major users are as follows:

- The most active user is responsible for 65.25% of the total wall-clock time usage, and 81.03% of the number of submitted jobs. We denote this user $U_{65}$.
- The second most active user represents 30.49% of the total usage and 6.58% of the number of submitted jobs. We denote this user $U_{30}$.
- The third most active user represents 2.86% of the total usage and 9.47% of the number of submitted jobs. We denote this user $U_{3}$.
- Consequently, the remaining users contributed 1.40% of the total usage and 2.93% of the number of submitted jobs, We denote this group of users as $U_{oth}$.

This approach is similar to the user-centric workload modeling discussed, e.g., in [17], but instead of user sessions (as present and central in Web traffic modeling) we instead aim to capture the general behavior of the dominating user groups, such as periodicity or bursty resource utilization. Due to the small number of jobs and low combined resource consumption, the jobs for all users but the three most active ones are grouped into a single user category to simply modeling.

The job arrival rate and size of incoming jobs (both with regard to the submitting user) are modeled separately. The following sections describes the separate models for job arrival and job size, respectively.

2) Job Arrival Modeling: The vast majority of jobs (81.03%) in the trace are submitted by $U_{65}$, and therefore the job arrival pattern of the trace is dominated by the job arrival pattern of $U_{65}$. This is illustrated in Figure 4.

The trace has been analyzed for periodicity using auto correlation functions [20], searching for daily, weekly, and monthly patterns for each user. However, no clear auto correlation patterns could be found. By isolating the job arrival for $U_{65}$, we can detect a pattern in job arrival about every three months (shown and emphasized in Figure 5).

Due to the dominance of jobs originating from $U_{65}$, we opted to model the job arrival for $U_{65}$ in more detail while fitting regular distribution functions to users $U_{30}$, $U_{3}$, and $U_{oth}$. For $U_{65}$ the job arrival model is partitioned into four

| Table I: Overview of algorithms projecting fairshare vectors to singular numerical values. |
|----------------|----------------|----------------|----------------|----------------|----------------|
|                | $\infty$ Depth | $\infty$ Precision | Subgroup Isolation | Proportional | Combinable |
| Fairshare vectors | ✓ | ✓ | ✓ | ✓ | ✓ |
| Dictionary Ordering | ✓ | ✓ | ✓ | ✓ | ✓ |
| Bitwise Vector | × | × | ✓ | ✓ | ✓ |
| Percental | ✓ | ✓ | × | ✓ | ✓ |
different phases. We know from the historical data that $U_65$ represents a large scale research project, and each cycle is likely to stem from different experimental cycles lasting about three months. A separate distribution fit is created for each of the identified phases (denoted $pn$). The combined probability density function (PDF) by scaling the result of each distribution with the percentage of jobs located within the corresponding section of the trace (denoted $pm_usage$). The combined function is shown in Equation (1), and illustrated along side the job arrival for $U_{65}$ in Figure 5.

For all users, the best fit was found by modeling each data set using a set of 18 different distributions, and choosing the best fit based on the Bayesian information criterion [21]. The set of distributions includes distributions such as normal, Weibull, Generalized Extreme Value (GEV), Birnbaum-Saunders (BS), Pareto, Burr, and Log-normal.

The resulting distributions, parameters, and their goodness of fit values using Kolmogorov-Smirnov [22] tests are shown in Table II. Matlab [23] is used to calculate median values, perform auto correlation, evaluate fits, and calculate goodness of fit values. Downey and Feitelson [12] make a strong case regarding the lack of relevance of mean and coefficient of variation (CV) metrics of the original data. Their main argument is that no data exists to tell if outliers are caused by corrupted data or by legitimate unusual events, which makes the decision to remove outliers or not (and how many) completely arbitrary, while having a considerable impact on the mean and CV metrics. Instead, they suggest the use of median values as a metric more resilient to outliers, which is what is used in this work. The median values calculated for both job inter-arrival times and job durations are even seconds, since the time stamps from the original trace are limited to second accuracy. As shown in Table II the median inter-arrival time of $U_3$ is listed as zero seconds, which means that more than half of the jobs arrive within the same measured second as the previous job.

As shown in Table II, most of the data sets are represented using the GEV distribution. The exception is $U_{30}$ for which we found a better fit using a Burr distribution. The goodness of fit values indicate that the resulting fitted distributions (including the composite distribution for $U_{65}$) are reasonably close to the original data sets. The data set with the worst fit is $U_3$, where the distribution cannot fully capture the burst in usage that the original data set inhibits. This behavior, and the resulting job arrival cumulative distribution functions (CDF) and empirical job arrival patterns for all users are illustrated in Figure 6.

When creating synthetic traces the inverse CDF (ICDF) is used to model arrival time as a function of probability, and random values in the $[0, 1]$ range are used to sample jobs arrival times using the ICDF. Distributions are not bounded by any designated time span, and although the probability of samples converge towards zero as the distance from the center increases there is always a slight possibility that samples are generated far from the intended region. To ensure that all samples are within the intended range, the distribution of random values $[0, 1]$ is therefore re-scaled to fit within the desired time frame. For example, in the case of $U_{65}$, the effective range $[7.451 \times 10^{-3}, 9.946 \times 10^{-1}]$ is used to ensure all generated values are within the same calendar year.

$$PDF_{U_{65}}(x) = \sum_{n=1}^{4} \left(\frac{pm_usage}{total_usage}\right) \ast PDF_{pn}(x) \tag{1}$$

Figure 4: Jobs arrival as a function of time. Bin size is one day. Shown is both total jobs and jobs for $U_{65}$.

Figure 5: Probability density of job arrival as a function of time. Bin size of histogram is one day. Shown is the empirical job arrival and the constructed job arrival function for $U_{65}$. Dashed lines delimiter the identified phases 1 to 4.

Figure 6: Cumulative probability of job arrival as a function of time. Thin lines indicate fitted functions, thick lines indicate empiric data.
Table II: Job arrival: Median inter-arrival value of original data (whole seconds), the best found fitted distribution for each data set and the corresponding Kolmogorov-Smirnov goodness of fit values.

<table>
<thead>
<tr>
<th>User</th>
<th>Median(s)</th>
<th>Fitted Distribution</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>U_{65} (p1)</td>
<td>2</td>
<td>GEV(k = -0.386, σ = 19.5, μ = 7.35 \times 10^4)</td>
<td>0.06</td>
</tr>
<tr>
<td>U_{65} (p2)</td>
<td>3</td>
<td>GEV(k = -0.371, σ = 30.6, μ = 7.35 \times 10^4)</td>
<td>0.05</td>
</tr>
<tr>
<td>U_{65} (p3)</td>
<td>2</td>
<td>GEV(k = -0.457, σ = 30.8, μ = 7.35 \times 10^4)</td>
<td>0.07</td>
</tr>
<tr>
<td>U_{65} (p4)</td>
<td>2</td>
<td>GEV(k = -0.301, σ = 21.4, μ = 7.35 \times 10^4)</td>
<td>0.05</td>
</tr>
<tr>
<td>U_{65} (p8)</td>
<td>2</td>
<td>(see Equation 1)</td>
<td>0.02</td>
</tr>
<tr>
<td>U_{30}</td>
<td>1</td>
<td>Burr(α = 7.4 \times 10^4, c = 8.6 \times 10^{-4}, k = 0.08)</td>
<td>0.08</td>
</tr>
<tr>
<td>U_{3}</td>
<td>0</td>
<td>GEV(k = 0.195, σ = 29.1, μ = 7.35 \times 10^4)</td>
<td>0.15</td>
</tr>
<tr>
<td>U_{oth}</td>
<td>13</td>
<td>GEV(k = 0.148, σ = 56.0, μ = 7.35 \times 10^4)</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Table III: Job duration: median job duration of original data (seconds), the best found fitted distribution for each data set and the corresponding Kolmogorov-Smirnov goodness of fit values.

<table>
<thead>
<tr>
<th>User</th>
<th>Median(s)</th>
<th>Fitted Distribution</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>U_{65}</td>
<td>1.70 \times 10^8</td>
<td>BS(β = 1.76 \times 10^4, γ = 3.53)</td>
<td>0.09</td>
</tr>
<tr>
<td>U_{30}</td>
<td>2.38 \times 10^7</td>
<td>Weibull(λ = 5.49 \times 10^4, k = 0.637)</td>
<td>0.04</td>
</tr>
<tr>
<td>U_{3}</td>
<td>1.12 \times 10^7</td>
<td>Burr(α = 2.07, c = 11.0, k = 0.02)</td>
<td>0.28</td>
</tr>
<tr>
<td>U_{oth}</td>
<td>3.37 \times 10^6</td>
<td>BS(β = 3.02 \times 10^4, γ = 7.91)</td>
<td>0.13</td>
</tr>
</tbody>
</table>

3) Job Size Modeling: The trace is comprised exclusively of bag-of-task jobs using a single processor per job. The empirical CDF of the job sizes for each user is shown in Figure 7. As can be seen in the figure, the job size distributions for users U_{65}, U_{3}, and U_{oth} are focused in the [0, 6 \times 10^5] range, while U_{30} exhibits a larger tail and generally exhibits larger job sizes.

The fitted job duration distributions along with median and goodness of fit values are shown in Table III.

A. Test Results

The statistical models have been used to generate a set of different synthetic workloads used to validate system functionality and performance on the test bed, and a subset of those tests are presented in this section. The test bed is comprised of seven ProLiant DL165 G7 machines, each with AMD Opteron(TM) Processors containing 32 CPUs and 56 GB of physical memory. During the tests, six of the hosts are configured to represent one cluster with 40 virtual hosts each for a total of 240 hosts, corresponding roughly to 10% of the national grid capacity. One machine is used to parse the input workload and submit the jobs to each of the clusters. Both stochastic and round-robin scheduling of jobs from the submitting node to the clusters have been evaluated without any noticeable difference, and the stochastic approach is used during the testing. Each of the simulated clusters hosts its own Aequus installation, and they communicate only by exchanging data though the USS services, just like a full scale deployment is likely to be. A unified name resolution service used by all clusters is co-hosted on the job submission host.

A key feature with using synthetic traces as input is the ability to scale down testing time and run full simulations in a shorter time frame. By performing a set of throughput tests, the test bed was found to support a sustained job submission rate of about 120 jobs per minute. The peak job submission rate during the bursty test shown in this article reaches 472 jobs per minute. During these tests, the traces contain a total load of 95% of the theoretical maximum of the combined infrastructure, and during testing we have found that the total utilization varies between 93% and 97%. The test length is six hours for all tests, and each trace contains 43,200 jobs.

Apart from a workload, any fairshare system also needs target policy shares that represent balance in the system. In production environments the target policy shares are usually either awarded by an allocation committee or set in relation to the fees paid for running on the infrastructure. In this case, the system strives to converge towards the specified shares and may or may not succeed depending on input. In our simulations, we take the opportunity to set target shares suitable for evaluation and illustration purposes, showing how the system behaves under certain, reproducible, conditions. As the focus of the following tests lies in how the system behavior is affected by integration and deployment rather than policy shares, the actual share from the workloads are used as targets for most of the tests. However, we also illustrate how the system behaves when policy shares are not aligned with the future load.

Fairshare is the only scheduling factor used during these tests. By focusing on fairshare, scheduler-specific semantics related to how fairshare is combined with other factors are avoided, and tests can focus on highlighting the behavior of the Aequus system. The percental projection approach is used during testing, as this is the configuration currently used in production. Complementary tests with other factors in addition to fairshare have been performed, and show that other factors have a smoothing effect (with impact relative to their weight) on the fluctuating behavior natural to fairshare.
2) Impact of Update Delay: In our tests workload modeling is used to project long term usage patterns to a shorter time span which is more suitable for repeated evaluation. A fundamental problem is that update and processing delays in the system cannot be shortened with the corresponding rate. The delay from when a job has finished executing until its resource consumption is considered for fairshare depends on (I) reporting delay from the local resource manager to Aequus, (II) cache time in USS, UMS, and FCS services, (III) cache time in libaequus, (IV) local resource manager re-prioritization interval.

To evaluate the impact of the delay we scaled the baseline test case up ten times, adjusting the arrival times and job durations while keeping the same number of jobs and same internal relations. The results, illustrated in Figure 11, show that a magnitude shorter update and delay times contribute to a 10% - 15% shorter convergence time compared with the baseline case (shown in Figure 10a). This eliminates relatively large update delays due to scaling as a significant error source.

3) Non-optimal Policy Test: In this test we show how the system behaves when the policy used in the test does not perfectly represent the usage in the input trace, as may often be the case in realistic usage scenarios. The test settings remain the same as during the first test, including the actual wall-clock time usage of each user. However, the policy file has changed to having a target policy of 70% for $U_{65}$, 20% for $U_{30}$, 8% for $U_{3}$ and 2% for $U_{oth}$. Test data is illustrated in Figure 12.

Test results show that the system is close to balance in the 120 to 180 minute range, but as the availability of $U_{65}$ jobs drops the system can no longer maintain balance. As new jobs from $U_{65}$ arrives around the 240 minute mark they are scheduled as soon as resources become available, and the priority and combined usage share temporarily starts converging again. At the end of the tests mostly jobs by $U_{30}$ are available, and to maximize utilization these jobs are run despite receiving a lower priority.

4) Partial Cluster Participation: An interesting scenario is when a subset of interconnected Aequus installations are not fully taking part in the global usage data exchange. This could either be, e.g., due to misconfiguration, local policies, or legislation. If an Aequus installation is neither receiving nor contributing usage data, that installation is disjunct from any other installations and have no impact on their operations.

In the test, one of six sites only reads global usage data but does not contribute, and another of the clusters contributes data but only considers local data for job prioritization. The site which only reads data considers both global and local usage for priority decisions.

The results show the priority on the site reading global data remains well aligned with the priority of fully participating sites. The site that only considers local data for scheduling...
converges towards the same priority levels but at a slower pace and with more fluctuations. The data from this site act as noise for the other sites, but this noise does not have a noticeable impact on the global fairshare prioritization.

5) Bursty Usage Test: For the final test the relative job submission rate of $U_3$ was considerably increased to emphasize the bursty usage behavior observed of this user. The job submission rate of the periodical user $U_{65}$ was deducted by a corresponding amount, and the other two users remained the same. To illustrate the systems ability to reach balance also when a user is not submitting, the burst was shifted to start after one third of the test run rather than early on as in the original case. The job arrival model is shown in Figure 13c.

The fractions of submitted jobs per user for this test are 45.5%, 6.5%, 45.5%, and 3% for $U_{65}$, $U_{30}$, $U_3$, and $U_{oth}$, respectively. The corresponding wall-clock time usage shares are 47%, 38.5%, 12%, and 2.5%. Note that the relative usage share of $U_{30}$ and $U_{oth}$ increase in this scenario, even though their relative job share stays constant with previous tests. This is because the job durations of $U_3$ are considerably shorter than those of $U_{65}$, and a larger share of $U_3$ jobs compared to the original case will therefore cause the total duration of the synthetic trace to be lower. To scale the trace load up to the desired system load, a higher scaling factor is required and therefore the relative usage shares of $U_{30}$ and $U_{oth}$ increase.

Figure 13b shows the priority for each user during the test. As previously mentioned the fairshare algorithm uses a configurable weight($k$) between absolute and relative distance calculations. During these tests, $k = 0.5$. The relative component is always in the range $[0, 1]$, while the absolute component is in the range $[0, (UserShare)]$. For $U_3$ in this test, this indicates a maximum priority value of $0.5 \times (1 + 0.12) = 0.56$, which is consistent with the data shown in Figure 13b. As shown in Figure 13a, the system converges towards a balanced state between minute 80 and minute 130, where the unused allocation of $U_3$ is divided between the other users. When the burst of usage by $U_3$ occurs around the 130 minute mark, the system readjusts and starts adapting towards the original shares.

V. CONCLUSIONS AND FUTURE WORK

In this work we show how Aequus, a global fairshare prioritization system, can be integrated with local resource management systems to enable grid-wide fairshare prioritization with minimal intrusion. The integration is performed by linking resource management systems to a unified system library, libaequus, either through native plug-in systems or by applying minor patches to the source code. The fully integrated solution has been deployed in production, and early data from production indicate that the system is stable and able to handle realistic workloads. Further tests of the system in a
ACKNOWLEDGMENTS

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 the High Performance Computing Center North (HPC2N) project. The authors would like to thank Tomas Ögren, Åke Sandgren, Magnus Jonsson, and Roger Oscarsson for technical assistance, integration work, and systems management done within the context of this project.

REFERENCES