KTH Computer Science
and Communication

Managing Service Levels in Grid Computing Systems

Quota Policy and Computational Market Approaches

THOMAS SANDHOLM

Licentiate Thesis
Stockholm, Sweden 2007

TRITA CSC-A 2007:6
ISSN 1653-5723 KTH School of Computer Science and Communication
ISRN KTH/CSC/A--07/06--SE SE-100 44 Stockholm
ISBN 978-91-7178-658-6 SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av filosofie licensiatsexamen i datalogi måndagen den 14 maj 2007 klockan 10.00 i Sal 304, Parallelldatorcentrum, Teknikringen 14, Kungl Tekniska högskolan, Stockholm.

Tryck: Universitetsservice US AB

Abstract

We study techniques to enforce and provision differentiated service levels in *Computational Grid* systems. The Grid offers simplified provisioning of peak-capacity for applications with computational requirements beyond local machines and clusters, by sharing resources across organizational boundaries. Current systems have focussed on access control, i.e., managing who is allowed to run applications on remote sites. Very little work has been done on providing differentiated service levels for those applications that are admitted. This leads to a number of problems when scheduling jobs in a fair and efficient way. For example, users with a large number of long-running jobs could starve out others, both intentionally and non-intentionally.

We investigate the requirements of High Performance Computing (HPC) applications that run in academic Grid systems, and propose two models of service-level management. Our first model is based on global real-time quota enforcement, where projects are granted resource quota, such as CPU hours, across the Grid by a centralized allocation authority. We implement the SweGrid Accounting System to enforce quota allocated by the Swedish National Allocations Committee in the SweGrid production Grid, which connects six Swedish HPC centers. A flexible authorization policy framework allows provisioning and enforcement of two different service levels across the SweGrid clusters; high-priority and low-priority jobs. As a solution to more fine-grained control over service levels we propose and implement a *Grid Market* system, using a market-based resource allocator called Tycoon.

The conclusion of our research is that although the Grid accounting solution offers better service level enforcement support than state-of-the-art production Grid systems, it turned out to be complex to set the resource price and other policies manually, while ensuring fairness and efficiency of the system. Our Grid Market on the other hand sets the price according to the dynamic demand, and it is further incentive compatible, in that the overall system state remains healthy even in the presence of strategic users.

Keywords: Grid Market, Computational Grid, Service Level Management, QoS, HPC, Grid Middleware

Sammanfattning

Vi studerar metoder för att tillhandahålla och upprätthålla olika servicenivåer i Grid system för storskaliga beräkningar. Grid modellen gör det enklare att tillgodose den maxkapacitet som storskaliga beräkningar kräver genom att möjliggöra ett dynamiskt och automatiserat utbyte av datorkraft mellan olika organisationer. Dagens Grid system fokuserar på behörighetskontroll, dvs hanterande av vem som tillåts köra applikationer på främmande system. Väldigt lite arbete har ägnats åt att erbjuda olika servicenivåer till de som har behörighet. Detta leder till åtskilliga problem när jobb ska distribueras och köras på ett effektivt och rättvist sätt. Användare som kör många långa jobb kan, t.ex. blockera andra körningar, både medvetet och omedvetet.

Vi undersöker kraven som storskaliga beräkningsapplikationer ställer på infrastrukturen i akademiska Grid system, och föreslår två modeller för att hantera servicenivåer. Vår första modell baserar sig på global kvotakontroll i realtid, där forskningsprojekt tilldelas en kvota datorkraft, som t.ex. CPU timmar, av en centraliserad allokeringsenhet. Vi implementerar SweGrid Accounting System, ett system för att se till att resurs kvota som tilldelats forskare av Swedish National Allocations Committee, levereras av ett nätverk av datorer, SweGrid, som sammanbinder sex super- och parallelldatorcentra i Sverige. Ett enkelt konfigurerbart policystyrt auktoriseringsramverk tillåter tillhandahållande och upprätthållande av två olika servicenivåer ; högprioritets- och lågprioritetsjobb. För att få ytterligare och bättre kontroll över servicenivå föreslår och implementerar vi en marknad för Grid resurser som avänder sig av Tycoon, ett marknadsbaserat allokeringssystem för datorresurser.

Slutsatsen av vår forskning är att trots att SweGrid Accounting lösningen erbjuder bättre servicenivåstöd än dagens Grid system, visade det sig vara komplicerat att konfigurera resurspris och andra policyvärden manuellt, och samtidigt tillförsäkra en rättvis och effektiv allokering. Vår lösning med en Grid-marknad å andra sidan sätter priser utefter efterfrågan dynamiskt, och den är *incitamentkompatibel*, dvs systemet som helhet förblir effektivt och rättvist trots att det finns strategiska användare som försöker utnyttja det.

Contents

Contents **v**

1 Introduction **1**
1.1 Problem Statement 2
1.2 Scope 2
1.3 Approach 2
1.4 Organization 3

I Background and Results **5**

2 Foundation **7**
2.1 Grid Computing 7
2.2 Market Theory 11

3 Software **15**
3.1 SweGrid Accounting System 15
3.2 Tycoon Grid Resource Allocator 17

4 Results **21**
4.1 Thesis Papers 21
4.2 Additional Publication Contributions 24
4.3 Related Work 25
4.4 Conclusions 27
4.5 Future Work 27
4.6 Acknowledgments 28

Bibliography **29**

II Papers **37**

Chapter 1

Introduction

Large-scale networks are evolving rapidly to become faster, more reliable, and more accessible, which is exemplified by the enormous technical as well as social impact of the Internet. This trend is a result of advances in computer science and engineering, such as more efficient hardware and network protocols, but it is also an indirect result of the advances in physical and social sciences, such as Bioinformatics, High-Energy Physics, and Economics demanding increased capacity for data processing, storage, and transfer. These demands are typically fluctuating over time, making it impractical to purchase dedicated hardware that is unutilized most of the time, and furthermore quickly becomes obsolete. As hardware, operating systems, and networking software are commoditized, it becomes more feasible to share these resources. A new array of computing systems thus evolved to govern the sharing of resources in large-scale networks.

The power-grid utility paradigm is often used to describe such systems. It should be as easy to upgrade your computing capacity as plugging in your appliance into a power socket and turning a knob to get more electricity. One set of problems that need to be tackled to achieve this involves agreeing on standards for communication interfaces and protocols, another is related to ensuring that the shared resources are correctly used in a secure way and that the usage is accounted and charged for regardless of where it was provisioned. A final set of problems involves offering a variety of service levels for customers with different needs and preferences in an economically fair and efficient manner.

The state-of-the-art Grid systems have made great progress in interface standardization recently and have also tackled many of the security related problems involved in executing applications remotely. Grid Accounting systems are in development but not yet widely deployed, and not yet standardized, which complicates charging for compute resource usage. However, the most apparent shortcoming of today's Grid systems is the lack of provisioning and enforcement of service levels. The problem has been addressed from a linguistic perspective by inventing new service level agreement languages and negotiation protocols, but very little

has been done to facilitate provisioning and enforcement of these agreements. As a result it is hard to make the current Grid deployments economically sustainable and thereby offered in a commercial setting as opposed to in a government-funded research project.

1.1 Problem Statement

In this thesis we[1] investigate what infrastructure can be added to existing HPC Grid systems to automatically provision and enforce service levels more accurately and easily.

Provisioning of service levels is the process when resource providers offer and advertise different levels of service performance to users, whereas enforcement, a.k.a. policing, of service levels involves making sure that the promised levels are indeed delivered. These two activities are deeply interrelated, and we thus consider the combination, referred to as service-level management here.

1.2 Scope

We examine service-level management from a middleware perspective. That is, we study what tools can be developed to help Grid application programmers take better advantage of the shared resources while still assuring that the overall state of the system is healthy. Our focus is not on advancing research in the graphical end-user interface design nor the design of the networking fabric, but rather to improve the technology that bridges the two.

1.3 Approach

We have investigated two different approaches to service-level provisioning and enforcement in Grids. The first approach relies on a Grid accounting system, which we developed, that allows centrally set project quota policies as well as locally configured resource provisioning policies determine the service-level for users across HPC clusters. The second approach is based on resource virtualization and slicing in a proportional share, market based resource allocator.

Simulations, benchmarks, experiments, and analyzes of production system deployments with real users are all methods used to verify the results and the feasibility of our models and their implementation in different settings and against alternative solutions.

[1]The term *we* is used throughout this thesis to denote the work lead and performed by the author while collaborating with other researchers. Where there are joint contributions, the parts done by the author are explicitly stated.

1.4 Organization

The thesis is organized as follows. In the first part we summarize the background and results of our work. Chapter 2 presents the problem domain and the underlying technology and theory. The software that was developed as part of the thesis research is described in some more detail in Chapter 3 and then the contributions and the thesis papers are summarized with future work in Chapter 4.

In the second part we include the thesis papers previously published in a journal, conference proceedings, a technical report, and a research manuscript.

Part I

Background and Results

Chapter 2

Foundation

In this chapter, we discuss the foundational concepts and theory of the work presented in this thesis. First, we describe the new paradigm of computing emerging in *Computational Grid* systems. Second, we review the underlying theory of markets including game theory, and fundamental micro-economic theory.

2.1 Grid Computing

In the context of this thesis the Grid refers to a collection of computational resources shared across organizational boundaries to deliver non-trivial Qualities of Service (QoS) [28, 27, 5]. Non-trivial here means that services beyond pure information sharing, as typical in the World Wide Web, are offered. What is in common for these more advanced services offered by a Grid is that they typically involve large-scale resource consumption within a dynamic community of users and providers spread across a large geographic area. One of the first super computing projects to span multiple organizations and utilizing a cross-Atlantic Grid was the I-WAY project [17], which paved the way for Grid computing as a scientific field. This community is known as a *Virtual Organization* (VO) [26]. An example VO architecture is shown in Figure 2.1.

Security

Many of the trust, privacy and general security issues appearing in the Grid revolves around management of rights within a VO. The idea is that a VO is a web of trust where information exchange and resource sharing can take place just like in a corporate Intranet. The difference is that Virtual Organizations may be created, managed and destroyed in a more dynamic manner. Examples include ATLAS [1], a particle physics experiment utilizing the computational Grid of the Large Hydron Collider at CERN; and HapGrid, a bioinformatics project performing haplotype

[1] http://atlas.web.cern.ch/Atlas/

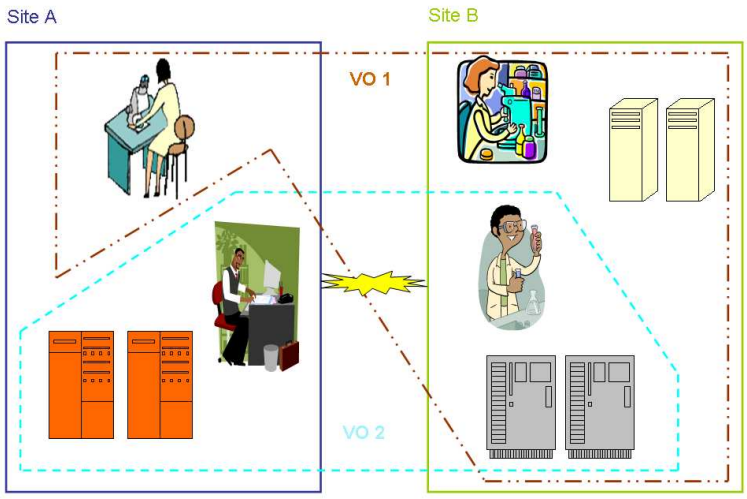

Figure 2.1: Virtual Organization Example.

reconstruction and frequency estimation using the SweGrid computational Grid resources [2].

The trust verification mechanism in Grid systems is based on the Public Key Infrastructure (PKI) [36], with extensions to allow delegation of rights and single sign-on using self-signed *proxy certificates* [65, 69]. A user will have a secret key on her local machine and then distribute a public key to all communication partners. A message can then be signed or encrypted with the private key by the sender to allow the recipient to verify the authenticity of the message including non-repudiation, and protection against denial-of-service (DoS) and replay attacks. The PKI handshake protocol where authenticity is verified has two main advantages compared to more traditional username and password based authentication protocols. First, no personal secret such as a password or private key needs to be sent across the communication link exposing it to eavesdropping. Second, mutual authentication of senders and receivers is seamless, making it a good fit for peer-to-peer like systems, such as the Grid. Another fundamental concept is the *Certificate Authority* (CA), which is a trust anchor asserting the identity of its users by signing their credentials (public keys). CA's may be established for individual VO's, a collection of VO's using a particular Grid environment, a country for its citizens, etc. Certificate Authorities may also be organized hierarchically, where the parent nodes assert the identity of their child nodes.

The use of proxy certificates allows brokers or agents to act on behalf of users

to complete a task. The broker will not simply receive the private key of the user, as it would violate the rule of *strong authentication*, which states that no long-lived personal secrets should be distributed as part of the identity verification process. Instead the user creates a temporary key-pair, signs it, encrypts it, and sends it to the broker. Proxy certificates thus enables single-sign on across a network of brokers.

Policy Management

In cross-organizational wide-area networks with community overlays, such as Virtual Organizations in Grid systems, managing policies for all stakeholders becomes a challenging task. The policies of resource providers, funding agencies, virtual organizations, and users must be combined in order to make accurate authorization and service-level decisions. Policies may either be *pulled* in from 3rd parties, by intercepting the message flow and making call-outs, or *pushed* to decision makers by attaching capability assertions to the message payload [69]. Combinations of the push- and the pull-models are also common and the policy decisions may be made both on the client and on the server side in a client-server interaction [47]. Policy-based systems aim to manage resources by enforcing, evaluating, retrieving, administering and combining policies with standard protocols. By communicating with all the heterogeneous resources in the same way, generic tools can automatically manage arbitrarily complex networks of resources and stakeholders by means of control feedback-loops, a.k.a. the MAPE (Measurement, Analysis, Planning, and Execution) or autonomic self-management model [41]. Policy-based management systems can dynamically change the configuration of resources in response to events that in turn were triggered by various system states occurring. A key to designing efficient policy-aware systems is to separate the policy-related tasks into different layers and making them agnostic to the application code. These layers, often referred to as *policy points*, can be stacked and combined in arbitrarily complex policy-decision trees. The different layers and their responsibilities, summarized below, comprise a common architecture and nomenclature for policy-based systems [1].

- A Policy Enforcement Point (PEP) intercepts the execution path and calling decision points, i.e. a PEP integrates the application with the policy mechanisms.
- A Policy Decision Point (PDP) combines and evaluates local as well as external policies, and may call information points to retrieve policies to base its decision on. The result from an evaluation is typically *permit*, *deny*, or *not applicable*. The result may also contain obligations that must be met for the result to take effect.
- A Policy Information Point (PIP) stores and retrieves policies, e.g. returns roles that an authenticated user has in an RBAC (Role-Based Access Control)

system [21]. The additional information can then be used by the PDP and is sometimes called evidence or context credentials.

- A Policy Administration Point (PAP) sets and configures policies used by the PDP and PIP layers.

Accounting and Systems Integration

Grid middleware services include secure remote execution management, remote data storage and replication, monitoring services, and high-volume file-transfer services. What distinguishes these services from other network services such as FTP, WWW, and LDAP, is that a Grid needs to handle higher volumes of data to transfer and store; and allow VO-enabled access control, and execution of arbitrary applications. Furthermore, all users and Virtual Organizations are accountable for their resource usage in a Grid, in order to promote fair sharing.

Accounting services thus play a fundamental role in Grids. Grid accounting systems must be able to handle VO-scoped accounting of the usage of heterogeneous resources. Standard accounting records that translate and coalesces resource and site specific usage need to be exchanged and coordinated across the Grid. This process is complicated by the diversity of resource management technology and policies (e.g., security, accounting, auditing) in HPC centers offering Grid resources [57, 19, 58].

Resource heterogeneity in Grid systems was first addressed by the Open Grid Services Infrastructure protocol [66], which specifies a standard protocol interface for managing the state of a Grid resource [25]. It was founded on state-of-the-art systems integration technology of the time, including the XML Web services stack with extensions, and it was adopted by the Global Grid Forum (GGF) standardization body. This work later evolved into the Web Services Resource Framework [30], within the OASIS standards group, which now makes up the backbone infrastructure of various distributed management standards, such as OASIS WSDM [10] and GGF WS-Agreement [3].

Resource Allocation

Service level and QoS enforcement was addressed in a Grid context in the Grid Advanced Reservation and Allocation (GARA) [22, 23] project allowing CPU, bandwidth and OS process resource capacity enforcement at different levels of service. Here resources were configured using resource specific control mechanisms, such as DiffServ and RSVP router management [6, 8], and DSRT CPU scheduling control [49]. This work evolved into the SNAP protocol [15] and then eventually was standardized in the WS-Agreement specification [3], by GGF, which also borrows many concepts from IBM's WSLA (SLA for Web services) solution [16] and SLAng [46].

Complimentary to protocol standardization, heterogeneity can also be addressed by resource virtualization. For example, virtualization of a host operating system [18] gives fine-grained control over the service levels offered. CPU, disk, memory, and other resource shares can be allocated to user specific virtual machines. This technique has been explored in the context of Grid job execution management in [40].

As the Grid deployments extend beyond academic projects, such as EDG [2] [7], EGEE [3] , TeraGrid [4], NEESit [5], ESG [6], and OSG [7] to self-sufficient commercial Grid environments, the need to charge for compute resource usage like any other commodity arises. This business model is in-line with many IT companies' utility computing strategy [31, 12, 35]. Economic models from the field of utility computing could also solve the growing problem in academic Grid projects of a small number of strategic users hogging the system. We will elaborate on how this could be approached in the next section.

2.2 Market Theory

When managing service levels, we would like to make sure that the system cannot be abused by strategic users, who could starve out competing resource consumers. We therefore turn to economic theory to study how mechanisms can be developed to ensure an overall healthy system even with strategic users.

Tragedy of the Commons

Consider the problem often referred to as the *Tragedy of the Commons* [33]. Farmers let their sheep eat grass on a common. A farmer can sell one of his sheep when it has been well fed and earn a profit compared to the original purchase price of the sheep. Let's further assume that the profit that an individual farmer gains from selling a sheep is higher than the relative cost of having one more sheep share the grass of the common, and thus leaving less grass available for other sheep. A strategic farmer who is trying to optimize his own profits would under such circumstances always choose to purchase another sheep. The main issue with this situation is that the overall health of the community of farmers declines as individuals optimize their profits, and eventually it will collapse when there are too many sheep on the common for any single one of them to get fed well enough to be sold. It is not hard to see that such situations could easily arise if compute power is offered as a common good without providing some incentive for users to constrain their usage.

[2] European Data Grid, http://www.edg.org
[3] Enabling Grids for EScienceE, http://egee-intranet.web.cern.ch/egee-intranet/gateway.html
[4] http://www.teragrid.org
[5] http://it.nees.org
[6] Earth System Grid, http://www.earthsystemgrid.org
[7] Open Science Grid, http://www.opensciencegrid.org

Game Theory

In *Game Theory* [52, 51] a number of *players* and their possible *actions* with associated individual *preferences* model a *game*. Other players' actions affect the *utility* or *payoff* a player receives from a game. However, the other players' actions may not be known before a player chooses an action. In order to choose an action each player hence needs to make a guess of other players' likely actions given past experience, which is referred to as forming a *belief*.

Let

$$a^* = \{a_1 ... a_k\}$$

be the set of actions taken by the k players in a game, where a_i is the action taken by player i. This set is called the *action profile* of the game.

We can now make statements about the *steady states* of a game, when no player has an incentive to change her action.

Nash Equilibrium

A *Nash equilibrium* is defined as an action profile a^* where no player i can get a higher utility by changing her action a_i^*, given that every other player j performs the action a_j^*. More concisely expressed

$$u_i(a^*) \geq u_i(a_i, a_{-i}^*)$$

for every action a_i of player i, where u_i is the utility function that represents player i's preferences and (a_i, a_{-i}^*) is the action profile where player i performs action a_i and all other players j perform action a_j^*.

It is important to note that a Nash equilibrium does not make any statements about uniqueness of the solution, and many games can indeed have multiple Nash equilibria.

To simplify the decision making process for a player given prior beliefs a *best response function* is typically defined. It yields the set of best actions to take for a player given an action profile of the other players, or more precisely

$$B_i(a_{-i}) = \{a_i \in A_i : u_i(a_i, a_{-i}) \geq u_i(a_i', a_{-i})|_{\forall a_i' \in A_i}\}$$

where A_i is the set of all possible actions player i can take, a_{-i} the action profile including all players except player i, and B_i is the set of best response actions.

Resource Allocation Game

In our case a game can be defined as the process of allocating available Grid resources, or shares of a resource, to the applications that users are requesting to run on those resources. The users can form their prior beliefs of other users' demand of the resources by studying the current resource prices on the market. In order to

analyze the efficiency and fairness of a resource allocation algorithm we need some additional definitions.

The *efficiency* or *price of anarchy* [53] is calculated as the sum of all users' utilities of a certain allocation outcome compared to the optimal utility in the system. The sum of all users' utilities is typically referred to as the *social welfare*, and it is an indication of the global health of the system.

The social welfare for an allocation scheme ω is defined as

$$U(\omega) = \sum_{i=1}^{k} u_i(r_i)$$

where r_i is the resource share allocated to user i, and u_i is the utility function of user i.

The fairness of a resource allocation scheme can be defined in terms of *envy-freeness* [67] which can be calculated as

$$\rho(\omega) = \min(\min_{i,j} \frac{u_i(r_i)}{u_i(r_j)}, 1)$$

where $u_i(r_i)$ is the utility that user i received from being allocated share r_i, whereas $u_i(r_j)$ is the utility user i would have received had she been allocated the resource share r_j of user j instead. In an envy-free system (optimally fair) $\rho(\omega)$ equals 1. The closer the value is to 0 the more envy there is, and the more unfair the allocation scheme is.

The task of an economically healthy resource allocation scheme is to enforce both high efficiency and high fairness in the Nash equilibrium states of the game.

When constructing a mechanism to allocate resources in a computational market, it is therefore important to force users towards taking actions that yield one of these equilibrium state. In a system where a *Tragedy of Commons* behavior is possible no equilibrium states will ever be reached. In other words, it should not be possible to game (trick) the allocator for individual benefit at the cost of the overall health of the system in terms of fairness and efficiency. A mechanism that yields an equilibrium state in the presence of strategic users is said to be *strategy proof*. Likewise a software system architecture implementing an computational economy is *truth-telling* if users have an incentive to restrict their signaled and actual usage of a resource to their true needs. Further, it is *incentive-compatible* if users who have an incentive to perform a task either perform it themselves or transfer the incentive to a broker to perform the task on behalf of them. Incentive-compatibility is key to any system to avoid the Tragedy of Commons problem occurring, and it necessitates the deployment of transposable and commensurable entities, e.g. a currency.

Best Response Agent

A game theoretical analysis tries to model the behavior of players and make statements about optimal strategies and mechanisms enforcing certain global behavior

based on local rules. Strategies can be implemented on behalf of a player by an *agent*. One example of an agent that implements an optimal strategy to solve the resource allocation game just described is the *best response agent* presented in [20, 72]. Given a fixed budget and a pool of *divisable* resources allocated according to the proportional share mechanism described above, the best response agent finds the distribution of bids across resources that yields the highest utility for an individual player. The prior beliefs of the demand used by the agent to make its decision is the sum of all bids in the previous bidding cycle for all the available resources. Zhang [72] shows that there always exists a Nash equilibrium when the players' utility functions are strongly competitive, i.e. when there are at least two users competing for each resource. Furthermore, a tight efficiency bound of $\Theta(\frac{1}{\sqrt{(}m)})$ and an envy-freeness of $2\sqrt{(}2) - 2$ or approximately 0.828 in Nash equilibria with m players are theoretically deduced.

Chapter 3

Software

The research results of this thesis were obtained by implementing service-level management support for two Grid and cluster middleware toolkits. Three types of experiments were then performed. First, local simulations were run to test the algorithms against theoretical models, where both resource users and providers were simulated. Second, simulated users were run against providers deployed in the real Grid. Finally, real users and applications were run against the real Grid.

The SweGrid Accounting System (SGAS) was implemented as a Grid accounting system on top of the Globus Toolkit, and the Tycoon Grid Resource Allocator was implemented as a Grid market broker on top of the Tycoon market-based resource allocator. The general design of the two systems will be discussed below.

3.1 SweGrid Accounting System

We developed the SweGrid accounting system[1] to meet the quota enforcement needs of SweGrid, a national compute resource integrating 600 nodes at six HPC Centers across Sweden [57, 19, 58]. Resource quota is granted to research projects after peer review by the Swedish National Allocation Committee (SNAC). The quota can then be consumed by running jobs on any of the six participating sites. The main challenge was to consolidate the heterogeneous local accounting and security policies into one uniform accounting system capable of charging and enforcing allocations globally and in real-time. Our solution was to develop a Web services architecture based on a generic authorization policy framework capable of administering, storing, enforcing, and validating stakeholder policies at runtime. The stakeholders in SweGrid are the Grid application users, the resource providers, and the allocation authorities. Service-level management is carried out jointly by three services: the *Bank*, the *Logging and Usage Tracking Service* (LUTS), and the *Job Account Reservation Manager* (JARM). An overview of these services is shown in Figure 3.1.

[1] http://www.sgas.se

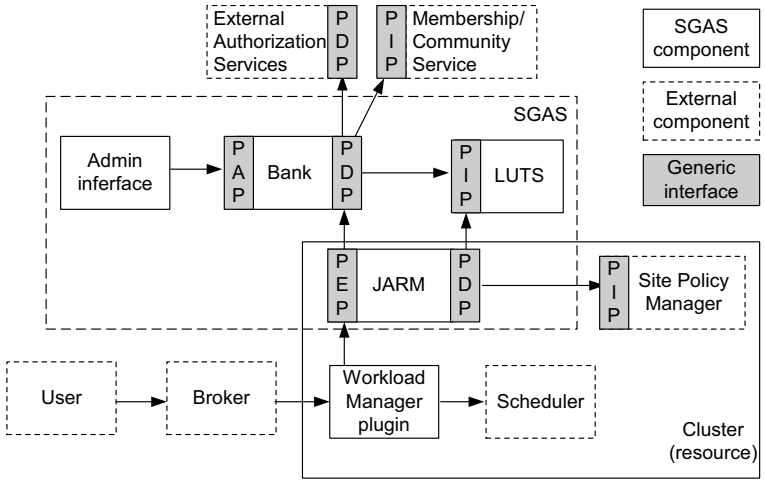

Figure 3.1: SGAS Components Overview.

Bank

The Bank service manages resource quota on a project and user basis. An account in the bank is created for each research project and then the principal investigator of the project can add all the users who should be allowed to submit jobs that are allowed to consume the project quota. The Bank service can be queried for available funds in an account, and *holds* of parts of the funds can be issued and then charged. The Bank thus represents the allocation authority stakeholder.

LUTS

The Logging and Usage Tracking Service allows off-line accounting after the jobs have run, and off-loads the bank from storing detailed logging and auditing records. The LUTS can be queried by all stakeholders as a means to making allocation and authorization decisions based on previous history.

JARM

The Job Account Reservation Manager component integrates the local accounting system and job manager infrastructure with the SGAS services. JARM implements the resource provider policies to enforce service levels. Currently only a binary

service-level model is implemented where the job either runs as a full-priority task if enough quota is available in the Bank, and as a low-priority task if the quota has been exceeded.

3.2 Tycoon Grid Resource Allocator

Tycoon

Tycoon is a market-based resource allocation system allowing resource shares to be auctioned out proportionally to users' bids [43, 44, 42]. In short it implements the resource allocation game and the best response agent as described in Section 2.2. Furthermore, Tycoon implements resource virtualization as described in Section 2.1. A user i bids on a resource by specifying a total bid size b_i and a bidding interval t_i. The bid is then calculated as $\frac{b_i}{t_i}$. If the total size of a resource is R, then r_i, the total amount of resource allocated to user i over a period P, is

$$r_i = \frac{t_i}{\sum_{j=0}^{n-1} \frac{b_j}{t_j}} R$$

If q_i is the amount of the resource consumed by user i in period P, then i pays at a rate of:

$$s_i = \min(\frac{q_i}{r_i}, 1)\frac{b_i}{t_i}$$

Note, that payments are made, as common for a utility, per time unit on a continuous basis. A resource exposes its price y as an indication of the price as the sum of all the current bids.

To determine the best response function yielding a distribution of bids across a set of resources given a total budget and the resource prices, Tycoon implements the best response algorithm [20] that solves the following optimization problem for a user: from a set of n resources pick the set $\{x_{i,j}...x_{i,n}\}$ that

$$\text{maximizes } U_i = \sum_{j=1}^{n} w_{i,j} \frac{x_{ij}}{x_{ij}+y_j} \text{ subject to } \sum_{j=1}^{n} x_{ij} = X_i, \text{ and } x_{ij} \geq 0. \quad (3.1)$$

where U_i is the utility of user i across a set of resources, $w_{i,j}$ is the preference of machine j as perceived by user i (for example the CPU capacity of the machine), $x_{i,j}$ is the bid user i should put on host j, y_j the total of all current bids or the price of host j, and finally X_i is the total budget of user i.

The prior beliefs of the demand used as input to the algorithm are represented by the y_j values, which are reported by all resource auctioneers after each completed bidding and accounting cycle, typically once a minute. However, users are allocated their appropriate shares instantaneously after bidding, which they can do at any time.

Grid Market

As part of our investigation of service-level management in Grid systems we developed a Grid broker on top of Tycoon (see Figure 3.2), which allows Grid HPC users to prioritize their jobs in an incentive-compatible way by transferring Tycoon credits to the broker. The broker receives credits from the user and automatically creates local virtual host accounts to execute the job on the resources picked by the best response algorithm described in Equation 3.1. The jobs run on each host at a service level determined by the Tycoon allocator proportional to the bid determined by the best response algorithm. The actual enforcement of the service level is done by the virtualization engine in Tycoon, which is Xen. An important addition to Tycoon that we also developed was to provide a tool for Grid users to predict future prices of resources in order to make better decisions on how much money should be spent on a resource to get a certain performance level.

The user interface of the broker uses the Nordugrid ARC *meta-scheduler* [62] which in turn is based on the Globus Toolkit [24], both extensively deployed in production Grid systems worldwide.

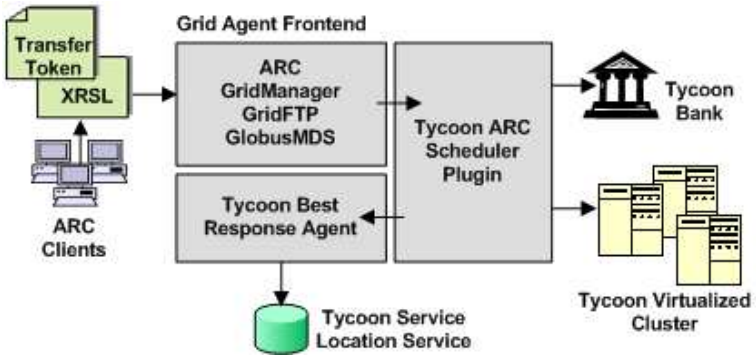

Figure 3.2: Tycoon Grid Market Architecture.

Our Grid market broker also performs a number of job related tasks on behalf of the user, and it is important to note that these tasks are all performed as a result of the user transferring additional money to the broker to maintain the incentive-compatible properties of Tycoon in the Grid market. Some of the tasks we added to the broker are enumerated here:

- **Job Payments.** A Grid user can pay for her jobs by attaching a transfer token to the job submission. The transfer token is receipt of a credit transfer from the user account to the Grid broker account. The token maps the Grid identity to a Tycoon bank account user identity. The token can also be issued by a 3rd party to clients who don't have any Tycoon components installed, and thereby use the token as a gift certificate. More commonly though the token

will be created as part of the job submission process on the client side. This design allows the broker to also utilize the full VO-authorization management support provided by the Grid job manager, a.k.a. the gatekeeper. It could be seen as a combination of identity based authentication, policy-based VO authorization and then finally capability based authorization in the Tycoon layer.

- **Price Prediction.** Future prices, performance estimates, at certain guarantee levels are communicated to the user in order to give guidance as to how much a job may cost.
- **Job Boosting.** A job that is running slower than first anticipated and that is not likely to meet its deadline can be boosted with initial funds without resubmitting the job.
- **Job Snapshots.** It is hard to tell from a generic infrastructure perspective, how close the job is to completing and whether it is therefore likely to meet its deadline. We therefore added an interface allowing users to get snapshots of their output files while the job is still running.
- **Job Stage-In, Stage-Out**. Input files are seamlessly transferred from the user to the compute node that was selected to run the job, and output files are gathered and transferred back to the user when a job has completed.
- **Multijob Support**. If multiple jobs are to be run at the same time it is preferrable to submit them all at once and let the best response algorithm take care of the optimal distribution and funding of them on each host. We therefore provide support for submitting one Grid job with different inputs for each individual compute node subjob.
- **Runtime Setup**. We use the YUM[2] installer to automatically provide a wide range of installation packages that may optionally be installed on demand before the job is run, and thus customizing the compute node configuration easily for the specific application needs and dependencies.
- **Bank Account Isolation and Refunds**. Each Grid user using our broker gets a separate local bank account used to fund end refund jobs. This improves accounting and isolation of individual user jobs, and allows the Grid broker to maintain the Tycoon property that users only pay for what they use.
- **Virtual Machine Recycling**. A user can create at most one virtual machine per compute node at any point in time to avoid the user competing with itself, and creating a higher price of the resource than necessary. It further helps in terms of avoiding starvation problems on a machine, since there are physical memory limitations in the virtualization engine of maximum number

[2]Yellow dog Updater, Modified. http://linux.duke.edu/projects/yum/

of virtual machines that can be served. In general the more slices a machine can handle the better effect does the market approach have. However, there is also substantial overhead incurred when creating and starting up a new virtual machine and installing the runtimes, so we allow the user to reuse virtual machine runtimes between job submissions (but not scratch space), but only if the idle virtual machine was not *outcompeted* by other users in the meantime. The reason why we don't support scratch space reuse is that the VM reuse should be transparent and only be detectable by means of a perceived performance improvement.

- **Seamless Backend Integration**. In order to allow seamless backend deployment of the Tycoon Grid scheduler into any Grid middleware job submission infrastructure we provides the same command line interface as OpenPBS [3], one of the most common cluster job submission toolkits.

[3] Open Portable Batch System.http://www.openpbs.org

Chapter 4

Results

In this chapter, the paper contributions attached to the end of this thesis are summarized. The papers represent the evolution of approaches used to solve the service-level provisioning and enforcement problem discussed in Section 1.1. In Paper 3, the contribution from the work conducted as part of this thesis is limited to the results section and to performing the experiments. All other papers were authored as full parts of this thesis.

We also summarize our contribution to other publications, which were only co-authored or only indirectly related to the service-level problem addressed here. Finally we summarize related work and conclusions.

Various research projects funded parts of this work, including the Swegrid accounting project (Swedish Research Council), Enabling Grids for ESciencE (European Union), NextGrid (European Union), Globus (Globus Alliance), and Tycoon (Hewlett-Packard and Intel JIP).

The thesis author's contribution level is given within parenthesis in each paper headline.

4.1 Thesis Papers

Paper 1: A Service-Oriented Approach to Enforce Grid Resource Allocations (90%)

In this journal article[1] [58] we discuss the initial approach of enforcing global resource quotas on a project basis across the SweGrid machines. SweGrid is the Swedish national Grid resource comprising 600 compute nodes distributed across six High Performance Computing (HPC) Centers and interconnected with a 10Gbit/s WAN. Various research projects are allocated CPU quota by the Swedish National

[1]First edition published in the proceedings of the 2nd ACM International Conference on Service-Oriented Computing, New York City, USA, November 2004. Second edition published in the World Scientific International Journal on Cooperative Information Systems, September 2006.

Allocation Committe (SNAC), after a peer review of the scientific value of the project and its computational needs. Allocations are administered and renewed on a six-month basis. The problem we are addressing in this work is how the allocations can be enforced in real-time on all of the SweGrid machines in a coherent manner.

The problem is to a large extent a systems integration problem, in that all HPC centers already use their own resource management system and their own accounting and access control policies and tools. We therefore introduced an integration platform based on a service-oriented XML Web services architecture entirely written in Java. The architecture comprises a Bank service, responsible for enforcing the global resource quota and managing project accounts; a Logging and Usage Tracking service, for off-line usage analysis and post-accounting; and finally a Job Account Reservation Manager, which integrates the local site resource manager into the global accounting system.

The most important research contribution from this work is the policy-based access control system, which, at real-time, lets user, resource, and allocation authority policies determine whether a Grid job should be allowed to run on a resource and at what level of service. We call this solution soft real-time allocation enforcement, because resources may not want to strictly refuse access if the quota has been exceeded, but instead downgrade the priority of the job. This model extends the state-of-the art in that a binary service-level is provisioned based on usage history and centrally allocated grants. A higher level of fairness is thus achieved, and problems like denial-of-service attacks and job starvation can be resolved.

Paper 2: Service Level Agreement Requirements of an Accounting-Driven Grid (100%)

In this technical report[2] [56] we discuss the requirements obtained after studying the first production deployment of the accounting system presented in [58]. We more specifically focus on how electronic contracts, a.k.a. Service Level Agreements, can be used to address some of the shortcomings of the existing system.

An enhanced, agent and policy-driven architecture is proposed, where the service levels are determined and enforced in a continuous and automatic way based on mutually signed contracts. The contracts represent a user capability as well as a resource provider obligation, and can thus be used as the basis for access control and service-level configuration.

The main contribution of this paper to the research presented in this thesis is the mapping of typical Grid user requirements to an agent-based, contract-driven architecture. The first insight gained from the SweGrid accounting system [58], was that it was very flexible to customize policies of all components, but determining what those policies should be quickly became a non-trivial task for a human actor.

[2]Published in the NADA TRITA technical report series at the Royal Institute of Technology, Stockholm, Sweden, September 2005.

Agents could thus use contracts embodying user and provider preferences to optimize user utility, or provider profit and utilization by automatically setting these policies.

Paper 3: The Design, Implementation, and Evaluation of a Market-Based Resource Allocation System (50%)

In this manuscript[3] [45] we introduce Tycoon, a market-based resource allocation system for large-scale networks like PlanetLab and the Grid. Tycoon allocates virtualized slices on hosts proportional to user bids. The main focus of this paper is to evaluate and benchmark the economic properties of the Tycoon resource allocation algorithms in a real cluster environment through a set of experiments. We study efficiency, based on the sum of the utilities across all users, a.k.a. as social welfare; and fairness, defined as the level of envy-freeness. Envy in turn is defined as the ratio between the maximum utility a user would get from another user's allocation and the utility of the allocation obtained. An optimally fair system would thus have an envy-freeness value of 1.

It is shown in our experiments that the Tycoon proportional share allocation is more efficient than an equal-share allocation algorithm like the one used in PlanetLab when slicing individual resources in shares. It is further shown that the Best Response algorithm implemented in Tycoon to distribute bids optimally across hosts yields a higher efficiency than other load balancing algorithms. In terms of fairness our experiments were not able to show as clear trends due to noise in the live cluster contributing to increased envy.

The results in this paper confirms previous simulation results and also shows how Tycoon can be used to dynamically trade off winner-takes-it-all and equal-share allocation algorithm properties. In essence, the higher the statistical variance on the bids is, the closer the Tycoon algorithm is to the winner-takes-it-all scheme. If the variance is 0 it is equivalent to an equal-share algorithm.

Paper 4: Market-Based Resource Allocation using Price Prediction in a High Performance Computing Grid for Scientific Applications (90%)

In this conference paper[4] [59] we combine the results from the previous three papers by providing a Grid resource market for HPC users. This market is further supported by a suite of prediction models and tools to allow users to spend their money more efficiently in the market to meet their requirements.

Our solution is to integrate a Grid meta-scheduler and resource manager with Tycoon. We thus maintain the cross organizational VO-supported PKI security

[3] Manuscript prepared for publication at Hewlett-Packard Laboratories, Palo Alto, USA, May 2006.

[4] Published in the proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France, June 2006

model and the support for high-volume data transfers to stage in and out jobs to compute nodes seamlessly. At the same time we leverage the economically efficient and fair Tycoon model including the Best Response scheduler and the proportional share allocator. The integration is achieved by two means, a) a transfer token used as a lightweight contract simulating a 'gift certificate' to purchase resource shares, b) a broker receiving the transfer token attached to the jobs to be submitted, which funds and executes the jobs according to the Best Response bidding algorithm.

The experimental results were obtained by running a Bioinformatics application, from SweGrid, in a cluster managed by the Tycoon Grid Market. It was shown that a continuous service level (as opposed to the binary one in SweGrid) could be offered proportional to the funding of the job. The account management is also simplified in our Grid Market, as the local accounts are created on demand and dynamically configured to match the service level purchased. Finally, rights delegation is seamless as it only involves transferring Tycoon credits between user accounts, and resources get credits when users run jobs that in turn can be used to submit jobs. Therefore, our Grid Market has the desirable property of offering a closed-loop sharing of resources among peers, true to the foundational idea of the Grid.

4.2 Additional Publication Contributions

Contribution 1 to 5 are co-authored papers and 6 is a lead-authored technical report.

Contribution 1 (10%)

The Global Grid Forum Open Grid Services Infrastructure (OGSI) specification [66] introduces many of the fundamental integration concepts that the SGAS work is based on. We contributed the XML rendering of that specification.

Contribution 2 (90%)

A conference version of Paper 1 was presented in [57]. It contains some additional Fuzzy Logic experiments and it is based on an earlier Web service integration platform. Paper 1 also contains some lessons learned from deploying the solution presented in [57] in SweGrid.

Contribution 3 (50%)

The Bank service of SGAS is presented in some more detail in the conference paper [19]. The Bank was implemented by a collaborator, but the core Web services infrastructure, and the access control and policy framework was contributed as part of this thesis. The overall design of the Bank was also a collaborative effort.

Contribution 4 (10%)

The SGAS authorization framework was contributed to the Globus Toolkit, and it is the foundation for extended work presented in the workshop publication [61]. Our authorization framework, in turn, borrows many concepts from the XACML architecture [1] and the GGF Authorization Working Group model [47].

Contribution 5 (10%)

SGAS provides a testbed for authorization management rights delegation, in the conference paper [60]. This work is also based on the authorization policy framework developed as part of SGAS, and extends it by integrating a 3rd-party authorization engine as a policy administration and decision point.

Contribution 6 (100%)

In the technical report [55] a philosophical view of the Grid is presented. The main contribution is to relate the concept of Ontologies in the Philosophy of Science community to the use of Ontologies in Computer Science in general and in Service Level Agreement protocols in particular. Ontologies play an important role in policy definition and embodies the universe of discourse used by agents to optimize the users' utility based on their preferences. The discussion in this report shows that work as early as Aristotle had striking similarities to the use of Ontologies today.

4.3 Related Work

Related work fall into three categories; first, systems that focus on the accounting aspect of the problem; second, general purpose computational economies; and finally Grid market systems. These categories can be related to our work with SGAS, Tycoon and the Tycoon Grid market respectively.

The DataGrid Accounting System (DGAS) [32] was an early approach to create a closed-loop accounting system for the LHC Grid at CERN, capable of exchanging virtual Grid credits for computational resource time. The project focussed mostly on providing an economic infrastructure for exchanging credits, but did not provide any price setting mechanism, like the one implemented in Tycoon. Furthermore, it did not take the integration approach used by SGAS, which made it difficult to deploy in Swegrid, without completely replacing the existing accounting and job submission infrastructure used by the different HPC sites. The GridBank project [4] took a similar approach to SGAS in that only a single call-out to a bank is necessary to verify the availability of funds to execute the job on the requested resource. They also took a similar approach to our Grid market by attaching a cheque-like token to the job-submission request to pay for the job. It, however, lacks the policy customization infrastructure of SGAS, allowing different resources to easily

implement different policies for running and charging for external jobs. SGAS also implements account holds which can be seen as soft reservations of a portion of the account balance, where jobs are only charged for the amount of resources actually consumed. A similar hold approach is implemented in the Gold accounting system [38], which also has expiring account quotas similar to SGAS. Gold did, however, not take the standards-based Web services architecture approach central to the design of SGAS, which also made it harder to integrate with an arbitrary local HPC site accounting system. Neither GridBank nor Gold have any price-setting mechanisms nor the same flexible authorization framework implemented in our work. Additional related accounting approaches can be found in [64, 37, 34].

Spawn [68], was one of the first implementations of a computational market, and Tycoon is an incarnation and evolution of many ideas presented in that work. Tycoon, in essence, extends Spawn by providing a best response agent for optimal and incentive-compatible bid distribution and host selection, and by virtualizing resources to give more fine-grained control over QoS enforcement. Tycoon also offers a more extensive price prediction infrastructure. However, the general, continuous bid and proportional share auction architecture is largely the same. Bellagio [50] uses a centralized allocator called SHARE. SHARE uses a centralized combinatorial auction allowing users to express preferences with complementarities. Solving the NP-complete combinatorial auction problem results in an optimally efficient allocation. The price-anticipating scheme in Tycoon is decentralized, i.e. runs an auction at every single host, and does not explicitly operate on complementarities. The efficiency in Tycoon may thus not be as high but all the overhead and computational complexities of combinatorial auctions, as well as the issues with strategic users gaiming the mechanism is avoided [45]. Related computational economy approaches are described in [54, 48, 29, 9, 63, 13].

Faucets [39] is a framework for providing market-driven selection of compute servers. Compute servers compete for jobs by bidding out their resources. The bids are then matched with the requirements of the users by the Faucets schedulers. Adaptive jobs can shrink and grow depending on utilization and prioritization. QoS contracts decide how much a user is willing to pay for a job. The main difference to our work is that Faucet does not provide any mechanism for price setting. Further, it has no banking service, use central server based username-password mechanisms, and does not virtualize resources. G-commerce [70] is a Grid resource allocation system based on the commodity market model where providers decide the selling price after considering long-term profit and past performance. It is argued and shown in simulations that this model achieves better price predictability than auctions. However, the auctions used in the simulations are quite different from the ones we use in our work. The simulated auctions are winner-takes-it-all auctions and not proportional share, leading to reduced fairness. Furthermore, the auctions are only performed locally and separately on all hosts leading to poor efficiency across a set of host. In Tycoon the best response algorithm ensures fair and efficient allocations across resources. An interesting concept in G-commerce is that users get periodic budget allocations that may expire, which could be useful

for controlling periodic resource allocations (as exemplified by our SGAS work) and to avoid price inflation. The price-setting and allocation model differs from our work in that resources are divided into static slots that are sold with a price based on expected revenue. The preemption and agile reallocation properties inherit in the bid-based proportional share allocation mechanism employed in our system to ensure work conservation and prevent starvation is, however, missing from the G-commerce model. Additional Grid Market models are described in [14, 71, 11].

4.4 Conclusions

Our work with the SweGrid Accounting System advances the state-of-the art of academic production Grid systems for High Performance Computing tasks by providing real-time quota enforcement across the Grid governed by a flexible policy framework. It thereby improves the overall fairness in the system. However, it only enforces two levels of service, and it does not provide any price-setting mechanisms. Furhtermore, it is very complex to manage all the policies manually without some broker or agent layer between users or providers, and the accounting system. The quota allocation model fits the SweGrid SNAC, and US NRAC periodic central allocation schemes, but it does not promote fast low-burden entry for new users.

Tycoon, addresses all of these problems by implementing a market for virtualized computational resources, allowing any service levels to be configured proportional to a user's bid and inversely proportional to the demand of the resources. Our main contribution in this thesis is hence the merge of the Tycoon market mechanisms with the Grid, thus creating a market appropriate for hosting both academic and commercial Grid applications. Dynamic host account creation and configuration according to purchased service levels, transfer of incentive compatible job tokens, and a combined identity and capability-based authorization model were all important parts of our solution.

4.5 Future Work

Tycoon implements a spot market, in order to quickly adapt the prices to the demand, and to allow important tasks to preempt currently running lower-priority tasks. However, these features come at the cost of less predictability and reduced guarantees of service levels. To address this issue we are working on enhanced prediction techniques to estimate future demand and give users tools to budget their future resource requirements more efficiently.

We would also like to investigate the combination of spot and reservation markets (such as derivative markets, e.g. options) for Tycoon, as well as contract brokers guaranteeing service levels, and offering discounts (paying penalties) if the promised level of service was not delivered.

4.6 Acknowledgments

First and foremost I would like to thank Bernardo Huberman and Kevin Lai, at Hewlett-Packard Laboratories in Palo Alto, both for their invaluable technical insights and feedback on my work, and for their continuous support to allow me to extend my stay at HP Labs to complete my work. I am also very grateful to all the technical support and contributions from Olle Mulmo at the Royal Institute of Technology in Stockholm related to the Grid security work in this thesis. Peter Gardfjell from Umeå University co-designed and co-authored the SGAS system with me and made great contributions to the Bank service. I would also like to thank his advisor Erik Elmroth for his help on finalizing the SGAS publications. Finally, I would like to thank my own advisor Lennart Johnsson, and Lars Rasmusson for their feedback.

Bibliography

[1] A. Anderson, A. Nadalin, B. Parducci, D. Engavatow, H. Lockhart, M. Kudo, P. Humenn, S. Godik, S. Abderson, S. Crocker, and T. Moses. eXtensible Access Control Markup Language (XACML) Version 1.0. Technical report, OASIS, 2003.

[2] Jorge Andrade and Jacob Odeberg. HapGrid: a resource for haplotype reconstruction and analysis using the computational Grid power in Nordugrid. *HGM2004: New Technologies in Haplotyping and Genotyping*, April 2004.

[3] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (ws-agreement). Technical report, Global Grid Forum, 2005.

[4] A. Barmouta and R. Buyya. Gridbank: A grid accounting services architecture (gasa) for distributed systems sharing and integration. In *Int. Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice, France, 2003. IEEE.

[5] F. Berman, G Fox, and A.J.G. Hey, editors. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[6] S. Blake, D. Black, M. Carlson, E. Davis, W. Zheng, and W. Weiss. Rfc 2475: An architecture for differentiated services. Technical report, IETF, 1998.

[7] Diana Bosio, James Casey, Akos Frohner, Leanne Guy, Peter Kunszt, Erwin Laure, Sophie Lemaitre, Levi Lucio, Heinz Stockinger, Kurt Stockinger, William Bell, David Cameron, Gavin McCance, Paul Millar, Joni Hahkala, Niklas Karlsson, Ville Nenonen, Mika Silander, Olle Mulmo, Gian-Luca Volpato, Giuseppe Andronico, Federico DiCarlo, Livio Salconi, Andrea Domenici, Ruben Carvajal-Schiaffino, and Floriano Zini. Next-generation eu datagrid data management services. In *Proceedings of Computing in High Energy and Nuclear Physics*, La Jolla, CA, USA, March 2003.

[8] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Rfc 2205: Reservation protocol (rsvp) version 1 functional specification. Technical report, IETF, 1997.

[9] Brent N. Chun and Philip Buonadonna and Alvin AuYoung and Chaki Ng and David C. Parkes and Jeffrey Shneidman and Alex C. Snoeren and Amin Vahdat. Mirage: A Microeconomic Resource Allocation System for SensorNet Testbeds. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors*, 2005.

[10] Vaughn Bullard, Bryan Murray, and Kirk Wilson. An introduction to wsdm. Technical report, OASIS, 2006.

[11] Rajkumar Buyya, Manzur Murshed, David Abramson, and Srikumar Venugopal. Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm. *Software: Practice and Experience (SPE) Journal*, 35(5):491–512, April 2005.

[12] Germano Caronni, Tim Curry, Pete St. Pierre, and Glenn Scott. Supernets and snHubs: A Foundation for Public Utility Computing. Technical Report TR-2004-129, Sun Microsystems, 2004. URL `http://research.sun.com/techrep/`.

[13] Anthony Chavez, Alexandros Moukas, and Pattie Maes. Challenger: a multi-agent system for distributed resource allocation. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 323–331, New York, NY, USA, 1997. ACM Press. ISBN 0-89791-877-0.

[14] Li ChunLin and Li Layuan. A two level market model for resource allocation optimization in computational grid. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 66–71, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-019-1.

[15] Karl Czajkowski, Ian Foster, Carl Kesselman, Volker Sander, and Steven Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. *Lecture Notes in Computer Science*, 2537:153–183, 2002.

[16] A. Dan, E. Davis, R. Kearney, A. Keller, R.P. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and Y.A. Web services on demand: Wsla-driven automated management. *IBM Systems Journal*, 43, 2004.

[17] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide Area Visual Supercomputing. *International Journal of Supercomputer Applications*, 10:123–130, 1996.

[18] Boris Dragovic, Keir Fraser, Steve Hand, Tim Harris, Alex Ho, Ian Pratt, Andrew Warfield, Paul Barham, and Rolf Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, 2003. URL `citeseer.ist.psu.edu/dragovic03xen.html`.

[19] Erik Elmroth, Peter Gardfjell, Olle Mulmo, and Thomas Sandholm. An ogsa-based bank service for grid accounting systems. In Jerzy Wasniewski, editor, *Lecture Notes in Computer Science: Applied Parallel Computing. State-of-the-art in Scientific Computing*. Springer Verlag, 2004.

[20] Michal Feldman, Kevin Lai, and Li Zhang. A Price-Anticipating Resource Allocation Mechanism for Distributed Shared Clusters. In *Proceedings of the ACM Conference on Electronic Commerce*, 2005.

[21] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[22] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. NAhrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proceedings of the International Workshop on Quality of Service*, 1999.

[23] I. Foster, A. Roy, V. Sander, and L. Winkler. End-to-end quality of service for high-end applications. Technical report, Argonne National Laboratory, 1999.

[24] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP'05: Proceedings of International Conference on Network and Parallel Computing*, volume 3799, pages 2–13. LNCS, Springer-Verlag GmbH, 2005.

[25] Ian Foster, Carl Kesselman, Jeffrey Nick, and Steven Tuecke. Grid services for distributed system integration. *Computer*, 7:37–46, March 2002.

[26] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organization. *International Journal of Supercomputing Applications*, 15(3), 2001.

[27] Ian Foster and Carl Kessleman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[28] Ian Foster and Carl Kessleman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[29] Yun Fu, Jeffrey Chase, Brent Chun, Stephen Schwab, and Amin Vahdat. SHARP: An Architecture for Secure Resource Peering. In *ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[30] S. Graham, A. Karmarkar, J. Mischkinksy, I. Robinson, and I. Sedukhin. Web services resource 1.2. Technical report, OASIS, 2005.

[31] Sven Graupner, Jim Pruyne, and Singhal Sherad. Making the Utility Data Center a Power Station for the Enterprise Grid. Technical Report HPL-2003-53, Hewlett-Packard Laboratories, 2003. URL `http://www.hpl.com/techreports/2003`.

[32] A. Guarise, R. Piro, and A. Werbrouck. Datagrid accounting system - architecture - v1.0. Technical report, EU DataGrid, 2003.

[33] Garrett Hardin. The Tragedy of the Commons. *Science*, 162:1243–1248, 1968.

[34] V. Hazelwood, R. Bean, and K. Yoshimoto. Snupi: A grid accounting and performance system employing portal services and rdbms back-end. 2001.

[35] Joseph Hellerstein, Kaan Katricioglu, and Maheswaran Surendra. An Online, Business-Oriented Optimization of Performance and Availability for Utility Computing . Technical Report RC23325, IBM, December 2003.

[36] R. Housley, W. Ford, W. Polk, and D. Solo. Rfc 2459: Internet x.509 public key infrastructure and crl profile. Technical report, IETF, 1999.

[37] S. Jackson. Qbank: A resource management package for parallel computers. Technical report, Pacific Northwest National Laboratory, Washington, USA, 2000.

[38] S. Jackson. The gold accounting and allocation manager, 2004. http://sss.scl.ameslab.gov/gold.shtml.

[39] Laxmikant V. Kale, Sameer Kumar, Mani Potnuru, Jayant DeSouza, and Sindhura Bandhakavi. Faucets: Efficient resource allocation on the computational grid. In *ICPP '04: Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)*, pages 396–405, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2197-5.

[40] Katarzyna Keahey, Karl Doering, and Ian Foster. From Sandbox to Playground: Dynamic Virtual Environments in the Grid. In *Grid 2004: Proceedings of the 5th International Workshop in Grid Computing*, Pittsburgh, PA, USA, November 2004.

[41] J. Kephart and D.M. Chess. The Vision of Autonomic Computing.

[42] Kevin Lai. Markets are Dead, Long Live Markets. *SIGecom Exchanges*, 5(4): 1–10, July 2005.

[43] Kevin Lai, Bernardo A. Huberman, and Leslie Fine. Tycoon: A Distributed Market-based Resource Allocation System. Technical report, arXiv, 2004. http://arxiv.org/abs/cs.DC/0404013.

[44] Kevin Lai, Lars Rasmusson, Eytan Adar, Stephen Sorkin, Li Zhang, and Bernardo A. Huberman. Tycoon: an Implemention of a Distributed Market-Based Resource Allocation System. Technical Report arXiv:cs.DC/0412038, HP Labs, Palo Alto, CA, USA, December 2004.

[45] Kevin Lai and Thomas Sandholm. The design, implementation, and evaluation of a market-based resource allocation system. Technical Report Manuscript for Publication, Royal Institute of Technology and Hewlett-Packard Labs, Stockholm, Sweden, May 2006.

[46] D. Lamanna, J. Skene, and W. Emmerich. SLAng: A Language for Defining Service Level Agreements. In *Proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS03)*, 2003.

[47] M. Lorch and D. Skow. Authorization Glossary. Technical report, Global Grid Forum, 2004.

[48] Thomas W. Malone, Richard E. Fikes, Kenneth R. Grant, and Michael T. Howard. Enterprise: A Market-like Task Scheduler for Distributed Computing Environments. In Bernardo A. Huberman, editor, *The Ecology of Computation*, number 2 in Studies in Computer Science and Artificial Intelligence, pages 177–205. Elsevier Science Publishers B.V., 1988.

[49] K. Nahrstedt, H. Chu, and S. Narayan. QoS-aware resource management for distributed multimedia applications. *Journal on High-Speed Networking* , December 1998.

[50] Chaki Ng, Philip Buonadonna, Brent N. Chun, Alex C. Snoeren, and Amin Vahdat. Addressing Strategic Behavior in a Deployed Microeconomic Resource Allocator. In *Proceedings of the 3rd Workshop on Economics of Peer-to-Peer Systems*, 2005.

[51] Martin J. Osborne. *An Introduction to Game Theory*. Oxford University Press, July 2002.

[52] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[53] Christos H. Papadimitriou. Algorithms, Games, and the Internet. In *Symposium on Theory of Computing*, 2001. URL `citeseer.ist.psu.edu/papadimitriou01algorithms.html`.

[54] Ori Regev and Noam Nisan. The Popcorn Market: Online Markets for Computational Resources. In *Proceedings of 1st International Conference on Information and Computation Economies*, pages 148–157, 1998.

[55] Thomas Sandholm. The philosophy of the grid: Ontology theory - from aristotle to self-managed it resources. Technical Report TRITA-NA-0532, Royal Institute of Technology, Stockholm, Sweden, September 2005. http://www.pdc.kth.se/ sandholm/trita/SandholmOntologyV2.pdf.

[56] Thomas Sandholm. Service level agreement requirements of an accounting-driven computational grid. Technical Report TRITA-NA-0533, Royal Institute of Technology, Stockholm, Sweden, September 2005. http://www.pdc.kth.se/ sandholm/trita/TRITA-SLA.pdf.

[57] Thomas Sandholm, Peter Gardfjell, Erik Elmroth, Lennart Johnsson, and Olle Mulmo. An ogsa-based accounting system for allocation enforcement across hpc centers. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 279–288, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-871-7.

[58] Thomas Sandholm, Peter Gardfjell, Erik Elmroth, Lennart Johnsson, and Olle Mulmo. A Service-Oriented Approach to Enforce Grid Resource Allocations. *International Journal of Cooperative Information Systems*, 2006. (to appear).http://www.worldscinet.com/ijcis/ijcis.shtml.

[59] Thomas Sandholm, Kevin Lai, Jorge Andrade, and Jacob Odeberg. Market-based resource allocation using price prediction in a high performance computing grid for scientific applications. In *HPDC '06: Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, June 2006.

[60] Ludwig Seitz, Erik Rissanen, Thomas Sandholm, Babak Sadighi Firozabadi, and Olle Mulmo. Policy administration control and delegation using xacml and delegent. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, November 2005. http://pat.jpl.nasa.gov/public/grid2005/index.html.

[61] Frank Siebenlist, Takuya Mori, Rachana Ananthakrishnan, Liang Fang, Tim Freeman, Kate Keahey, Sam Meder, Olle Mulmo, and Thomas Sandholm. The globus authorization processing framework, April 2005. http://lotos.site.uottawa.ca/ncac05/index.html.

[62] O. Smirnova, P. Erola, T. Ekelöf, M. Ellert, J.R. Hansen, A. Konsantinov, B. Konya, J.L. Nielsen, F. Ould-Saada, and A. Wäänänen. The NorduGrid Architecture and Middleware for Scientific Applications. *Lecture Notes in Computer Science*, 267:264–273, 2003.

[63] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):048–063, 1996. ISSN 1066-8888.

[64] W. Thigpen, J. Hacker, L. McGinnis, and B. Athey. Distributed accounting on the grid. Technical report, Global Grid Forum, 2001.

[65] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. IETF RFC 3820. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile, 2004. http://www.ietf.org/rfc/rfc3820.txt.

[66] Steven Tuecke, Karl Czajkowski, Ian Foster, Jeff Frey, Steven Graham, Carl Kesselman, Tom Maquire, Thomas Sandholm, David Sneling, and Peter Vanderbilt. Open Grid Services Infrastructure (OGSI) Version 1.0. Technical report, Global Grid Forum, 2003.

[67] Hal R. Varian. Equity, Envy, and Efficiency. *Journal of Economic Theory*, 9: 63–91, 1974.

[68] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and W. Scott Stornetta. Spawn: A Distributed Computational Economy. *Software Engineering*, 18(2):103–117, 1992. URL `citeseer.nj.nec.com/waldspurger91spawn.html`.

[69] Von Welch, Ian Foster, Carl Kesselman, Olle Mulmo, Laura Pearlman, Steven Tuecke, Jarek Gawor, Samuel Meder, and Frank Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. In *Proceedings of the 3rd Annual PKI R&D Workshop*, 2004.

[70] Rich Wolski, James S. Plank, Todd Bryan, and John Brevik. G-commerce: Market formulations controlling resource allocation on the computational grid. In *IPDPS '01: Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, page 10046.2, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-0990-8.

[71] Lijuan Xiao, Yanmin Zhu, Lionel M. Ni, and Zhiwei Xu. Gridis: An incentive-based grid scheduling. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, page 65.2, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2312-9.

[72] Li Zhang. The efficiency and fairness of a fixed budget resource allocation game. *Lecture Notes in Computer Science*, 3580:485–496, 2005. ISSN 0302-9743.

Part II

Papers

International Journal of Cooperative Information Systems

A SERVICE-ORIENTED APPROACH TO ENFORCE GRID RESOURCE ALLOCATIONS

THOMAS SANDHOLM

Department of Numerical Analysis and Computer Science and PDC, Royal Institute of Technology, SE-100 44 Stockholm, Sweden

PETER GARDFJÄLL, ERIK ELMROTH

Department of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden

OLLE MULMO, LENNART JOHNSSON

Department of Numerical Analysis and Computer Science and PDC, Royal Institute of Technology, SE-100 44 Stockholm, Sweden

Received (Day Month Year)
Revised (Day Month Year)

We present the SweGrid Accounting System (SGAS) – a decentralized and standards-based system for Grid resource allocation enforcement that has been developed with an emphasis on a uniform data model and easy integration into existing scheduling and workload management software.

The system has been tested at the six high-performance computing centers comprising the SweGrid computational resource, and addresses the need for soft, real-time quota enforcement across the SweGrid clusters.

The SGAS framework is based on state-of-the-art Web and Grid services technologies. The openness and ubiquity of Web services combined with the fine-grained resource control and cross-organizational security models of Grid services proved to be a perfect match for the SweGrid needs. Extensibility and customizability of policy implementations for the three different parties that the system serves (the user, the resource manager, and the allocation authority) are key design goals. Another goal is end-to-end security and single sign-on, to allow resources to reserve allocations and charge for resource usage on behalf of the user.

We conclude this paper by illustrating the policy customization capabilities of SGAS in a simulated setting, where job streams are shaped using different modes of allocation policy enforcement. Finally, we discuss some of the early experiences from the production system.

Keywords: OGSA;Grid;Accounting;HPC.

1. Introduction

Advances in network technology, in addition to the more distributed and collaborative nature of todays research projects, have prompted high-performance computing (HPC) centers to improve the ease of use of their resources to a larger and more dispersed user base, as well as responding to the need for unified access procedures to collections of resources from multiple administrative domains. As a result, monolithic and esoteric systems, albeit more performance tuned, have had to make way for ubiquitous and open, standards-based solutions. It has become feasible to integrate the centers into Grids that enable flexible resource sharing and load balancing across organizational boundaries.[1]

Virtualization across management and security policy domains not only leads to a complex resource negotiation situation, but also makes it harder to track usage and enforce allocations. It is the latter issue that we address in this paper. We have developed an accounting system to enforce nationally allocated resource quotas across six HPC centers in Sweden.

Key requirements on the accounting system include: soft real-time allocation enforcement based on resource usage collected from existing site schedulers; coordinated quota management across all clusters; uniform usage retrieval; policy negotiation and customization between user, resource, and allocation authority; and finally a flexible, policy-driven, and standards-based authorization framework.

Our contribution and differentiator against existing accounting systems is fourfold: (1) we provide a decentralized accounting solution based on standard, open protocols in compliance with the proposed Open Grid Services Architecture (OGSA),[2,3] (2) we facilitate 3-party (user, resource, allocator) policy customization, (3) our system is non-intrusive to existing local site accounting systems and end-user tools, and thus offers light-weight deployment, and (4) all accounting components are governed by a scalable cross-organizational authorization framework based on state-of-the-art Web services protocols.

The paper is organized as follows: Section 2 contains an overview of recent standardization efforts in the field of wide area distributed computing relevant to Grid computing. The SweGrid network and its accounting requirements are outlined in Section 3. In Section 4, we present some existing accounting systems and architectures, and we discuss why they do not meet the SweGrid needs. Section 5 describes the SweGrid accounting system design and Section 6 the implementation. Section 7 presents some results from simulations of reservations against our system, and discusses some feedback obtained from the production system. Finally, in Section 9, we summarize our contribution and our future research and development plans.

This paper is an extended and revised version of Ref. 4.

2. OGSA and Web Services

OGSA was developed in order to solve the complex task of sharing and integrating fine-grained heterogeneous resources distributed across security domains in a wide

area network. The architecture combines the elaborate control mechanisms of mainframe systems with ubiquitous Web and Internet technologies. Key concepts include virtualization and discovery of resources based on service-oriented interactions. It can be seen as the Web Services Architecture (WSA) applied to Grid computing.[5] Another key aspect of OGSA is the management of distributed state, such as discovery, introspection, notification, and lifetime management. Although Web services, by design, are considered stateless for scalability and decoupling reasons, state needs to be managed to control shared resources in an application and client agnostic way and thus enabling interoperable state-aware interactions. In highly dynamic systems, there is often a trade-off between fine-grained control of state, and strict enforcement of decoupling. The core Web services specifications such as SOAP and WSDL do not fully address the problem of managing application state, but they provide extensibility features that may be leveraged by other specifications.[6,7] The often quoted REST architecture for interacting with resources only solves the problem partially by putting the burden of maintaining state on clients or client agents, and is further targeted towards large-grain hypermedia transfers, and thus very limited in its scope.[8] REST is a very similar approach to the one taken by Web services workflow languages, such as BPEL4WS.[9] Request-broker-influenced specifications, however, address client agnostic fine-grained resource state sharing.[10]

Our work is based on the broker model, because it also fits better with existing programming language technology. Web services protocols all use XML as a foundational building block, and therefore are convenient for self-describing, document-centric interactions (as opposed to the less flexible API-centric model) often used in large-scale integration environments with little or no control over the participating parties implementation policies.

In OGSA-based Web-services environments, the complexity of setting and applying policies to optimize the user quality of service, as well as resource utilization leads to the need for Service Level Agreement (SLA) management. Agreement, and negotiation protocols such as Global Grid Forum's emerging WS-Agreement, and FIPA's Contract Net protocols are example technologies addressing that need.[11,12] Providing complete SLA lifecycle management support in a computational Grid is outside the scope of the work presented here, but we discuss how such a solution may be designed in Ref 13.

3. SweGrid

SweGrid is a national computational resource, initially joining together one cluster at each of six high-performance computing centers across Sweden, and currently comprising 600 computing nodes. The clusters located at the individual sites are interconnected with the 10 Gb/s GigaSunet network. The sites also operate several other resources for computation and storage, and they have developed their own security and accounting systems over time to serve local needs and the requirements following from different sources of funding. SweGrid job submissions

are currently performed using the Globus Toolkit or the NorduGrid job submission tools, interfacing cluster-level schedulers at the local sites.[14,15] Compute time on the SweGrid resources are allocated to research projects by the Swedish National Allocations Committee (SNAC), akin to the NRAC (National Resource Allocations Committee) in the US. Projects within the Swedish science community and with the appropriate needs and promising research may apply for SNAC allocations. The allocations are currently made in node hours, and the decisions are made after a scientific peer-review process evaluating the research proposals. Prior to interconnecting the HPC centers in a Grid, allocations were targeted at individual clusters, and the prospective research participants would have to acquire valid user accounts at each of the centers at which quotas were awarded to be able to run their jobs. This manual and static allocation thus not only caused sub-optimal job-to-resource mappings, but further led to large administrative overhead. Hence, SNAC is now allocating quotas to the SweGrid as a whole. However, this has a large impact on how accounting is done, because it is thereby not sufficient to just do local site accounting, and quota enforcement. The allocation enforcement must be coordinated across all sites, and the sites must be able to produce uniform usage records that comply with each other.

Thus a real-time enforcement solution is required that allocates resources to projects in a fair manner while taking current user policies, resource policies, and allocation-authority policies into account. A resource may, for instance, allow jobs lacking sufficient quota to be run and put in a low priority queue if the current utilization is low. A user, on the other hand, may only want to execute a job if there are sufficient funds, and finally some allocation authorities (e.g., SNAC or project leaders) may not allow jobs to go through from certain users who have used up a large chunk of a common project quota. This three-way negotiation needs to be flexible enough to allow various parties to configure their systems according to local policies.

Even though SweGrid currently consists of a fairly homogenous compute farm with similar middleware installations, it is expected that both the hardware and software solutions will evolve and become distinctly heterogeneous in the future, as more resources are added. The SweGrid accounting system must hence be non-intrusive to the existing systems, i.e., easy to deploy or plug into existing infrastructure, without replacing the local accounting and scheduling systems.

4. Grid Accounting

Cluster-targeted scheduling systems as well as operating systems commonly have built-in accounting systems to track resource usage. However, they often assume a homogenous run-time environment, and they lack standard and uniform ways to obtain and represent information from several heterogeneous clusters. Thus there has been a strong need for Grid accounting systems that integrate local accounting solutions similarly to the way Grid meta-schedulers and co-allocation managers

coordinate, and administer job submissions across several schedulers. Some general issues that need to be solved by distributed accounting systems on the Grid, including the need for a standard usage record format, are outlined in Ref. 16.

In the European Data Grid Accounting System (DGAS), users need to pay for resources that they use in a virtual currency called Grid Credits.[17] Resources earn Grid Credits by offering their services to users, thus stimulating market-economy driven resource sharing. All currency transactions are mediated by decentralized bank services. The implementation is tightly coupled to the DataGrid workload manager software, and thus hard to deploy without affecting the local cluster software environment.

GridBank is provided as an extension to the Globus job manager, and it calculates job cost based on standard XML usage records.[18] It is thus not as intrusive as DGAS, but it still requires modifications of a particular workload manager. An interesting feature of GridBank is that it makes use of decentralized Trade Servers to negotiate resource prices. GridBank is also modeled around an economy-driven workload management system utilizing resource price matrices.[19]

Neither GridBank nor DGAS are based on open, standard Grid protocols such as Web services or OGSA, thus limiting their prospective scope of interoperability with other Grid systems.

The Grid Economic Services Architecture (GESA) specified by the Global Grid Forum (GGF), presents an OGSA-based architecture using the concept of chargeable services.[20] When developing a service one may associate it with a cost that can be charged in a bank based on standard usage records. GESA is, hence, quite intrusive to the service since it requires the service interface to be changed in order to charge for its usage. Furthermore, GESA was designed to be orthogonal to the security model chosen, and does not address the security issues related to accounting.

SNUPI provides extensions to the Linux operating system, and it allows cluster usage data to be collected and stored in RDBMS databases, and then queried from user-friendly portal Web interfaces.[21] SNUPI is, however, not service-oriented and assumes a homogenous cluster environment.

QBank is a resource allocation management system developed for parallel computers.[22] Its successor, Gold, adds more advanced accounting features such as price quotes, funds transfers, and timestamped allocations.[23] Gold also allows role-based authorization and transaction journaling. Although it would fulfill most of the core accounting needs discussed in this paper, it is not developed using open, Grid, or Web services protocols, and is thus limited in its interoperability as well as cross-platform support. Its security model is also limited compared to our work.

5. SGAS Design

We have developed the SweGrid Accounting System (SGAS), with the aim of meeting the accounting needs of SweGrid presented in Section 3, and with a particu-

lar focus on shared quota enforcement across organizational boundaries, simplicity of deployment, and security.[24] The accounting system is fully transparent, or imposes only marginal additional requirements, to the end-users, allowing for a smooth transition into an accounting-enabled Grid. In this section, we present the design rationale of the various system components.

We start by describing the flexible authorization framework (Section 5.1). In addition to a bank service (Section 5.2), which provides most of the accounting functionality, there is a workload manager integration component (Section 5.4), and a usage tracking service (Section 5.3). Figure 1 shows an overview of SGAS.

The operational flow is as follows: a user submits a job (potentially via a brokering service) to a workload manager service running on the resource. (We make use of a generic term here to stress that SGAS is a generic system that can be integrated with more than a single software stack.) The resource integration component intercepts the request by way of a workload manager plugin, and it interacts with the bank to reserve sufficient quota. This interaction is further explained in Section 5.2 through 5.4.

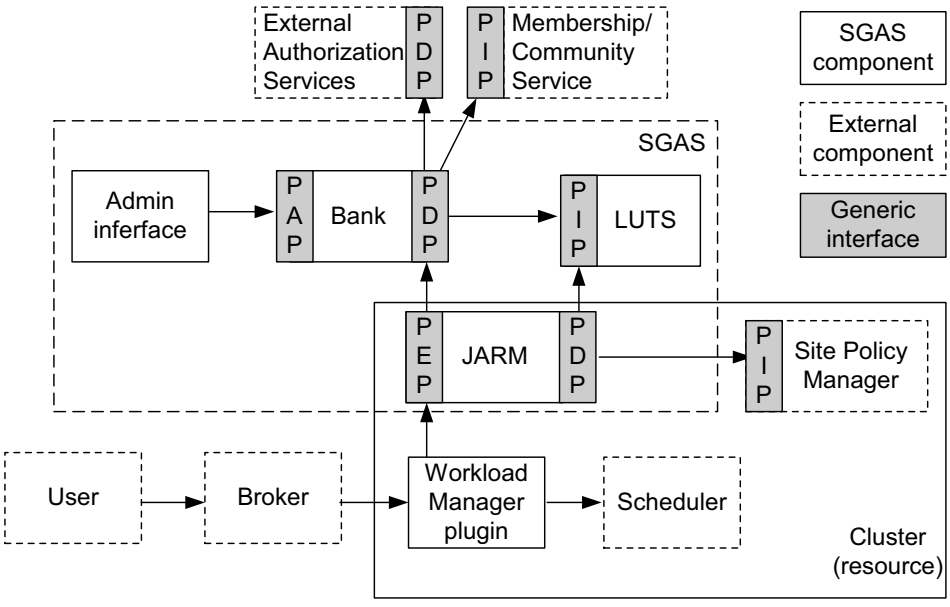

Fig. 1. SGAS Components Overview.

5.1. *Authorization Framework*

Three parties are involved in our accounting scenario: the user, the resource, and the allocation authority (front-ended by the bank). The system has multiple decision points at various levels, allowing for both policy overlay (combining policies from multiple sources when making a decision) and retention of local control. This allows us to honor the requirements of all three stakeholders, as well as facilitating decentralized control and system management.

We make use of the terminology and overall architecture as proposed by the Global Grid Forum working groups on Authorization Frameworks and Mechanisms, and OGSA Authorization.[25,26] We allow access permission policies to be specified in XACML (eXtensible Access Control Markup Language).[27] Figure 2 shows an example policy in XACML, governing what set of users may use a certain allocation in the bank. While the implementation makes use of XACML, we emphasize that the framework allows for any policy language understood by the pluggable authorization engines. To illustrate this we have successfully experimented with Delegent, an authorization service capable of rights management delegation (not supported in XACML) as an alternative back-end authorization service, or Policy Decision Point (PDP), for the bank.[28]

Multiple information providers are used in the system. The bank, for instance, may be configured to associate any user in a particular Virtual Organization (VO) with a particular account. To achieve this, external services such as VOMS and CAS may be used to gather membership evidence (Policy Information Point, PIP).[29,30]

The allocation authority adopts a delegated security model, controlled by policies that can be associated within each research project. The highest level of authorization authority is the national allocations committee, which allocates quotas to projects/VOs. On a project level, the principal investigator (PI) can specify additional policies through the Policy Administration Point (PAP), to allow various project members to use the quota. To enable flexible self provisioning, the PI may additionally give away a possibly restricted subset of its own management privileges to other members.

Allocation requests and decisions are authenticated and integrity protected by the use of XML digital signatures, and/or WS-SecureConversation.[30,31] In addition, when the resource contacts the bank, both the users delegated credentials associated with the requested job (made available by the workload manager plugin), as well as the resources own credentials, are used to authenticate the allocation request.

The resource checks the quota on a soft real-time enforcement basis. The check is soft (as opposed to strict) in that policies on the client, as well as on the resource, can allow the job to be run even if the allocation authority decides that sufficient quota is not available. This is a critical requirement from the HPC centers, because the allocations are done periodically (for 6 or 12 months) whereas user resource usage tends to be bursty (e.g., just before the scientists are to publish a paper, usage goes up). Additionally, the users may specify that they only want to execute long-running

jobs on resources that allow the jobs to complete within the available quota limit, and the resource may disallow jobs without enough available quota during peak utilization periods. Such usage-based allocation decisions can be made by querying the usage service, and by allowing the resource Policy Enforcement Point (PEP) to shortcut the bank authorization (modulo local site manager configuration), and overrule the final decision.

```
<Policy PolicyId="SweGridTestProjectPolicy"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
 <Target>
  <Subjects><AnySubject/></Subjects>
  <Resources><AnyResource/></Resources>
  <Actions><AnyAction/></Actions>
 </Target>
 <Rule RuleId="RequestHoldRule" Effect="Permit">
 <Target>
  <Subjects>
  <Subject>
   <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
         /O=Grid/O=NorduGrid/OU=pdc.kth.se/CN=ThomasSandholm
   </AttributeValue>
   <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                               AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
   </SubjectMatch>
  </Subject>
  <Subject>
   <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
         /O=Grid/O=NorduGrid/OU=cs.umu.se/CN=PeterGardfjell
   </AttributeValue>
   <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                               AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
   </SubjectMatch>
  </Subject>
  </Subjects>
  <Resources><AnyResource/></Resources>
  <Actions>
  <Action>
   <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
   <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"> requestHold
   </AttributeValue>
   <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
   </ActionMatch>
  </Action>
  </Actions>
 </Target>
 </Rule>
 <Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

Fig. 2. Example of an XACML-encoded bank account policy granting withdrawals for two project members.

5.2. *Bank*

The bank component is central to the design of SGAS. It implements coordinated quota enforcement across all the SweGrid sites. The component consists of three Web services, the Bank-, the Account-, and the Hold- services. The bank design is presented in some more detail in Ref. 33.

The Bank service is responsible for creating and locating Accounts, corresponding to a research-project allocation. The Account service hands out soft-state, lease-based fund reservations called Holds to authorized Account members. Account members may be added or removed, or their rights may be modified through XML-defined policies. When a Hold is created, a specified amount of the total quota or funds is locked, meaning it may not count towards other reservations or withdrawals (c.f. making reservations on a credit card). The Hold is further only accessable by the party creating the Hold, typically the resource. The Hold can be renewed (its lifetime extended) and it can be committed (released). A commit operation triggers an accounting transaction record and debits the Account that the Hold was held against. The amount reserved in a Hold does not have to match the committed amount, because they correspond to estimated vs. actual cost to run a job. It is up to the resource to decide whether a conservative overbooking reservation strategy should be applied to be sure that the job completes within the reservation time, or to be more optimistic and reserve a smaller amount that potentially can be renegotiated if the job did not manage to complete in time.

The cost and the allocations are expressed in a virtual currency, Grid Credits, and may thus be mapped into any physical resource-specific cost. Typically, the cost is mapped directly to wall-clock time, because it makes it easy for the existing HPC centers scheduling infrastructure to enforce as well as to measure the quota. How physical costs should be mapped into Grid Credits is, however, decided by resource policy. A resource may, for instance, use a standard usage record and give the various containing attributes weights used to calculate the total cost. The typical wall-clock approach can thus be seen as giving the wall-clock attribute the weight of 1 and all other attributes the weight of 0. Another advantage of the wall-clock mapping is that it becomes intuitive for users to set a maximum wall-clock time attribute, which corresponds to the granted SNAC time, in their job specifications using, e.g., the Globus [14] Resource Specification Language (RSL). They thereby initiate an implicit in-blanco signing process with the resources.

Figures 3 and 4 shows the bank design, and a resource and bank interaction scenario. The interfaces shown should be seen as conceptual entities or roles of responsibility rather than programming language constructs. The interface technology used, typically WSDL or Java, depends on the distribution of the components. The bank is designed with a minimal set of data-centric operations to make it as easy as possible to interact with and to allow for future extensions. Security interfaces are clearly separated from application interfaces, allowing the security implementation to be customized or replaced without affecting the core bank implementation.

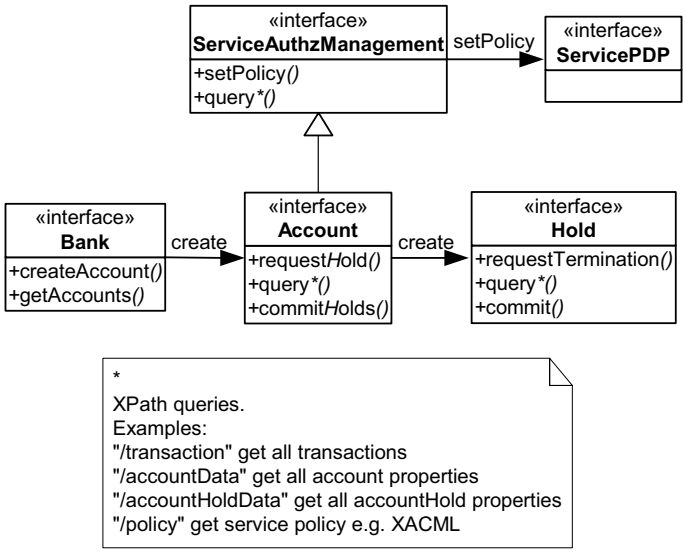

Fig. 3. Bank Interfaces.

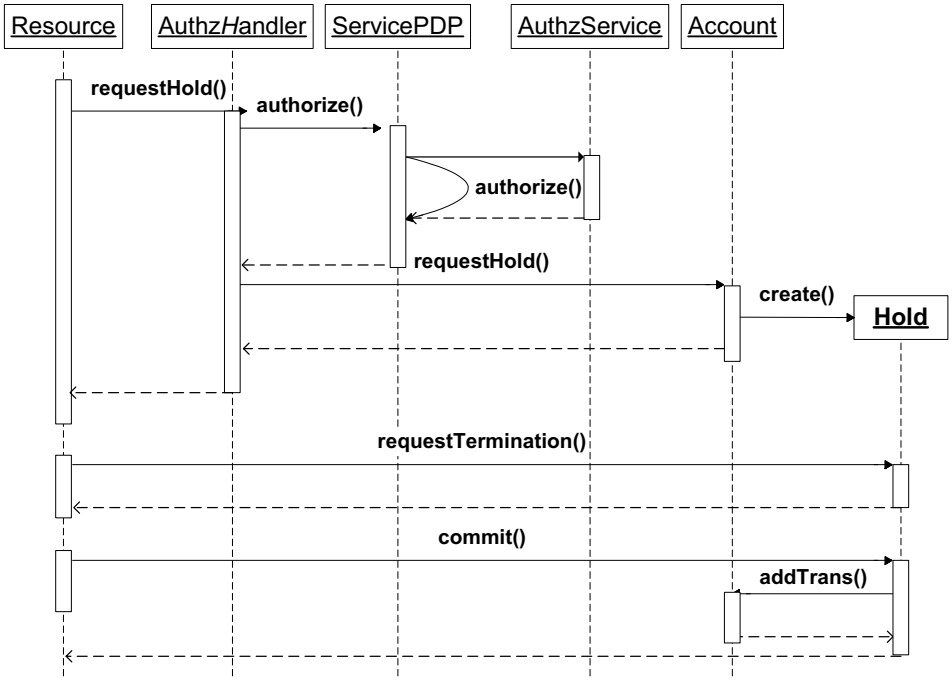

Fig. 4. Bank and resource interactions.

5.3. *LUTS*

The Logging and Usage Tracking Service (LUTS) is used to store usage records compliant with the GGF Usage Record (UR) XML format.[34] Depending on who should have access to the service, resources may share the same LUTS in order to allow users to query for detailed information regarding the resources consumed by their jobs across multiple sites. The query language is XPath-based and therefore very flexible and extensible.[35] LUTS is schema agnostic, which means that the UR may be extended with information, such as job tracking information, that a particular subset of resources and users understands without having to change or reconfigure LUTS. A batch of Usage Records may be logged at the same time to improve performance and scalability. The service builds on the same security infrastructure as the other SGAS services allowing, for instance, dynamic access control permissions to be set up specifying who is allowed to query or publish data in the service, and allowing message payloads to be encrypted and/or signed.

5.4. *JARM*

The Job Account Reservation Manager (JARM) component is responsible for integrating local cluster systems into SGAS. JARM intercepts a job submission and calculates the estimated cost of the job based on, for example, the users job specification (using RSL in our case), and current system load. It then contacts the appropriate Account, which is either specified in the RSL by the client or alternatively searched for in the Bank. A Hold (account reservation) is created with the estimated cost, and the timeout of the Hold is set to the estimated duration of the job plus a margin. The resource also lets the Bank know whether overdrafts are accepted, a policy that may be requested by the client. If the Hold was created successfully, JARM lets the local workload manager continue with the job submission; otherwise, an error is generated and logged.

After the job has completed, JARM collects the usage information, converts it into the standard GGF UR format, logs it into LUTS, calculates the actual cost of the job, and commits the Hold (which is then destroyed). All this typically happens in batch mode, asynchronously in regard to the job submission, to induce as little overhead as possible to the user-perceived response time. In addition, it allows for higher throughput at moments of peak load. A Site Policy Manager implementation can easily be customized for particular workload managers and site policies. Note that JARM shields the Site Policy Manager from knowledge about the bank system (see Figure 1). A generic NorduGrid/ARC[15] Site Policy Manager has currently been implemented.

SGAS is mainly concerned with allocation enforcement, and because it is workload-manager agnostic, scheduling and brokering functionality is outside of its scope. However, we recognize that economic brokering algorithms based on a thorough analysis of economic models and business needs belongs to the future of both scientific and industrial Grids, and that the use of cyber money as well

as virtual money is going to be a future requirement. We therefore provide plug points for calculating, setting, and publishing the price in the Site Policy Manager component. Note that this does not mean that the resources need to decide on appropriate prices in isolation to the rest of the system. Trading and pricing services as described in Ref. 36, and Ref. 17 may, for instance, be used. The use of cyber money or real money in conjunction with Grid Credits, is in SGAS best done at the allocation authority level, where Bank services may charge real money for filling up accounts with quotas.

6. SGAS Implementation

In this section we present experience gained from implementing the accounting solution described in the previous section.

6.1. *Implementation Approach*

For interoperability reasons, the SGAS design is based on the latest Grid and Web services protocols. In our implementation, we go one step further by reusing toolkits implementing these standards. The general approach taken was to compose the solution from standards-based toolkit primitives, as opposed to re-implementing low-level middleware or communication libraries. Apart from the obvious advantages of developing complete applications more rapidly and following the latest specifications closer, we also safeguard our solution against protocol changes in the standards, and we can leverage the interoperability testing done by the protocol implementers. For example, the initial implementation implemented all services compliant to the OGSI specification whereas the newest version uses WSRF.

Reuse is done on three levels: development environment (e.g., build system), compile-time environment (APIs), and run-time environment (application server containers and system-level services). The first two are commonly applied by most projects, whereas the third is more common in the software industry than in academia. We focus our discussion here on run-time environment reuse in a Grid environment.

6.2. *Container Framework*

The Globus Toolkit (GT) provides a Java-based container implementation of the OASIS Web Services Resource Framework (WSRF) protocols, a realization of the OGSA model.[14,37] Both WSRF and GT are designed as a set of primitives that can be freely mixed, composed, extended, and embedded. WSRF facilitates cross-language interoperability, whereas the GT Java container provides a consistent, portable programming model. Below, we first summarize how the various WSRF concepts are leveraged in SGAS, and we then continue with describing how the GT container features are used to achieve this.

Soft-state management (server-side managed, client-lease controlled state) is commonly applied in both the Internet and Grid networks, and it is a fundamental

component of WSRF (WS-ResourceLifetime). We control expiration and extension of Holds using the WS-ResourceLifetime soft-state protocol. Service property introspection (with its associated query and notification framework), as a means to minimize brittle APIs for flexible information retrieval, is another key component of WSRF (WS-ResourceProperties). We use this concept to query transaction records in the Bank and Usage Records in LUTS, and to get notifications when Holds are about to expire. WSRF recommends a factory pattern to create stateful resources in a uniform manner, which we apply. The factory pattern is used to create both Account, and Hold resources.

GT allows code to be plugged into the container on three different levels: message, operation, and back-end storage. Message interceptors are mainly used for service-orthogonal functionality, such as transaction management and security. In SGAS, a GT-provided authorization-interceptor plugin is used to implement the interaction between the PEP in JARM and the PDP in the bank. Furthermore, mutual authentication, message encryption, and message signing, are all carried out by GT transparently to the application code in message handlers using generic implementations of WS-SecureConversation, XML-Encryption, and XML-Signature, respectively.[32,38,31]

Operation providers allow decoupled implementations of parts of service interfaces. A service implementation is typically made up of a set of toolkit-supplied operation providers, and one or many application-specific providers. The providers are specified at deployment time, and thus promote a development model based on composition of primitives. All SGAS services (the bank services and LUTS) are made up of operation providers. LUTS is composed of GT supplied operation providers exclusively, and thus does not have any application-specific code or APIs. The unique behavior of LUTS is achieved by a back-end storage and query plugin that leverages an XML database implementation (eXist [39]) and XPath (Xalan [35]) as a query engine. GT operation providers implement soft-state management, service creation, notification and inspection of service state transparently to the SGAS code.

6.3. *Systems Integration and Scalability*

Although the general design is to introduce as few new APIs as possible, there are a number of high-level APIs that may be used as a means to integrate SGAS with other systems. We expect other Grid services to be built on similar core OGSA fabric, and infrastructure components in the future, such as WSDL and WSRF. This in itself offers a baseline for low-level API interoperability that could be used, e.g., by generic management tools. As an example, Globus resource property browsers and monitors were used to manage the Bank and LUTS services. Further, the high-level Bank and Policy management APIs provided by SGAS and expressed in WSDL, serve as a public integration point to other accounting and authorization components. The Bank APIs are discussed more thoroughly in Ref. 33.

Simplicity and scalability are central to the SGAS design. SGAS should be able to scale down to very small, as well as to large-scale nation- and Grid-wide deployments. As a means to scale up, the load can be balanced across many Bank and LUTS services. Additionally, charging and logging are done in batches with intervals customized to the overall system load.

6.4. *Toolkits and Standards*

We summarize the toolkits and standard protocols used to implement central features of SGAS in Table 1. SunXACML is used in the bank as a standard, self-contained PDP engine, which checkpoints policies to the eXist database.[40] Some schedulers already have support for the GGF UR format, but for others we provide an XSLT style sheet transformer based framework to simplify SGAS integration at local sites.[35]

Table 1. Toolkits and Standards.

Toolkit	SGAS Feature	Standards Implemented
Globus	Service state management	WSRF
Globus	Mutual authentication, credential delegation	GSI profile of WS-SecureConversation
Globus	Payload integrity, and privacy	XML-Signature, XML-Encryption
eXist	XML database (for policy and service state)	XML:DB
Xalan	Query engine, stylesheet transformation	XPath, XSLT
SunXACML	XACML PDP	XACML
Axis	Web services engine	SOAP, WSDL
SGAS	SGAS Usage Records	GGF XML Usage Record

7. Illustration of Policy Customization Capabilities

In order to illustrate the policy customization capabilities of SGAS and their effects, we built a simulation framework aimed at measuring job turnaround times. The usage pattern that was simulated consisted of an allocation authority that periodically adds new resource allocations to a bank account, and account members that continuously consume their allocations by submitting jobs (thereby creating and committing account reservations).

The behavior was studied for different policy configurations and two separate job flows. First, a fair flow is a stream of job submissions that is produced by a user, who does not try to make more reservations than is allocated to him/her within a given time period. Second, an unfair flow is a stream of job submissions that is produced by a user, who tries to make reservations of twice the allotment. The unfair flow can be shaped using various policies and overdraft protection algorithms to optimize fairness and resource utilization. In case fairness is the sole objective, allocation enforcement can be carried out strictly which would disallow any quota-exceeding jobs in the unfair flow. In the simulations, we employed soft allocation

enforcement, which allows jobs to run even though the user account is overdrawn, thus representing a trade-off where resource utilization is increased at the expense of fairness. If the reservation fails due to an overdraft violation, then there is a penalty in job execution time, simulating the job being put in a low priority queue by the scheduler. The degree of enforcement "softness" may be controlled through an overdraft limit policy, which we base on access control policy (XACML) rule conditions that may be set by account administrators.

It should be noted that the XACML policies used are mere examples of viable algorithms that may easily be applied using the SGAS customizability, and extensibility features. That is, the aim here is not to show an optimal algorithm, but rather to illustrate how a certain policy (overdraft protection in this case) can be implemented in SGAS. However, the simulated policy of tracking overdrafts and causing jobs exceeding their quota to run slower using the algorithm described below is indeed employed in the SweGrid production system today, and the results are hence relevant to the current use of SGAS.

Table 2 lists the configuration used in the simulation runs. Fair and unfair submit intervals denote the duration between successive job submissions for a well- and ill-behaved user, respectively. For each job, an account reservation for 60 credits is placed (corresponding to the 60 second job duration). Allocation interval refers to the time between two successive allocations, which in the SNAC case typically is one month (see Section 3 and 5.1). Allocation amount is the size of each such allocation. Overdraft penalty is the extra execution time added due to an overdraft (that is not allowed by policy). This simulates a scenario where jobs within the allocation limit are started immediately, whereas quota-exceeding jobs are penalized with an extra 60 seconds of queue time.

All the simulations can be reproduced, and the source code can be obtained by downloading the SGAS Open Source distribution.[24]

Table 2. Simulation Setup.

Simulation Property	Time (s)
Fair Submit Interval	10
Unfair Submit Interval	5
Reservation Amount (Job Duration)	60
Allocation Amount	300
Allocation Interval	50
Overdraft Penalty	60

Figure 5 compares the job turnaround times of a fair and an unfair flow disallowing all overdrafts. The area below the flow curves represents the accumulated execution time of all jobs in the flow. Thus, the larger the area is, the worse is the turnaround, and the higher is the aggregated penalty time. The peaks of the flows resulted from overdraft violations. Thus, the thinner the peaks are, the closer is the

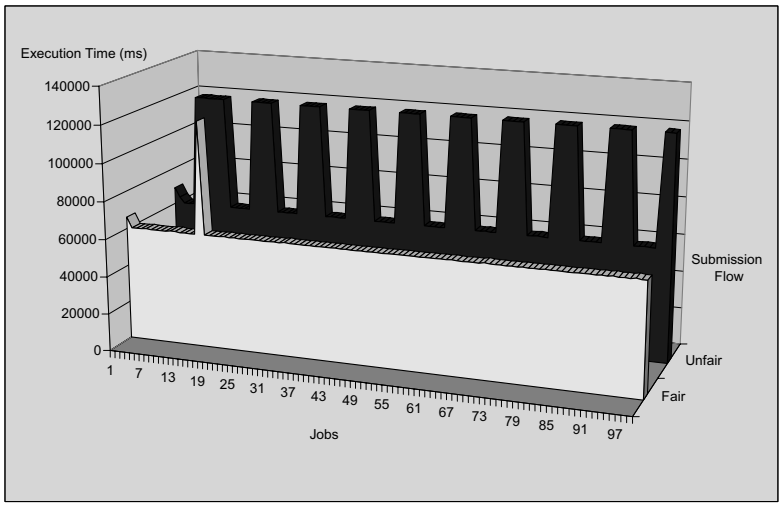

Fig. 5. Submission flow simulation using fair and unfair flows (zero overdraft limit).

user to the actual allocation. Note that the fair flow was shaped to get minimum job turnaround time by avoiding overdrafts. The occasional peak at the beginning of the fair flow simulation was caused by the fact that the reservations and the periodic allocations were not started simultaneously, and thus the first allocation happened too late to avoid an overdraft. Over time, however, the allocations and reservations were synchronized. The periodicity of the peaks in the unfair flow corresponds to the available quota running low shortly before the new allocations are granted.

The authorization framework used by SGAS (Section 5.1) allows account owners to set policies regulating access to their accounts with XACML policies. An example policy condition is given in Figure 6. The actual value of the XACML attribute sgas:overdraw:percent:requested is calculated as:

$$\frac{a_s + a_r + r_r}{a_t},$$

where a_s is the allocation spent, a_r is the allocation reserved, r_r is the requested reservation, and a_t is the total allocation. The value may hence be less then 100%. In that case reservations must not completely exhaust the total allocation available in order to be successful. The condition can be associated with any rule like the ones exemplified in Figure 2.

In our simulations we tested three different policies allowing 25, 50, and 75 per-

```
<Condition FunctionId=
   "urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal">
  <Apply FunctionId=
     "urn:oasis:names:tc:xacml:1.0:function:integer-one-and-only">
    <EnvironmentAttributeDesignator  AttributeId= "sgas:overdraw:percent:requested"
                                     DataType="http://www.w3.org/2001/XMLSchema#integer"/>
  </Apply>
  <AttributeValue DataType= "http://www.w3.org/2001/XMLSchema#integer">
     175
  </AttributeValue>
</Condition>
```

Fig. 6. Example of an overdraft policy allowing 75% overdraft.

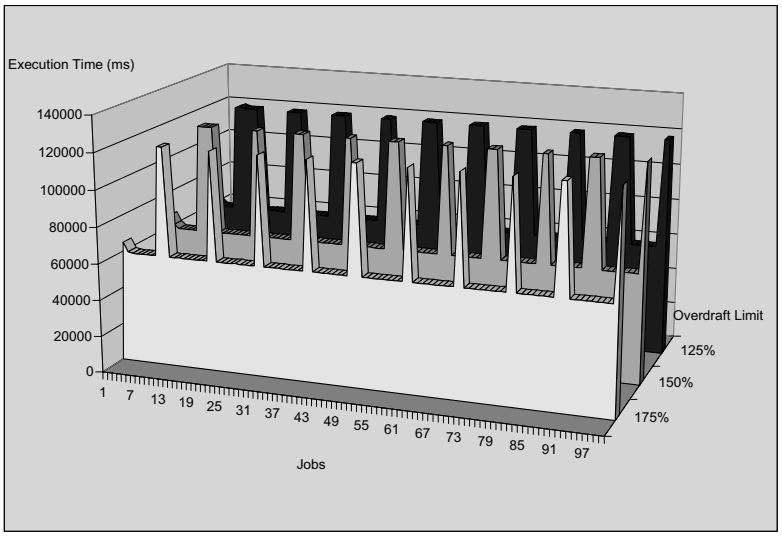

Fig. 7. An unfair submission flow simulation using policy-based overdraft protection (25%, 50%, and 75% overdraft limits).

cent overdraft against the unfair job reservation flow. The results can be seen in Figure 7. We note that turnaround in the 25% policy case is close to the unregulated unfair flow, whereas, the more permissive, 75% policy is getting closer to the turnaround of the fair flow.

8. Early Experiences

SGAS has been tested in the SweGrid production system since September 2004. After the original submission of this paper, SGAS was ported from OGSI to WSRF and Globus 4. One of the initial experiences gained from the production system was that some large-scale jobs requesting 100s of CPUs could cause the bank communication to become a bottleneck for the sites. We therefore allowed the sites to use transport-level security (SOAP over SSL) as an alternative to WS-Secure Conversation. The WSRF port made the Web container consume less memory (as a result of a cleaner separation between a Web service and the stateful resource being managed) and as a result solved some scalability problems we were experiencing in the persistency layer. We also implemented automatic phase outs of old accounts and LUTS records to cope with the production level load of SweGrid.

From a policy management point of view it is still a too manual of a task to distribute allocations to projects and too complicated for PIs to redistribute their quotas to project members. Further, the service-level differentiation offered at the resource sites is often too coarse grained (regular operation, or free pool low priority operation). The price setting is not dynamic enough and does not give users an incentive to use less powerful resource at less loaded times. SGAS has improved the fairness of resource sharing in SweGrid but more could be done to allow users who need to submit jobs urgently to pay more and get a higher priority.

To address some of these issues we have studied related economic theory applicable to computational markets. Tycoon, a proportional share market-based resource allocation system, provides a low-level infrastructure that can ensure both a high degree of economic efficiency in terms of social welfare, or aggregated total user utility as defined in Ref. 41, and a high degree of fairness in terms of envy-freeness, as defined in Ref. 42.[43] Current research focuses on integrating the Tycoon system with Grid environments typified by SweGrid.

As an alternative to a market-driven solution, we have also investigated a complementary resource allocation mechanism in Ref. 44, where we demonstrated the potential of a decentralized architecture for a Grid-wide fairshare scheduling system in SweGrid-like environments and usage scenarios. Here, job priorities are individually assigned to each job based on the deviation between allocated and consumed share of resources. The system, which preserves local site autonomy, enforces both locally and globally scoped share policies, allowing local resource capacity as well as aggregate VO capacity to be logically divided across different groups of users. The policy model is hierarchical and subpolicy definition can be delegated so that, e.g., a VO can distribute shares of its aggregate resource capacity across its projects, which in turn can divide their shares between project members. Notably, there is no need for a central coordinator as policies are enforced collectively by the resource schedulers. Each local scheduler adopts a Grid-wide view on utilization in order to steer local resource utilization to not only maintain local resource shares but also to contribute to maintaining global shares across the entire set of Grid resources.

We acknowledge that SGAS needs to be tested and validated in more diverse applications and Grid networks, and we hope that the recent inclusion of SGAS in the Globus toolkit distribution and an ongoing integration with the CERN LCG Grid in the EGEE project will give us data for future analysis of this kind.

9. Conclusions and Future Work

We have presented an architecture, and an implementation of an accounting system based on open, standard Grid and Web services protocols to solve the resource quota-enforcement needs of a national-scale Grid network. Easy non-intrusive deployment, and integration with pre-existing, local accounting solutions prompted the use of XML document-centric communication and transformations and the use of a minimal set of APIs. This design is apparent in the policy administration API allowing arbitrary XML-specified policies to be defined for allocation decision points with a single operation. Another example is the non-existing API between the workload manager and the JARM component. It is designed as a message interceptor, obtaining its required input via runtime context and environment settings.

A customizable security model based on multiple PDPs, and PIPs, but with a single PAP and PEP, makes it possible to easily add new authorization services without affecting the service usage.

The three-party policy negotiation design allows the resources to implement site-specific policies to optimize utilization and prioritize between users with different usage patterns and job-specification requirements. Furthermore, it enables allocation authorities such as SNAC, PIs, or individual project members to regulate the quota distribution according to dynamic policies.

The novel set of accounting features presented in this paper to solve the particular needs of SweGrid, and implemented in the SGAS system, do not exist in any other existing Grid accounting system to date. Although SGAS is primarily developed for SweGrid, it is based on open protocols, and has generic-enough functionality to be used in any Grid accounting setting.

The design is made general with respect to the type of mechanisms that are used for balancing load between resources or for achieving fairness between users. For example, the bank can be used in an environment driven by market-economy strategies where resources and resource brokers negotiate price and QoS agreements solving the supply and demand problem. It fits equally well into a more planned-economy model where the main aim is to achieve fairness between users, based on given allocations to users or projects.

Besides improving baseline functionality, scalability and robustness, we mainly intend to continue to improve this system in two directions: (1) more sophisticated pre-allocation mechanism to allow, for instance, SAML assertions to be used as quota cheques for a collection of jobs, and thus limiting the bank interaction overhead of individual jobs, (2) use of more elaborate negotiation protocols such as Contract Net and WS-Agreement to handle Service Level Agreement (SLA)

contract policing and obligation enforcement. With a more advanced negotiation protocol in place, we also intend to investigate soft computing, and game theory based decision-making procedures to automate SLA refinement.

10. Acknowledgements

We would like to thank our colleagues, Åke Sandgren, Lars Malinowsky, Michael Hammill, and Bo Kågström for their feedback on this work; and Leif Nixon, and Aleksandr Konstantinov for their help with the NorduGrid integration. We would also like to thank Babak Sadighi, Tomas Olsson, Ludwig Seitz, and Erik Rissanen for their work on integrating Delegent into our authorization framework. Finally, we would like to thank Martin Folkman for his work on developing SGAS administration tools.

This work has been supported by The Swedish Research Council (VR) under contracts 343-2003-953, 343-2003-954, and 621-2005-3667.

References

1. I. Foster, C. Kesselman (eds.). *The Grid: Blueprint for a New Computing Infrastructure* (Morgan Kaufmann, 1999).
2. I. Foster, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, H. Kishimoto, F. Maciel, A. Savva, F. Siebenlist, R. Subramaniam, J. Treadwell and J.V. Reich, The Open Grid Services Architecture, Version 1.0, (Global Grid Forum, 2004).
3. I. Foster, C. Kesselman, J.M. Nick and S. Tuecke, Grid Services for Distributed System Integration, in *Computer*. **35**(6)(2002), pp. 37–46.
4. T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O. Mulmo. An OGSA-based accounting system for allocation enforcement across HPC centers, in *ICSOC'04*, (ACM, 2004), pp. 279–288.
5. D. Booth, H. Haas, F. McCabe, E. Newcomber, M. Champion, C. Ferris and D. Orchard, Web Services Architecture, (W3C, 2003).
6. N. Mitra, SOAP Version 1.2 Part 0: Primer - Section 1.1, (W3C, 2003).
7. R. Chinnici, M. Gudgin, J. Moreau, J. Schlimmer and S. Weerawarana, Web Service Description Language (WSDL) Version 2.0 Part 1: Core Language - Section 2.8, (W3C, 2003).
8. R.T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Ph.D. Dissertation at the Information and Computer Science Department, University of California (Irvine, 2000).
9. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic and S. Weerawarana, Business Process Execution Language for Web Services Version 1.1., ed. S. Thatte, (Microsoft, IBM, Siebel Systems, BEA, SAP, 2003).
10. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling and P. Vanderbilt, Open Grid Services Infrastructure (OGSI) Version 1.0, (Global Grid Forum, 2003).
11. A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke and M. Xu, Web Services Agreement Specification (WS-Agreement), Draft, (Global Grid Forum, 2004).
12. *Contract Net Interaction Protocol Specification* (FIPA, 2003).

13. T. Sandholm, Service level agreement requirements of an accounting-driven computational grid. Technical Report TRITA-NA-0533, (Royal Institute of Technology, Stockholm, September, 2005).
14. I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems, in *IFIP International Conference on Network and Parallel Computing*. **3799** (Springer-Verlag LNCS, 2005), pp. 2–13.
15. O. Smirnova, P. Eerola, T. Ekelöf, M. Ellert, J.R. Hansen, A. Konstantinov, B. Kónya, J.L. Nielsen, F. Ould-Saad and A. Wäänänen, The NorduGrid Architecture and Middleware for Scientific Applications. in *Lecture Notes in Computer Science*, Vol. 2657 (Springer Verlag, 2003), pp. 264–273.
16. W. Thigpen, J. Hacker, L. McGinnis and B. Athey, Distributed Accounting on the Grid, (Global Grid Forum, 2001).
17. A. Guarise, R. Piro and A. Werbrouck, DataGrid Accounting System - Architecture - v1.0, (EU DataGrid, 2003).
18. A. Barmouta and R. Buyya, GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration, in *Int. Parallel and Distributed Processing Symposium (IPDPS'03)*, (IEEE, Nice, France, 2003).
19. D. Abramson, J. Giddy and L. Kotler, High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, in *Proc. Int. Parallel and Distributed Processing Symposium (IPDPS)*, (Cancun, Mexico, 2000), pp. 520–528.
20. S. Newhouse, Grid Economic Services Architecture, (Global Grid Forum, 2003).
21. V. Hazelwood, R. Bean and K. Yoshimoto, SNUPI: A Grid Accounting and Performance System Employing Portal Services and RDBMS Back-end. in *Linux Clusters: The HPC Revolution*, (Urbana/Champaign, USA, 2001).
22. S. Jackson, QBank: A Resource Management Package for Parallel Computers, Pacific Northwest National Laboratory, (Washington, USA, 2000).
23. S. Jackson, The Gold Accounting and Allocation Manager, (2004), http://sss.scl.ameslab.gov/gold.shtml.
24. SGAS, (2005), http://www.sgas.se.
25. M. Lorch and D. Skow, Authorization Glossary, (Global Grid Forum, 2004).
26. V. Welch, F. Siebenlist, D. Chadwick, S. Meder and L. Pearlman, Use of SAML for OGSA Authorization, (Global Grid Forum, 2004).
27. A. Anderson, A. Nadalin, B. Parducci, D. Engovatov, H. Lockhart, M. Kudo, P. Humenn, S. Godik, S. Abderson, S. Crocker and T. Moses, eXtensible Access Control Markup Language (XACML) Version 1.0., eds. S. Godik and T. Moses, (OASIS, 2003).
28. L. Seitz, E. Rissanen, T. Sandholm, B. Sadighi Firozabadi, O. Mulmo, Policy Administration Control and Delegation using XACML and Delegent. in *Proc. 6th IEEE/ACM Int. Workshop on Grid Computing* (Seattle, November, 2005).
29. R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lorentey and F. Spataro, VOMS, an Authorization System for Virtual Organizations. in *1st European Across Grids Conference* (Santiago de Compostela, February 13-14, 2003).
30. L. Pearlman, V. Welch, I. Foster, C. Kesselman and S. Tuecke, A Community Authorization Service for Group Collaboration. in *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks* (2002).
31. M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon, XML-Signature Syntax and Processing, eds. D. Eastlake, J. Reagle and D. Solo, (W3C, 2002).
32. G. Della-Libera, B. Dixon, P. Garg and S. Hada, Web Services Secure Conversation (WS-SecureConversation), eds. C. Kaler and A. Nadalin, (Microsoft, IBM, VeriSign, RSA Security, 2002).

33. E. Elmroth, P. Gardfjäll, O. Mulmo and T. Sandholm, An OGSA-based Bank Service for Grid Accounting Systems, in *Applied Parallel Computing. State-of-the-art in Scientific Computing. Lecture Notes in Computer Science*, Vol. 3732 (Springer Verlag, 2005), pp. 1051–1060.
34. S. Jackson and R. Lepro Metz, Usage Record – XML Format, (Global Grid Forum, 2003).
35. Xalan Java, (Apache Software Foundation, 2004), http://xml.apache.org/xalan-j.
36. R. Buyya, D. Abramson and J. Giddy, A Case for Economy Grid Architecture for Service Oriented Grid Computing, (Global Grid Forum, 2001).
37. S. Graham, A. Karmarkar, J. Mischkinksy, I. Robinson and I. Sedukhin, Web Services Resource 1.2, (OASIS, 2005).
38. T. Imamura, B. Dillaway and E. Simon, XML Encryption Syntax and Processing, (W3C, 2002).
39. eXist, (2005), http://exist.sourceforge.net.
40. Sun's XACML Implementation, (Sun Microsystems, 2004), http://sunxacml.sourceforge.net.
41. C. H. Papadimitriou. Algorithms, Games and the Internet, in *Symposium on Theory of Computing* (2001).
42. H.R. Varian. Equity, Envy, and Efficiency, *Journal of Economic Theory*. **9**(1974), pp. 63–91.
43. M. Feldman, K. Lai and L. Zhang, A Price-Anticipating Resource Allocation Mechanism for Distributed Shared Clusters, in *ACM Conference on Electronic Commerce* (June, 2005).
44. E. Elmroth and P. Gardfjäll. Design and Evaluation of a Decentralized System for Grid-wide Fairshare Scheduling, in *e-Science 2005. First IEEE Conference on e-Science and Grid Computing*, IEEE Computer Society Press, USA, pp. 221–229, 2005.

Service Level Agreement Requirements of an Accounting-Driven Computational Grid

Thomas Sandholm
Dept. of Numerical Analysis and Computer Science and PDC
Royal Institute of Technology
SE-100 44 Stockholm, Sweden
+46-8-7907811

Abstract
In this paper[1] we present the requirements of a national computing Grid. In particular we discuss the issues involved in managing complex policies of multiple stakeholders in such a large-scale, dynamic, and heterogeneous Grid. We also propose a Service Level Agreement (SLA) and agent-based architecture to address these issues. This work is a continuation of the work performed and experiences gained when we developed a Grid accounting system for the Swedish national Grid network, called SweGrid, which provides the foundation for the investigation presented here. We conclude that many SLA concepts fit very well within the SweGrid network to address some of the issues of the current system. Future work includes prototyping parts of the SLA framework and running simulations before eventually deploying it in the SweGrid production environment.

1 Introduction

In recent years, the Internet has had a virtually explosive growth in number of users, largely due to the fact that there has also been matching technological progress. Moore's law is often quoted when comparing CPU speed, but the advances in network performance have improved even more rapidly. As a result, we are now at a point where communication between computers distributed over large geographic areas almost matches the internal bus communication of PCs.

This trend, in conjunction with the ubiquity of the Internet and the price drop of high performance computing devices, prompted some researchers in the mid 90's to try to build virtual supercomputers operating across WANs like the Internet [1]. The initial results were very promising and this new architecture became known as the Grid to emphasize that computing power could be viewed as a utility akin to the electric power grid [2-4].

Today, the Grid has moved beyond the pure academic research projects, and the industrial involvement has gained momentum [5, 6]. For industrial innovators, the Grid offers a means to fulfill some long-sought dreams, such as charging for service usage, and outsourcing and automating management of high-end resources [7].

The main issue with the current Grid infrastructure is that, like the Internet, it operates on a best-effort basis. That is, there are no guarantees of delivered service quality. This mode of operation is sufficient when it comes to free information provisioning via, the World Wide Web, for example, but when it comes to delivering services or making computations to solve complex problems, for which a customer may have paid a large amount of money, new solutions are needed. Some low-level solutions already exist to provide differentiated services over Quality of Service (QoS)-enabled networks capable of guaranteeing offered bandwidth [8, 9] or CPU [10]. The biggest problem with these solutions is that the low-level infrastructure of the Grid is intrinsically very heterogeneous, and it

[1] This work was partly funded by The Swedish Research Council (VR) under contract 343-2003-953

thus requires designs of much higher abstraction and coarser granularity. The key to such designs is interoperability through standardization, which also serves one of the cornerstones of the Grid: ubiquity.

As a first step towards guaranteeing QoS, service usage needs to be tracked. This is typically done by Grid accounting systems [11-14] and it is elaborated on in our previous work [11]. The next step involves the ability to request and claim agreed upon QoS parameters, such as response time and price. This capability can be provided by setting up electronic contracts between the stakeholders embodying the agreed-upon service level. This kind of contract is referred to as a Service Level Agreement (SLA). It is a non-trivial task to make sure that SLA contracts in a Grid environment are adhered to, because they often span many software and hardware abstraction layers. Coordinated monitoring and management software is thus vital for such a system. Furthermore, policing contracts is not simply a matter of making sure that users will not exceed their QoS grants, and that resources fulfill their promises. Such a static view may in fact lead to worse resource utilization and less service offered to the user. This is where policies are introduced to allow the stakeholders to state their preferences. The inherent large scale of the Grid and the disparity of its users make policy management a complex task subject to automation.

Automation and reasoning about complex and even contradictory and partly unknown policies is an issue that has occupied the mobile agent and soft computing communities. Some preliminary efforts are now underway to merge their resulting technology with the Internet [15] and the Grid [16].

In this paper, we discuss the requirements of an accounting enabled national Grid: SweGrid, set up to serve the Swedish research community. We then propose an architecture based on SLA management and agent-driven reasoning to use as a starting point for addressing these requirements.

Section 2 discusses Service Level Agreement technology in more detail. In Section 3, the SweGrid system and its requirements are presented. Section 4 then describes how SLAs could be used in SweGrid. An SLA Management system is thereafter proposed in Section 5. Finally, we draw some conclusions and discuss future work in Section 6.

2 Service Level Agreements

A Service Level Agreement (SLA) is a contract between a user and a provider of a service specifying the conditions under which a service may be used. An SLA specifies the agreed-upon level of availability, serviceability, performance, and operation both in high-level business-value terms understood by end-users, and low-level technical terms that can be enforced to reserve resource capabilities. Typically, an SLA contains Quality-of-Service commitments (including penalties and rewards), pricing policies, authorization policies, and negotiation policies. To ensure the authenticity of an SLA it is digitally signed by all parties using a trusted-third-party (TTP)-based model such as X.509 certificate authorities, a.k.a. the Public Key Infrastructure (PKI) [17]. An SLA can be constructed either by the user or the provider of the service. In the former case it embodies a request for resources or capabilities, and in the latter case it represents an offer of available services and guarantees. Once an SLA has been mutually agreed upon and signed it has to be actively managed to ensure that all the commitments are attained.

The SLA is a mutual agreement among all stakeholders in a service interaction, and everyone must therefore also be able to subscribe to possible modifications triggered either by other stakeholders directly or by an automatic upgrade or downgrade by the agents representing them. Not all stakeholders may, however, have the same flexibility in changing various parts of the agreement. Typically, the service provider only allows a very limited set of parameters to be renegotiated and there might also be temporal restrictions, e.g. when and how frequently updates can be made. Monitoring systems must, hence, be tied directly to the agreement both for the users and the providers. The monitoring system must also report when violations arise due to either overuse or failure to provide the qualities committed to. From a provider's point of view it might also be interesting to

detect underutilization, in order to reallocate idle resources. In a fully automated management solution it is, furthermore, desirable to predict when violations are about to happen by analyzing usage history and patterns in order to take corrective actions, such as adjusting the number of resources in an active pool, before the agreement is broken. The management system is only aware of the conditions under which the agreement can be considered violated, and what possible penalties it may cause various parties, but it must be possible to specify and configure application specific actions to take when certain violation events occur. To determine whether the SLA is in fact violated it must be possible for the management infrastructure to validate it automatically and deterministically whenever the agreement is in effect.

Agreement offers that may be negotiated are referred to as SLA templates, and play an important role in the discovery of appropriate services. Hence, the templates give providers a way to communicate possibly dynamic properties such as load constraints to consumers. The templates can also be viewed as the typical contract that the provider is most likely to be able to fulfill based on assessment of previous contracts.

SLA Languages have been proposed in [18-20] and are now converging within the Grid community in the WS-Agreement specification in the Global Grid Forum. Some initial experiments on WS-Agreement templates have been performed in [21]. As yet, negotiation and self-adapting capabilities have not been explored in this context.

3 National Grid Requirements

SweGrid is a national Grid providing compute resources to scientific projects in Sweden. It comprises 600 commodity PCs interconnected with a 10Gb/s high performance network evenly distributed among six High Performance Computing (HPC) centers. A Grid meta- scheduler [22] is responsible for publishing resource information as well as selecting the appropriate cluster where a compute job could run. The matchmaking process must, for instance, make sure that the user has the required authorization rights to run the job, and that the correct OS and run time environments are installed on the target machine. Most of the jobs are long running and trivially parallel (with minimal inter-subjob communication). A typical job could, for example, perform some brute-force, number-crunching task on a series of machines in an iterative manner, improving the precision of the calculation the longer the job runs. It is therefore important that the allocated and requested execution (wall) time of the job is actively enforced, as many of these jobs never stop running by themselves. The local cluster scheduling system has the low-level control of jobs. What makes this environment a typical Grid network is that the various HPC centers involved are very heterogeneous in terms of local polices used, such as security, and job reservation and prioritization strategies employed.
To enable efficient and fair resource sharing in this Grid an accounting system built on top of standard Web and Grid services protocols and middleware was developed [11] to track resource usage, and to enforce Grid-wide resource quotas in real time. Figure 1 shows a high-level overview of the accounting-enabled Grid.

1. In order to submit jobs that are charged against a quota granted to a research project, an account for the project needs to be created. The account embodies the resource entitlement of a group of researchers and is decorated with project specific policies that are enforced before the job is submitted at a cluster. Deployment and activation of the account is typically done by an accounting service administrator.
2. All the resources shared on the Grid need to run a resource information service collecting resource capability and QoS parameters. The parameters may be both static and highly dynamic.
3. The information service, in turn, registers itself and its cluster in one or more global index services with a subset of the collected QoS in order to allow for more efficient resource discovery.

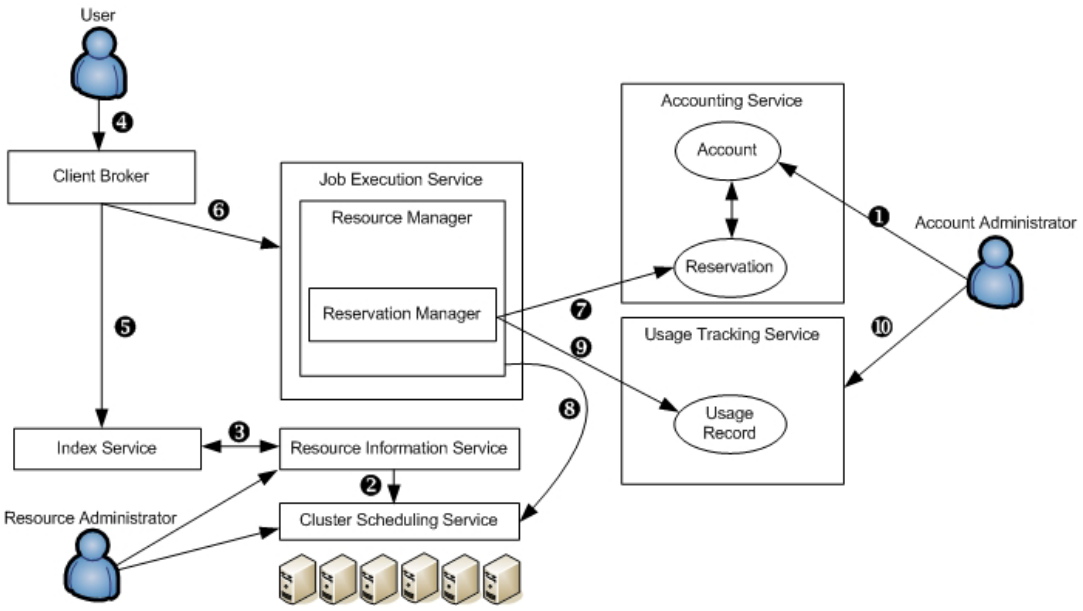

Figure 1: SweGrid Architecture

4. The user composes a resource request using a job specification language. The specification defines what capabilities are required to run the job, such as the required wall or CPU time, the number of cluster nodes needed, and the OS that the job executable can run on. It also contains job specific information including the name of the executable to run, and the required input parameters that need to be staged in to the target resource.
5. The job and resource specification is then passed into a client broker for submission. The first task of the client broker is to select a cluster that matches the user request using some matchmaking algorithm to distribute the load efficiently.
6. The second task of the broker is to submit the job to the cluster selected in the previous step and to perform all the required file transfers to stage in the input parameters to the local resource manager and scheduling system.
7. Before the job is actually put in a queue to run on one or more cluster nodes, the accounting system intercepts the call via a reservation manager component in the resource manager. The reservation manager looks at the resource specification that was initially composed by the client to determine what resources need to be reserved, for how long, and at what price. This decision is taken after evaluating local resource policies, which may differ between the clusters in the Grid depending upon, for instance, current load and cluster scheduling features available. A time-limited reservation is then created in the appropriate account, which was created in Step 1.
8. The job specification is now translated into the local scheduling system syntax and then submitted. The priority of the job is determined by the results of the reservation attempt in the previous step. If the quota of the account was exceeded, the resource, again depending on policy, may decide to downgrade the priority of the job or refuse to submit it altogether.
9. After the job has finished, the accounting system intercepts the job execution process once again in order to calculate the actual price of the job, log an accounting record, and charge the account. If the job finishes prematurely, a discount may, for example, be given on the price reserved in Step 7. The cost must, however, not be greater than was reserved by the user. How to treat and charge jobs that failed to execute is also subject to local resource policy. Regardless of what happened during the job submission, an accounting log is always recorded both locally at the resource as well as globally in a logging and usage tracking service.
10. To monitor how the accounting policies were enforced and to get a general idea of the performance of the system, the account administrator may query transaction information in the accounting service and more detailed usage information in the usage tracking service.

There are a number of limitations to and issues with the scenario just described. These issues are discussed next to give some background to the SLA re-factoring proposed in the next section.

- Account administrators currently have to add all account members (researchers who may consume quota granted to the account) manually and set up scripts to distribute a biannual allocation according to some policy, such as monthly staggered allocation chunks using a command line administration tool. This is error prone and exposes too much internal service detail to administrators. Further, there is no convenient way to see what allocation approach is in effect.
- The index service only exposes very limited QoS parameters for a cluster, typically only execution permissions. More detailed information may be queried by federating the query to resource information services, but the information gathered there is not usage based. Hence, it is, very hard to determine the ability of service providers (in this case the clusters) to meet their promised QoS parameters, and to support quality of compliance or reputation (trust) based service selection.
- The client broker algorithm is static and provides no flexible way of applying a policy to a group of job submissions to be performed as part of the same task. Handling of failed jobs, possible migrations, cancellation and monitoring of jobs must be done manually by the end-user. The result is that users either write their own scripts on top of the broker to their best ability without any form of algorithm or best practices reuse, or that they do the coordination by hand. In other words, the user-task goal fulfillment is completely manual and without any form of guarantees.
- The local resource policy is not directly linked to the user request or the service promised by the provider. Thus, negotiating policies is difficult, and as a result only very basic *demand for service* [7] negotiation is supported (the resource must accept all requests as is, or the job will fail). No automatic QoS monitoring or enforcement is carried out in relation to the job request. Therefore, violations cannot be detected and the parties involved cannot be notified to take countermeasures, such as migrating to another cluster if the job gets stuck in a queue due to some temporary node overload or downtime. Furthermore, there is no well-defined endpoint where these violation notifications can be sent. Because there is no infrastructure to detect violations, there is no support for predicting violations either.
- The resource-administrator policy for distributing resources fairly among clients while maintaining high utilization may be customized, but there is no middleware support for simplifying this task. There is no means to adapt the system automatically to changed circumstances or user requirements, or to automatically learn from past experience. The policies are all managed on a trial and error basis and by using off-line *a posteriori* analysis. It is not possible to correlate policy settings and resource infrastructure configuration with the ability to meet the QoS parameters promised to users.
- In general, the accounting-enabled Grid system described above is very flexible in terms of the ability to configure local stakeholder policies. However, this flexibility comes at the price of complexity, and it is very hard to map policies to higher-level goals. Examples include minimizing the overall completion time of a series of end-user job submissions while keeping the cost within a given budget; and maximizing utilization, throughput and resource sharing profits while meeting the offered QoS guarantees.
- The current use of implicit QoS contracts between the stakeholders makes it hard to offer discounts and enforce penalties to reflect the actual level of service delivered. The implementation of various economic market-simulation and pricing models is also complicated for the same reason. Further, there is no formal contract signing procedure making it hard for users or resource providers to enforce non-repudiation.

The issues listed above are summarized and mapped to the stakeholders, who are most negatively affected by the issue, in Table 1.

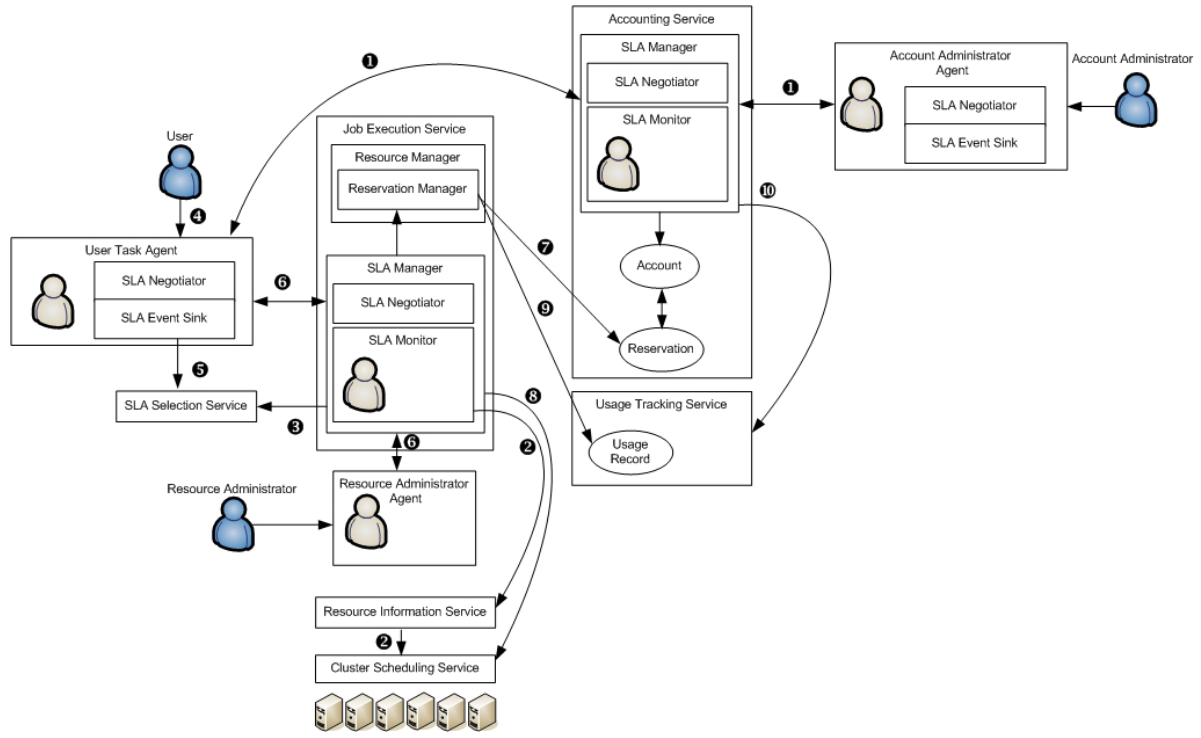

Figure 2: SLA-aware SweGrid Architecture

Table 1: Issues with current Grid Accounting System

Issue	Stakeholder(s)
Manual, error prone account management	Account Administrator
No usage based service selection	End-User
Manual, error prone multi-job coordination	End-User
No service level negotiation and enforcement leading to problems detecting service level violations	Resource Administrator, End-User
Complex policy configuration to optimize higher level goals	Resource Administrator, Account Administrator, End-User
Lack of adaptation and automatic reconfiguration support	Resource Administrator
No explicit, non-repudiation agreement support between stakeholders	Resource Administrator, Account Administrator, End-User

4 An SLA-Aware National Grid

We now propose an architecture aimed at eliminating the limitations and issues discussed in the previous section. The approach taken is to introduce an SLA-aware, middleware-driven architecture. Akin to the Grid accounting system presented in [11, 12] we want the new infrastructure to be as non-intrusive as possible and based on standard Web and Grid services protocols, because experience has shown that this is the best way to achieve both interoperability and early and smooth adoption. The internals of the SLA-aware components are presented more elaborately in the next section. Here we just show how these components can be applied to the Grid system presented in the previous section.
Figure 2 shows the SLA-enabled Grid architecture corresponding to the accounting-enabled Grid depicted in Figure 1.

1. We introduce an account administrator agent responsible for executing tasks on behalf of the account administrator and negotiating SLAs with the end-user representatives holding accounts in the accounting services administered. The contract negotiated by this agent can be seen as a fairly static overarching VO policy [23]. Contracts negotiated between end-users and resources in order to consume account quota must all comply with the terms in this contract. The agent can also set up new accounts and distribute allocations periodically according to a policy in the contract determined by the administrator. A sample SLA is shown in Figure 3. The contract must be signed by the account administrator credentials and may optionally be signed by the end-user representative. One-party signing is acceptable since the contract will only describe the obligations of the accounting service.
2. As in the SLA-unaware architecture the Resource Information service collects resource information from the local scheduling service. This information may for instance be in the form of the standard GLUE schema or JSIM resource models.
3. The SLA Manager component introduced within the Job Execution Service (JES) is now responsible for collecting the information pertaining to the QoS parameters in the contracts offered by the JES, and constructing SLA templates that are published in a discovery service together with contract compliance history and other reputation and trust building information.
4. Mirroring the administrator agent design, we introduce a user task agent responsible for executing job-submission-related tasks on behalf of end-users. The user describes a set of jobs and their resource requirements in a language such as JSDL and passes it into the agent together with some higher-level goals expressed in terms of policies applying to the task. For example, all the jobs must finish before a certain time and must not cost more than a budget limit. If a job fails to meet its SLA, then the agent will receive a notification and resubmit or migrate the job to another provider automatically. If the agent receives warnings in the form of violation predictions, it may try to renegotiate contracts to still meet the higher-level goals of the end-user.
5. The agent is also responsible for selecting an appropriate set of resource providers (clusters) by querying the discovery service, SLA Selection Service. A subset of the clusters may be selected with which individual call-for-proposal based bid and contract-negotiation interactions are started. For instance, the price may not be available in the SLA Selection Service, but may need to be dynamically negotiated just before submission time so the current load on the respective resource can be taken into account.
6. The end-user agent drives the negotiation interaction with the clusters selected in the previous step and signs agreements on behalf of its user. The resource administrator agent signs the same contract on behalf of the resource administrator. The resource administrator agent is responsible for simplifying and automating resource configuration and learning-capable adaptation based on higher-level resource administrator goals. The resource administrator agent controls the SLA manager component of JES and indirectly sets policies that determine the price of resources and the job prioritization policies. A sample account (VO) SLA is shown in Figure 4.
7. The price information is then used by the reservation manager as well as by the negotiation component to drive the appropriate quota reservation and enforcement interaction with the accounting service; and to give end-users service-level, price-quote offers in the SLA templates respectively.
8. The job is submitted to the local cluster scheduler with the negotiated priority and other job specific properties.
9. When the job has completed, the JES SLA manager must calculate the final charge of the job based on the service level delivered and possible SLA violations, obligations, and penalties. The outcome is then stored locally in the resource administrator agent's knowledge repository to adapt the existing configuration and policies to better meet the resource administrator goals. The accounting information is logged in the usage tracking service and the appropriate account is charged; the reservation committed and removed; the consumed quota withdrawn from the account; and finally a transaction record is logged in the accounting service.

10. The accounting service SLA manager can now access the usage information and the transaction record to update its policy and configuration to better meet the higher-level goals of the account administrator.

```
account_sla = { parties,
                monitoring,
                qos_parameters,
                guarantees,
                constraints }
parties = { account_administrator,
            project_leader,
            project_members }
project_members = Σ project_member
monitoring = { monitoring_services, report_specification }
monitoring_services = { account_service, logging_service }
report_specification = { report_interval, report_parameters }
report_parameters = { overdraft,
                      allocation,
                      distribution,
                      job_usage_record }
qos_parameters = {account_service.account_data,
                  logging_service.usage_record }
guarantees = {overdraft < 10,
              allocation = 360000,
              distribution_type = staggered,
              distribution_interval = monthly }
constraints = { start_time = 2005-01-01:10:00,
                end_time = 2005-07-01:10:00 }
```

Figure 3: Sample Account SLA

```
resource_sla = { parties,
                 monitoring,
                 qos_parameters,
                 guarantees,
                 constraints }
parties = { resource_administrator,
            project_member }
monitoring =  resource_information_service
qos_parameters = resource_information_service.jobs
guarantees = { priority = high,
               cpu_time = 36000,
               completion_time = 2005-02-01:10:00,
               cost = 36000,
               charge_account = snic-01-01 }
constraints = { start_time = 2005-01-01:10:00,
                end_time = 2005-02-02:10:00 }
```

Figure 4: Sample Resource SLA

5 SLA Manager Architecture

The SLA Manager component that we propose for making Web and Grid services SLA-aware in a non-intrusive way is described in some more detail in this section. The manager can be divided into six subcomponents 1) SLA Negotiator (SLAN), 2) SLA Monitor (SLAM), 3) SLA Event Sink (SLAS), 4) SLA Policy Manager (SLAP), 5) SLA Rating Engine (SLAR) and 6) SLA Task Manager (SLAT).

The subcomponents are governed by a couple of general-purpose services, the Decision Making System (DMS) for pluggable decision making engines such as AI goal-driven, rule-based engines, and the Component Bus (CB) for efficient asynchronous inter and intra component communication. Figure 5 depicts this architecture and the most important interactions. All of these components are designed with clearly defined interfaces and interactions to make it easy to replace a subcomponent with a custom implementation.

5.1 Decision Making System (DMS)

The DMS service exposes an abstract set of interfaces that can be used to communicate with arbitrary decision making software plugged into the SLA framework. Figure 6 gives some examples of various DMS implementations that might be integrated.

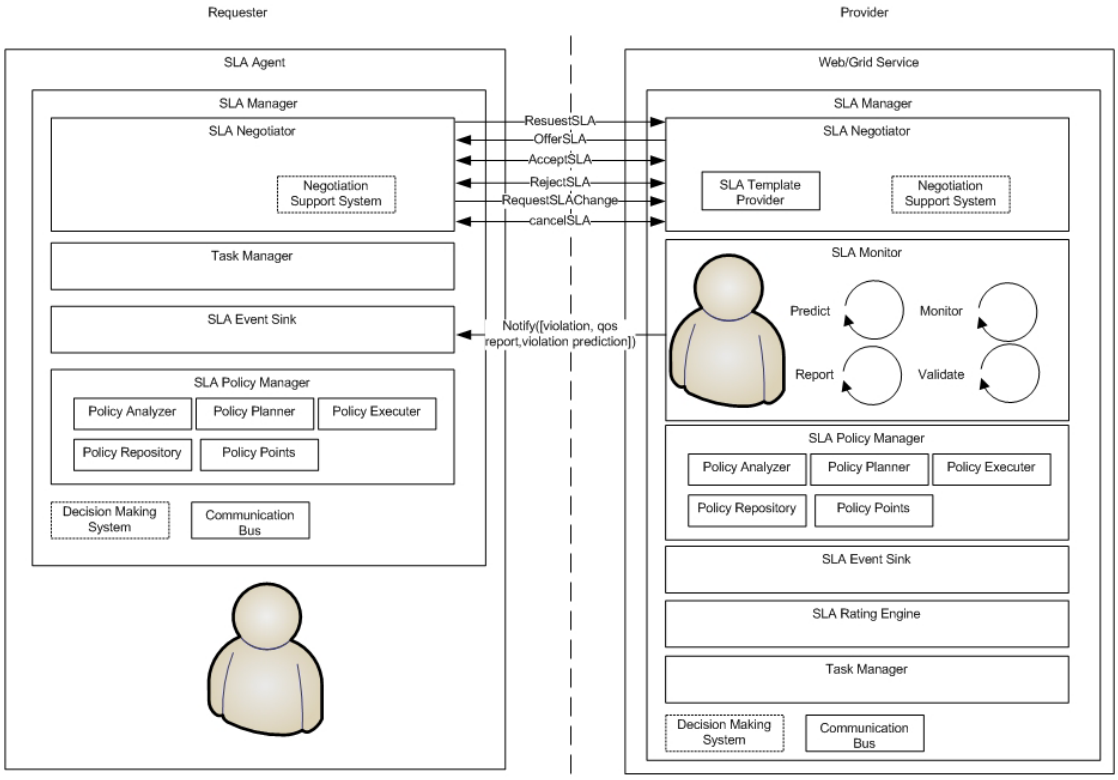

Figure 5: SLA Manager Architecture

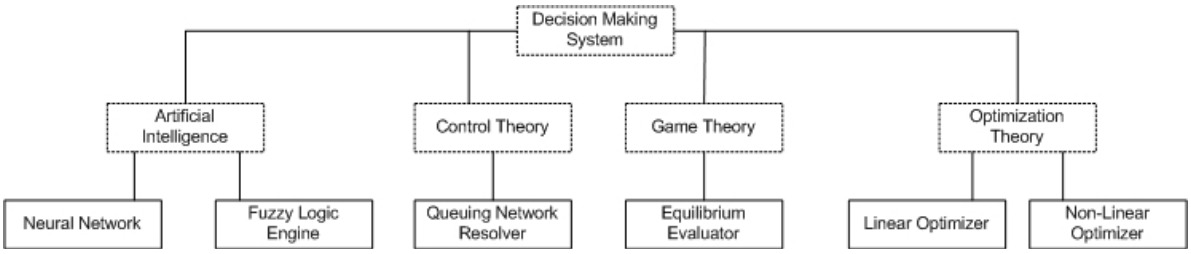

Figure 6: Decision Making Systems

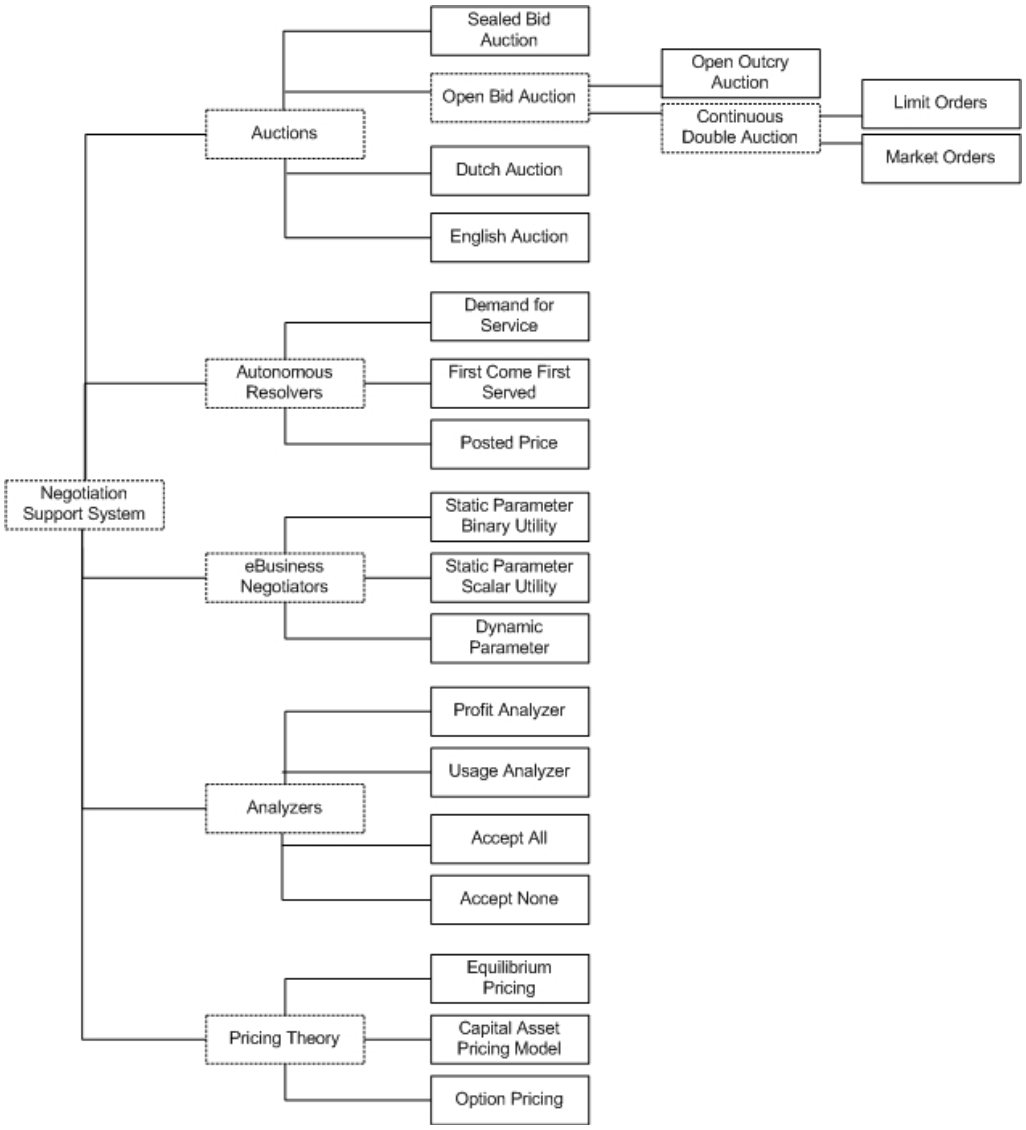

Figure 7: Negotiation Support Systems

5.2 Component Bus (CB)

The different SLA manager sub-components need a consistent and efficient way of communicating with each other asynchronously. Similarly, distributed SLA Managers need to be able to communicate with each other, for example, to negotiate contracts or share provider reputation knowledge or usage reports. The Component Bus hides the protocol specifics from the rest of the SLA Manager to make it easier to switch between different transport bindings and implement reliable messaging on behalf of the component. Protocol details such as WSDL, SOAP, WSRF, and WSDM specifics are also encapsulated in this service. The set of WSDL interfaces supported are, however, published to make it easy to write custom, remote components that communicate with this service.

5.3 SLA Negotiator (SLAN)

The negotiator subcomponent is responsible for constructing SLA templates that can be offered in bid publications. It must therefore collect enough information from other subcomponents in order to offer SLAs that are possible for the provider to attain. During a point-to-point negotiation phase more information can be collected about the other endpoint in the negotiation, and usage history may be taken into consideration. The interaction consists of 6 message primitives with payloads defined by the ontology in effect, which in turn is determined by the Negotiation Support System (NSS). The interaction is carried out between two SLAN components and the primitives are:

- **RequestSLA**. The service user issues a call for proposal to service providers
- **OfferSLA**. The service provider sends an offer to one or more service requesters
- **AcceptSLA**. Either the provider or the requester may confirm acceptance of the SLA. As a result the contract is signed by the party sending the message
- **RejectSLA**. Either the provider or the requester may reject the SLA offered or requested
- **RequestSLAChange**. The requester can try to change a previously negotiated SLA, for example, to request fewer resources to get discounts. Provider and NSS policies may determine whether the requester is allowed to renegotiate after the SLA has been signed and other temporal restriction. One policy might be that SLAs cannot be renegotiated but must be cancelled, and then a new SLA has to be negotiated from scratch. Our model does not support provider-initiated renegotiation since this is modeled through penalties in the SLAs.
- **CancelSLA**. This message could be used both by the requester and the provider to cancel a request or an offer. This is typically done before the contract has been signed but some policies and negotiation algorithms may allow SLAs to be canceled at any time to avoid wasting time on enforcing SLAs that one or more parties have lost interest in.

NSS is designed akin to the DMS described in Section 5.1. Figure 7 shows anticipated Negotiation Support System implementations.

5.4 SLA Monitor (SLAM)

The SLA Monitor subcomponent is responsible for automatically measuring, monitoring, validating, and triggering reports and violation notifications according to the SLA specification. The only input to this component is the signed SLA, which thus must be self-contained and possible to validate automatically to detect violations. The SLA Monitor may also provide some prediction functionality. Forecasts on possible future violation risks may be compiled based on usage history and sent out to the SLA parties to take countermeasures. The predictions may also be used internally in the SLA Manager to modify configuration and policies to avoid violations. Although this component in reality may interact with legacy information-provisioning, back-end services, it must be transparent to the implementation in order to make the component easy to deploy in any infrastructure. Wrappers may for instance be written, whose endpoints are made available in the SLA.

5.5 SLA Event Sink (SLAS)

Outbound messages are put on the CB. Incoming messages on the bus are, however, forwarded to the SLA Event Sink where the filtering occurs to find the correct sub component configuration specifying how the event should be treated. Automated actions may, for example, be specified to trigger a chain

of events or to take counter measures automatically in case of violations. The SLAS may internally be implemented as a JMS or MQSeries enabled event channel offering component specific QoS.

5.6 SLA Policy Manager (SLAP)

At the heart of autonomous computing is the MAPE control loop (Measurement, Analysis, Planning, Execution) and the Policy-based management architecture with policy decision and enforcement points. The SLA Policy Manager subcomponent marries these two architectures and maintains a policy repository with the most recent policies, which are continuously updated as a result of further usage and SLA compliance history analysis.

5.7 SLA Rating Engine (SLAR)

The rating engine is responsible for setting the price on resources and services provided, and for calculating discounts and rates applicable. It should be able to give both price quotes as well as to calculate the actual price of a delivered service to be charged in a billing or accounting system. The inputs to the engine are the price plan, the usage record, and the SLA.

5.8 SLA Task Manager (SLAT)

The final subcomponent is the SLA Task Manager, which embodies the high level goal that is to be achieved, as well as a schedule for how to achieve it, defined in some workflow engine interpretable language. Clients may group a set of jobs to be executed in a task, with well-defined criteria and actions to handle faults and perform automatic migration. The account manager discussed in Section 3 may set up a reoccurring, cron-job like, task to distribute granted allocations to research projects on a monthly staggered basis. The SLAP, discussed in Section 5.6, may use the task manager to execute its plans to maintain and attain service levels committed in the SLA being managed.

6 Conclusions and Future Work

We now refer back to the requirements summarized in Table 1 to highlight how they were addressed in our SLA solution.

An account administrator agent was introduced to simplify and automate management of accounts. This involves monitoring usage to adjusted policies automatically as well as setting up repeating tasks such as account allocations. Services can be selected based on usage by means of the SLA selection service exposing SLA templates that may be negotiated. Both the construction of the templates as well as the point-to-point negotiation may involve consulting the usage records. If users misconduct and submit more jobs than they were granted on the resources all the stakeholder agents, the user, the resource, and the account administrator, automatically detect the violation and can react upon it. The resource may stop jobs from being submitted or submit them into a low priority queue, alternatively change the price of the resource for that user in future negotiations. The account administrator may stop the particular user from using up shared account quota for a limited time. Finally, the user will also get a notification and can then choose to either pay more for the same service to maintain the job flow, or decrease the submission flow. By the introduction of decision making engines capable of inference reasoning over rule-bases, high-level policy rules may be entered into the agent knowledge bases to optimize certain QoS criteria in response to system events and to set more fine-grained policies and configurations automatically. The most important benefit from the SLA framework is, however, the formal signing of agreements, which may be used as a more controlled way of using resources at certain guaranteed service levels at a certain price. The agreements allow all stakeholders to merge their preferences to achieve the best overall system performance, utilization, and usability.

Future work includes implementing the contracts using WS-Agreement and the management infrastructure on top of WSRF and WSDM. For automatic metrics measurements we also intend to utilize WSLA. Fuzzy Logic rule engines and OWL-based inference engines and usage-based analyzers will provide the first candidates for the initial prototype of the Decision Making System and Negotiation Support System incarnations. Simulation test-beds will be developed allowing statistical

measurements to be performed in a controlled environment to fine-tune the framework and the policies before deploying it in the SweGrid production Grid.

Acknowledgments

I would like to thank my colleagues Lars Malinowsky, Peter Gardfjäll, and Olle Mulmo for many discussions related to the topics discussed in this paper. I would also like to thank Sandra Brunsberg, and my supervisor Lennart Johnsson for reviewing the paper.

References

[1] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss, "Overview of the I-WAY: Wide Area Visual Supercomputing," *International Journal of Supercomputer Applications*, vol. 10, pp. 123-130, 1996.

[2] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, 1999.

[3] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, 2 ed: Morgan Kaufmann, 2003.

[4] F. Berman, G. Fox, and A. J. G. Hey, *Grid Computing: Making The Global Infrastructure a Reality*: John Wiley & Sons, 2003.

[5] J. Joseph, M. Ernest, and C. Fellenstein, "Evolution of grid computing architecture and grid adoption models," *IBM Systems Journal*, vol. 43, pp. 624-645, 2004.

[6] J. Carolan, S. Radezsky, P. Strong, and E. Turner, *Building N1 Grid Solutions Preparing, Architecting, and Implementing Service-Centric Data Centers*: Sun BluePrints, 2004.

[7] J. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, pp. 41-50, 2003.

[8] S. Blake, D. Black, M. Carlson, E. Davis, W. Zheng, and W. Weiss, *RFC 2475: An Architecture for Differentiated Services*: IETF, 1998.

[9] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, *RFC 2205: ReSerVation Protocol (RSVP) version 1 functional specification*: IETF, 1997.

[10] K. Nahrstedt, H. Chu, and S. Narayan, *QoS-aware resource management for distributed multimedia applications*: IOS Press, 1998.

[11] T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O. Mulmo, "An OGSA-Based Accounting System for Allocation Enforcement across HPC Centers," presented at 2nd International Conference on Service Oriented Computing, New York, NY, 2004.

[12] E. Elmroth, P. Gardfjäll, O. Mulmo, and T. Sandholm, "An OGSA-based Bank Service for Grid Accounting Systems," *Applied Parallel Computing. State-of-the-art in Scientific Computing. Lecture Notes in Computer Science. (to appear) Springer Verlag.*, 2004.

[13] A. Barmouta and R. Buyya, "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration," presented at International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France, 2003.

[14] S. Jackson, "QBank: A Resource Management Package for Parallel Computers," Pacific Northwest National Laboratory, Washington, USA 2000.

[15] J. Hendler, T. Berners-Lee, and E. Miller, "Integrating Applications on the Semantic Web," *Journal of the Institute of Electrical Engineers of Japan*, vol. 122, pp. 676-680, 2002.

[16] D. De Roure, N. R. Jennings, and N. R. Shadbolt, "The Semantic Grid: A future e-Science Infrastructure," in *Grid Computing - Making the Global Infrastructure a Reality*: John Wiley and Sons Ltd., 2003, pp. 437-470.

[17] R. Housley, W. Ford, W. Polk, and D. Solo, *RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile*: IETF, 1999.
[18] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Cordinating Resource Management in Distributed Systems," *Lecture Notes in Computer Science*, vol. 2537, pp. 153-183, 2002.
[19] D. D. Lamanna, J. Skene, and W. Emmerich, "SLAng: A Language for Defining Service Level Agreements," presented at The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03), 2003.
[20] A. Dan, E. Davis, R. Kearney, A. Keller, R. P. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and Y. A., "Web services on demand: WSLA-driven automated management," *IBM Systems Journal*, vol. 43, 2004.
[21] H. Ludwig, A. Dan, and B. Kearney, "Cremona: An architecture and Library for Creation and Monitoring of WS-Agreements," presented at 2nd International Conference on Service Oriented Computing (ICSOC 2004), New York, 2004.
[22] O. Smirnova, P. Eerola, T. Ekelöf, M. Ellert, J. R. Hansen, A. Konstantinov, B. Kónya, J. L. Nielsen, F. Ould-Saad, and A. Wäänänen, "The NorduGrid Architecture and Middleware for Scientific Applications," *Lecture Notes in Computer Science*, vol. 2657, pp. 264-273, 2003.
[23] G. Wasson and M. Humphrey, "Toward Explicit Policy Management for Virtual Orginizations," presented at 4th International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, 2003.

The Design, Implementation, and Evaluation of a Market-Based Resource Allocation System

Kevin Lai*
kevin.lai@hp.com

Thomas Sandholm†
sandholm@pdc.kth.se

ABSTRACT

This paper examines the use of market-based techniques to allocate the virtualized resources of distributed hosts. We present Tycoon, a system in which users bid for virtualized resources and receive in proportion to their bid. Tycoon differs from other work in economic resource allocation by using a distributed price-anticipating model for bidding and allocation. This allows the system to be scalable, fault-tolerant, and agile, while making economically efficient allocations.

We present the design of a complete system using market-based techniques, including characterization of resources, secure banking protocols, and distributed bidding algorithms. Using an implementation of this design, we measure the economic efficiency and fairness of various allocation algorithms. We find that traditional proportional share allocation achieves high fairness, but poor efficiency, while the social optimum achieves optimal efficiency, but poor fairness. The Tycoon system, despite being highly decentralized, achieves both high efficiency and high fairness.

1. INTRODUCTION

The primary advantage of distributed shared computing platforms like the Grid [17] and PlanetLab [31] is their ability to pool together shared computational resources. This allows increased throughput because of statistical multiplexing and the bursty utilization pattern of typical users. Sharing nodes that are dispersed in the network allows lower delay because applications can store data close to users. Finally, sharing allows greater reliability because of redundancy in hosts and network connections.

However, resource allocation in these systems remains a challenge. More specifically, the problem is to allocate resources to *strategic* users: those who act purely in their own interest and compete with other users for resources. Although not all users are completely strategic, most are, and will compete for computational resources just as they compete for money, recognition, and every other desirable thing. One system with only eleven participant organizations exhibited strategic behavior after fewer than four months [8].

Several non-economic allocation algorithms have been proposed [40, 4, 45], but these typically assume that task values (i.e., their importance) are the same, or are inversely proportional to the resources required, or are set by an omniscient administrator. However, in many cases, task values vary significantly [8], are not correlated to resource requirements, and are difficult and time-consuming for an administrator to set, especially in large systems. Instead, economic resource allocation systems [3, 12, 10, 16, 33, 43, 44] rely on users to set the values of their own jobs and provide a mechanism to encourage users to truthfully reveal those values. This allows the allocation mechanism to make allocations that are *economically efficient*, where resources go to the users who value them the most.

In practice, previous economic resource allocation systems have had disadvantages: high overhead, high latency, and/or centralization. The overhead of previous systems has consisted of frequent interactive bidding, or, conversely, infrequent bidding that increases the latency to acquire resources. Most users would prefer to run their program as they would without a market-based system and forget about it until it is done. The latency to acquire resources is important for applications like a web server that need to allocate resources quickly in reaction to unexpected events (e.g., breaking news stories from CNN). Another disadvantage is that many market-based systems rely on a centralized market that limits reliability and scalability.

To overcome these disadvantages, we examine the Tycoon decentralized *price-anticipating* [20] allocation model in which a user bids for a resource and receives the ratio of his bid to the sum of bids for that resource. This proportional scheme is simpler, more scalable, and more responsive [23] than auction-based schemes [16, 33, 43, 3, 12]. Previous work has analyzed price-anticipating schemes in the context of allocating network capacity for flows for users with unlimited budgets [20, 19] or strictly in simulation [15]. In this work, we describe the design, implementation, and evaluation of an operational Tycoon system of 40 hosts.

More specifically, our contributions are as follows:

• **We describe a secure banking protocol.** A secure banking protocol is necessary for any economic system. We describe a simple banking protocol based on *receipts*. The basic operation is for one user to transfer credits from his account to another user's account and receiving a receipt from the bank as proof of the operation. The user then presents this receipt to the credit recipient. This system avoids the double-spending problems of micro-payment schemes and the race conditions of credit-checking systems.

• **We describe a scalable distributed bidding protocol.** Distributed markets allow increased reliability and scalability without requiring centralized trust. However, they introduce the question of how users should bid in multiple markets simultaneously. Previous work [15] described the fixed budget best-response algorithm for distributed bid-

*Information Dynamics Laboratory, HP Labs, Palo Alto, CA 94304
†KTH – Royal Institute of Technology, SE-100 44 Stockholm, Sweden

ding. Here we describe a protocol for this algorithm.

• **By extending the price-anticipating model to allocate a continuous flow of resources, we show that this model can be low overhead and handle multiple types of resources.** Previous work on price-anticipating mechanisms [20, 10, 19, 15] only considered one-shot, instantaneous allocation of resources. The allows tractable models, but real applications require a continuous stream of resources over time. Here we model resources as a *flow* instead of a volume. A *continuous* bid consists of a rate of spending for a rate of resources (e.g., $1 for 1 GHz). This provides the added benefit of reducing the need to do fine grained bidding because bids can remain in effect for as long as a user desires. In addition, by only charging for usage instead of for allocation, we reduce the overhead for users of adjusting their bids in response to changing usage of resources by their applications. This significantly reduces the bidding overhead for applications that have unpredictable and quickly changing loads (e.g., any kind of server). Finally, we show that continuous bids can handle spatial resource like memory and disk space by characterizing their capacity and usage through time as well as space.

• **We show that this mechanism can be efficiently implemented as a thin layer on top of an existing virtualization platform.** The recent proliferation of virtualization platforms (e.g., VMWare [1], Xen [13], Microsoft Virtual Server, and VServer [2]) has shown that there are many different useful operating points in the tradeoff of degree of virtualization and overhead. In addition, compatibility and licensing issues may dictate one platform over another. Given this, it is important for a resource allocation system to be portable across different virtualization platforms while minimizing overhead. Previous work [24] has shown that a simpler price-anticipating mechanism can be implemented with low overhead on VServers. Here we show that the more sophisticated multi-resource Tycoon allocator can be implemented on top of Xen with less than 1% overhead.

• **We present experimental results showing that the price-anticipating resource allocation mechanism achieves both high efficiency and fairness.** A proportional share algorithm is shown to consistently have a low social welfare in our experiments when varying task importance, number of user, number of hosts and user host preferences. Further, a socially optimal algorithm is found to have a low level of fairness in tests where users compete for the same CPU cycles on a host. The efficiency of the Tycoon allocation algorithm is shown to scale from the proportional share values, when there is no difference in task importance between users over time, to the social optimum, when there is a very high variation of task importance. Finally, in the fairness experiments, the Tycoon allocations are showing better results than the social optimum.

As a result, we believe that shared distributed systems based on the fixed budget game can be highly decentralized, yet achieve a high degree of efficiency and fairness.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the system architecture. We describe the banking protocol in Section 3. In Section 4, we describe the design and implementation of the auctioneer, allocation algorithm, and bidding protocol. Section 4.5 contains details of how client agents manage distributed bids using the fixed budget best response algorithm. In Section 5, we describe our experiment setup and benchmark results. We describe related work in Section 6. We conclude by discussing some limit of our model and future work in Section 7.

2. ARCHITECTURE OVERVIEW

Tycoon is split into the following components: bank, auctioneer, client agent, and service location service (SLS) (each shown in Figure 1). The bank maintains records on each user's accounts. Each auctioneer manages the resources of a single host on behalf of the host's owner. A client agent manages its owner's bids at one or more auctioneers.

Auctioneers use the service location service to advertise resources, and agents use it to locate resources (as shown in steps 1 and 2 in Figure 1). The prototype uses a simple centralized soft-state server, but the other components would work just as well with more sophisticated and scalable service location systems (e.g., Ganglia [25] and SWORD [28]). Auctioneers register their status with the SLS every 30 seconds and the SLS de-registers any auctioneer that has not contacted it within the past 120 seconds. This status consists of the total amount spent on the host for each resource and the total amount of each resource type available (e.g., CPU speed, memory size, disk space), etc. The status is cryptographically signed by the auctioneer and includes the auctioneer's public key. Clients store this key and use it to authenticate the results of later queries and also to authenticate direct communications with the auctioneer.

The soft-state design allows the SLS to be robust against many forms of hardware and software failures. The querying agents may receive stale information from the SLS, but they will receive updated information if they choose to contact an auctioneer directly.

3. BANK AND BANKING PROTOCOL

The bank maintains account balances for all users and providers. Its main task is to transfer funds from a client's account to a provider's account (shown in step 3 in Figure 1). The source of the transfer receives a receipt that the transfer recipient can use to verify that the transfer occurred.

We assume that the bank has the public keys of all the users, and they all have the bank's public key. A further assumption is of roughly synchronized clocks. Alice and Bob are a fictional example sender and receiver. Alice begins by sending a message to the bank as follows:

$$Alice, Bob, amount, time,$$

$$Sign_{Alice}(Alice, Bob, amount, time)$$

$Sign_{Alice}$ is the DSA signature function using Alice's private key. The bank verifies that the signature is correct, which implies that the message is from Alice, that the funds are for Bob, and that the amount and time are as specified. The bank keeps a list of recent messages and verifies that this message is new, thus guarding against replay attacks. Assuming this is all correct and the funds are available, the

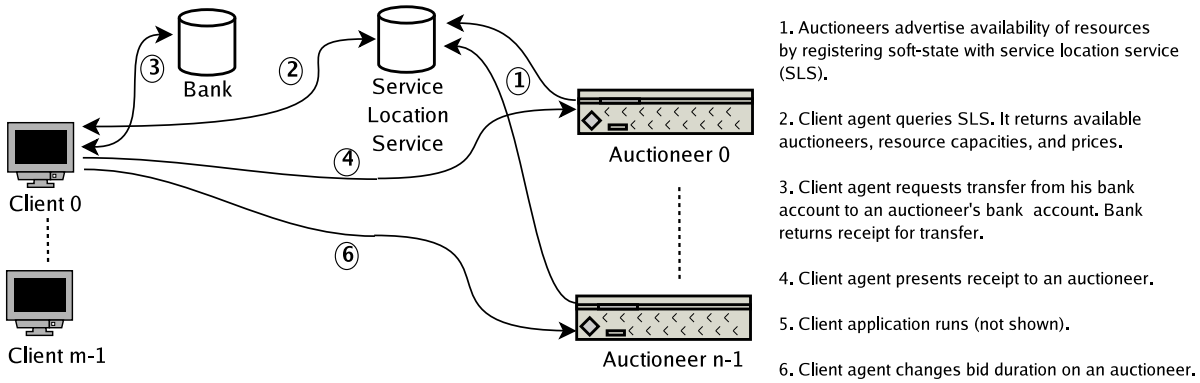

Figure 1: This figure gives an overview of how the Tycoon components interact.

bank transfers *amount* from Alice to Bob and responds with the following message (the *receipt*):

$$Alice, Bob, amount, time,$$

$$Sign_{Bank}(Alice, Bob, amount, time)$$

The bank sends the same time as in the first message. Alice verifies that the amount, time, and recipient are the same as the original message and that the signature is correct. Assuming the verification is successful, Alice forwards this message to Bob as described in § 4. Bob keeps a list of recent receipts and verifies that this receipt is new, thus guarding against replay attacks.

The advantages of this scheme are simplicity, efficiency, and elimination of double spending. Micro-currency systems are generally complex, have high overhead, and only discourage counterfeiting. The disadvantages of this approach are scalability and vulnerability to compromise of the bank. However, bank operations are relatively infrequent (see § 4.2 for how bids can be changed without involving the bank), so scalability is not a critical issue for moderate numbers of users and hosts. The vulnerability to compromise of the bank could be a problem. Possible solutions are discussed in § 7.

3.1 Monetary Policy

Monetary policy consists of deciding how to manage the flow of monies in the system. Part of this is determining how users obtain funds. Two possibilities are *open loop* and *closed loop* funding policies. In an open loop funding policy, users are funded at some regular rate e.g., from a national allocations committe as exemplified in [36]. The system administrators set their income rate based on exogenously determined priorities. Providers accumulate funds and return them to the system administrators. In a closed loop (or *peer-to-peer*) funding policy, users themselves bring resources to the system when they join. They receive an initial allotment of funds, but they do not receive funding grants after joining. Instead, they must earn funds by enticing other users to pay for their resources. A closed loop funding policy is preferable because it encourages service providers to provide desirable resources and therefore should result in higher economic efficiency.

In our initial deployment, we have used an open loop policy to bootstrap the system, but are gradually transitioning to a closed loop system as we add resource providers.

Another part of monetary policy is taxation. This is necessary to prevent users from hoarding currency. Hoarding can allow infrequent users to cause unpredictable (and in small systems, large) spikes in demand. Also, in closed loop systems, hoarders can cause an economic depression as they take currency out of circulation. Regardless of the consequences, the solution to hoarders is taxation. Other systems [8] have taken a similar approach.

4. AUCTIONEER

Auctioneers serve four main purposes: management of local resources, collection of bids from users, allocation of resources to users according to their bids, and advertisment of the availability of local resources.

4.1 Virtualization

To manage resources, an auctioneer relies on a virtualization system and a local allocation system. The implementation uses Xen 2.0 for virtualization. Xen provides each user with a separate file system and gives the appearance that he is the sole user of a machine, even if the physical hardware is being shared. The user accesses this virtual machine by using `ssh`.

For local allocation, Tycoon uses the Xen Borrowed Virtual Time [14] scheduler, which implements the standard proportional share scheduling abstraction [39].

4.2 Setting Bids

The second purpose of auctioneers is to collect bids from users. Auctioneers store bids as two parts for each user: the local account balance, and the bidding interval. The local balance is the amount of money the user has remaining locally. The bidding interval specifies the number of seconds over which to spend the local balance. Users have two methods of changing this information: `fund` and `set_interval`. `fund` transfers money from the user's bank account to the auctioneer's bank account, and conveys that fact to the auctioneer. It has the disadvantage that it requires significant latency (100 ms) and it requires communication with the bank, which may be offline or overloaded. `set_interval`

sets the bidding interval at the auctioneer without changing the local balance. It only requires direct communication between the client and the auctioneer, so it provides a low latency method of adjusting the bid until the local balance is exhausted.

Alice and Bob already have each other's public keys and Alice has the value $nonce_{Alice}$. A nonce is a unique token which Bob has never seen from Alice before. In the current implementation it is an increasing counter. First, Alice gets a bank receipt as described above. She then sends the following message to Bob:

$$Alice, Bob, nonce_{Alice}, interval, receipt,$$

$$Sign_{Alice}(Alice, Bob, nonce_{Alice}, interval, receipt)$$

The nonce allows Bob to detect replay attacks. Bob verifies that he is the recipient of this message, that the nonce has not been used before, that the receipt specifies that Alice has transferred money into his account, that the bank has correctly signed the receipt, and that Alice has correctly signed this message. Assuming this is all correct, Bob increases Alice's local balance by the amount specified in the receipt and sets Alice's bidding interval to *interval*. `set_interval` is identical, except that it does not include the bank receipt.

The key advantage of separating `fund` and `set_interval` is that it reduces the frequency of bank operations. Users only have to fund their hosts when they wish to change the set of hosts they are running on or when they receive income. For most users and applications, this is on the order of days, not seconds. Between fundings, users can modify their bids by changing the bidding interval, as described in the next section.

4.3 Allocating Resources

The third and most important purpose of auctioneers is to use virtualization and the users' bids to allocate resources among the users and account for usage. Although the current implementation only allocates CPU cycles because of virtualization limitations, the following applies to both rate-based (e.g., CPU cycles and network bandwidth) and space-based (e.g., physical memory and disk space) resources. A proportional share-based function is described here, but there are other allocation functions with desirable properties (e.g., Generalized Vickrey Auctions, described below).

For each user i, the auctioneer knows the local balance b_i and the bidding interval t_i. The auctioneer calculates the bid as b_i/t_i. Consider a resource with total size R (e.g., the number of cycles per second of the CPU or the total disk space) over some period P. The allocation function for r_i, the amount of resource allocated to user i over P, is

$$r_i = \frac{\frac{b_i}{t_i}}{\sum_{j=0}^{n-1} \frac{b_j}{t_j}} R. \quad (1)$$

Let q_i be the amount of the resource that i actually consumes during P, then the amount that i pays per second is

$$s_i = min\left(\frac{q_i}{r_i}, 1\right) \frac{b_i}{t_i}. \quad (2)$$

This allows users who do not use their full allocation to pay less than their bid, but in no case will a user pay more than his bid.

There are a variety of implementation details. First, the auctioneer gets the number of cycles used by each user from the kernel to determine if $q_i < r_i$. Second, the implementation sets $P = 10s$, so the auctioneer charges users and recomputes their bids every 10 seconds. This value is a compromise between the overhead of running the auctioneer and the latency in changing the auctioneer's allocation. With tighter integration with the kernel and the virtualization system, P could be as small as the scheduling interval ($10ms$ on most systems). Third, users whose bids are too small relative to the other users are logged off the system. Users who bid for less than .1% of the resource would run infrequently while still consuming overhead for context-switching, accounting, etc., so the auctioneer logs them off, starting with the smallest bidder.

The advantages of this allocation function (1) are that it is simple, it can be computed in $O(n)$ time, where n is the number of bidders, it is fair, and it can be optimized across multiple auctioneers by an agent (described in § 4.5). It is fair in the sense that all users who use their entire allocation pay the same per unit of the resource.

The disadvantage is that it is not *strategyproof*. In the simple case of one user running on a host, that user's best (or *dominant*) strategy is to make the smallest possible bid, which would still provide the entire host's resources. If there are multiple users, then the user's dominant strategy is to bid his valuation. Since, the user's dominant strategy depends on the actions of others, this mechanism is not strategyproof. One possible strategyproof mechanism is a Generalized Vickrey Auction (GVA) [42]. However, this requires $O(n^2)$ time, it is not fair in the sense described above, and it is not clear how to optimize bidding across multiple GVA auctioneers.

4.4 Advertising Availability

The auctioneer must advertise the availability of local resources so that user agents can decide whether to place bids. For each resource available on the local host, the auctioneer advertises the total amount available, and the total amount spent at the last allocation. In other words, the auctioneer reports

$$\sum_{j=0}^{n-1} s_i. \quad (3)$$

This may be less than the sum of the bids because some tasks did not use their entire allocation. This is reported instead of the sum of the bids because it allows the agent to more accurately predict the cost of resources (as required the algorithm described in § 4.6). Note that this information allows agents to make appropriate bids without revealing the exact amounts of other users' individual bids. Revealing that information would allow users to know each other's valuations, which would allow gaming the auctions.

4.5 Agent

The role of a tycoon agent is to interpret a user's preferences, examine the state of the system, make bids appropriately, and verify that the resources were provided. The agent is involved in steps 2, 3, 4, and 6 of Figure 1. Given the diversity of possible preferences, Tycoon separates agents from the infrastructure to allow agents to evolve independently. This is a similar approach to the end-to-end principle used in the design of the Internet [7, 11, 35], where application-specific functionality is contained in the end-points instead of in the infrastructure. This allows the infrastructure to be efficient, while supporting a wide variety of applications.

There are a wide variety of preferences that a user can specify to his agent. Tycoon provides for both high-level preferences that an agent interprets and low-level preferences that users must specify in detail. Examples of high level preferences are wanting to maximize the expected number of CPU cycles or to seek machines with a minimum amount of memory, or some combination of those preferences. Tycoon allows uncertainty in the exact amount of resource received because other applications on the same host may not use their allocation and/or other users may change their bids.

4.6 Best Response Algorithm

In a system with many machines, it is very difficult for users to bid on individual machines to maximize their utilization of the system. Tycoon allows the user to only specify the total bids, or the budget, he is willing to spend and let the agent compute the bids on the machines to maximize the user's utility. In order to compute the optimum bids, the agent must first know the user's utility as a function of the fraction of the machines assigned to the user. Since it is difficult, if not impossible, to figure out the exact formulation of the utility function, this model assumes a linear utility function for each user. That is, each user specifies a non-negative weight for each machine to express his preference of the machine. Such a weight is chosen by the user and determined mainly by two factors: the system configuration and the user's need. They may vary from user to user. For example, one user may have higher weight on machine A because it has more memory, and another user may have higher weight on B because it has a faster CPU. The weights are kept private to the users.

Now, suppose that there are n machines, and a user has weight w_i on machine i for $1 \le i \le n$. If the user gets fraction r_i from machine i, then his utility is

$$U = \sum_{i=1}^{n} w_i r_i \,. \tag{4}$$

The agent's goal is to maximize the user's utility under a given budget, say X, and the others' aggregated bids on the machines. Suppose that y_i is the total bid by other users on machine i. The user's share on i is then $\frac{x_i}{x_i+y_i}$ if he bids x_i on machine i. Therefore, the agent needs to solve the following optimization problem:

$$\text{maximize} \sum_{i=1}^{n} w_i \frac{x_i}{x_i + y_i} \,, \quad \text{s.t.} \tag{5}$$

$$x_i \ge 0 \,, \quad \text{for } 1 \le i \le n, \text{ and} \tag{6}$$

$$\sum_{i=1}^{n} x_i = X \,. \tag{7}$$

This optimization problem can be solved by using the following algorithm.

1. sort $\frac{w_i}{y_i}$ in decreasing order, and suppose that
$$\frac{w_1}{y_1} \ge \frac{w_2}{y_2} \ge \cdots \ge \frac{w_n}{y_n} \,. \tag{8}$$
2. compute the largest k such that
$$\frac{\sqrt{w_k y_k}}{\sum_{j=1}^{k} \sqrt{w_j y_j}} (X + \sum_{j=1}^{k} y_j) - y_k \ge 0 \,. \tag{9}$$
3. set $x_i = 0$ for $i > k$, and for $1 \le i \le k$, set
$$x_i = \frac{\sqrt{w_i y_i}}{\sum_{j=1}^{k} \sqrt{w_j y_j}} (X + \sum_{j=1}^{k} y_j) - y_i \,. \tag{10}$$

The above algorithm takes $O(n \log n)$ time as sorting is the most expensive step. It is derived by using Lagrangian multiplier method. Intuitively, the optimum is achieved by the bids where the bid on each machine has the same marginal value. The challenge is to select the machines to bid on. Roughly speaking, one should prefer to bid on a machine if it has high weight on the machine and if other's bids on that machine is low. That is the intuition behind the first sorting step.

One problem with the above algorithm is that it spends the entire budget. In the situation when there are already heavy bids on the machines, it might be wise to save the money for later use. To deal with the problem, a variation is to also prescribe a threshold λ to the agent and require that the margin on each machine is not lower than λ, in addition to the budget constraint. Such problem can be solved by an easy adaptation of the algorithm.

4.7 Predictability

Instead of maximizing its expected value, some applications may prefer to maintain a minimum amount of a resource. An example of this is memory, where an application will swap pages to disk if it has less physical memory than some minimum, but few applications benefit significantly from having more than that. Tycoon allows agents to express this preference by putting larger bids on fewer machines. Let R be the total resource size on a host and B be the sum of the users' bids for the resource, excluding user i. From (1), the user i's agent can compute that to get r_i of a resource, it should bid

$$b_i = \frac{r_i B}{R - r_i}. \tag{11}$$

However, this only provides an expected amount of r_i. To provide higher assurances of having this amount, the agent bids more than b_i. To determine how much more, the agent maintains a history of the bids at that host to determine the likelihood that a particular bid will result in obtaining the required amount of a resource. Assuming that the application only uses r_i of the resource, the user will pay more per

unit of the resource than if his agent had just bid b_i (see § 4.3), but that is the price of having more predictability.

4.8 Scalability

Since the computational overhead of the agent is low, the main scalability concern is communications overhead. When making bids, a user agent may have to contact a large number of auctioneers, possibly resulting in a large queueing delay. For example, to use 100 hosts, the agent must send 100 messages. Although the delay to do this is proportional to the amount of resources the user is using, for very large numbers of hosts and a slow and/or poorly connected agent host, the delay may be excessive. In this case, the agent can use an application-layer multicast service (e.g., Bullet [22]) to reduce the delay. Since changing a bid consists of simply setting an interval, the user agent can use a multicast service to send out the same interval to multiple auctioneers. This would essentially make the communication delays logarithmic with respect to number of hosts.

4.9 Verification

One potential problem with all auction-based systems is that auctioneers may cheat by charging more for resources than the rules of the auction dictate. However, one advantage of Tycoon is that it is market-based so users will eventually find more cost-effective auctioneers. Cost-effectiveness is an application-specific metric. For example, an application may prefer a slow host because it has a favorable network location. Users who are interested in CPU cycles would view that as a host with poor cost-effectiveness. However, in many applications, the agent can measure cost-effectiveness fairly accurately.

The measured cost-effectiveness is then used as the host weight for the best-response algorithm. This algorithm will automatically drop a host from bidding when it sees that it is significantly less cost-effective than the others. Effectively, Tycoon treats a cheating host as a host with poor cost-effectiveness. Therefore, sophisticated techniques to detect or prevent cheating are not necessary. If no agent wants to spend credits at a cheating auctioneer, the monetary incentive to cheat is greatly reduced.

5. EXPERIMENTAL RESULTS

In this section we present the results of a series of experiments aimed at highlighting the different characteristics of a number of resource allocation algorithms. We investigate how the algorithms adapt to changes in utility over time, number of users, number of hosts, and host preferences.

5.1 Overview

The experiments are divided into two sets of tests; single-host resource allocation, and resource allocation across multiple hosts. In the former set we study the allocation of CPU cycles on a host between competing users. In the latter we look at algorithms for distributing the workload across a number of hosts with potentially different load and user preferences. The algorithms are analyzed based on latency, economic efficiency in terms of social welfare, and fairness in terms of envy-freeness, defined as follows:

Efficiency (Price of Anarchy). For an allocation scheme $\omega \in \Omega$, denote by $U(\omega) = \sum_i U_i(\mathbf{r}_i)$ the social welfare under ω. Let $U^* = \max_{\omega\in\Omega} U(\omega)$ denote the optimal social welfare — the maximum possible aggregated user utilities. The efficiency at an allocation scheme ω is defined as $\pi(\omega) = \frac{U(\omega)}{U^*}$. Let Ω_0 denote the set of the allocation at the Nash equilibrium. When there exists Nash equilibrium, i.e. $\Omega_0 \neq \emptyset$, define the *efficiency* of a game Q to be $\pi(Q) = \min_{\omega\in\Omega_0} \pi(\omega)$.

It is usually the case that $\pi < 1$, i.e. there is an efficiency loss at a Nash equilibrium. This is the *price of anarchy* [29] paid for not having central enforcement of the user's good behavior. This price is interesting because central control results in the best possible outcome, but is not possible in most cases.

Fairness. While the definition of efficiency is standard, there are multiple ways to define fairness. We consider envy-freeness [41], used extensively in Economics. Enviness concerns how the user perceives the value of the share assigned to him, compared to the shares other users receive. Within such a framework, define the *envy-freeness* of an allocation scheme ω by $\rho(\omega) = \min_{i,j} \frac{U_i(\mathbf{r}_i)}{U_i(\mathbf{r}_j)}$. When $\rho(\omega) \geq 1$, the scheme is known as an envy-free allocation scheme. Likewise, the envy-freeness $\rho(Q)$ of a game Q is defined to be $\rho(Q) = \min_{\omega\in\Omega_0} \rho(\omega)$.

5.2 Setup

5.2.1 Configuration

There are two actors involved in the experiments, the providers hosting resources, and end-users consuming resources. To simplify the discussion we will hereafter refer to a communicating pair of these actors as a user and assume that both actors have the same utility function based on the latency of the requests being served.

To investigate how the algorithms adapt to changes over time we introduce distinct time intervals. Within each time interval we keep the conditions and settings constant, but we change certain parameters such as load, utility and host preferences between the intervals. The intervals are used as a means to allow algorithms to adjust to changes in settings. Values depicted in the experiment graphs are calculated as averages over all data collected in an interval. Four intervals are used where the first two have the same configuration as the last two in order to compensate for non-deterministic overhead. The interval time used is two minutes, so, T, the total test time is approximated as

$$T = 8 n_v n_a$$

where n_v is the number of variations investigated and n_a is the number of algorithms compared. A typical benchmark experiment has $n_v = 5$ and $n_a = 3$, and would hence run for about 2 hours, which is a time frame within which a stable external noise impact can be assumed.

A virtual machine is created for each user on all benchmark hosts. These virtual machines are then given shares of the CPU according to the auctioneer model described in [24] and implemented in Tycoon. The actual application that is run in the experiments by all users on their associated virtual machines is a CPU intensive script that can be customized

to run with different intensities. Virtual machines created on a physical host compete for the same CPU cycles.

The physical host machines have around 50GB of disk, 2GB of RAM, and two 2.8Ghz CPUs, whereas the virtual machines, controlled by Xen 2, have 2GB of disk, and varying CPU and RAM shares according to the Tycoon auctioneer allocation granted (in 10 second intervals). The machines are all running Fedora Core 3.

All clients are launched from separate virtual machines with equal resource shares to minimize client side overhead in the results.

The following table summarizes the experiment configurations. The values in the **Importance** column is the ratio between the standard deviation and mean task importance in percent. The **Pref.** column shows the ratio between the standard deviation and mean host preference in percent. Both of these values are calculated across all users and hosts. The specifics of the algorithms compared will be explained in more detail below.

Exp.	Algorithms	Hosts	Users	Importance	Pref.
I	TY, OP, PS	1	2	0..90	N/A
II	TY, OP, PS	1	2..8	50	N/A
III	BR, TU, PS	2..26	8	50	50
IV	BR, TU, PS	26	8	50	0..31

The number of hosts chosen in the experiments were limited by the number of physical machines we could isolate in our benchmark cluster. The number of users were limited by the number of active virtual machines that can be created with Xen on a physical machine without causing too much external overhead dominating the results. However, it should be noted that we exercise peak concurrent load for all users across all machines, so in a more realistic scenario, where not all users are active at the same time on all their hosts, virtual machine hibernation could be used to serve a larger number of users. In our experiments we simply maximized the load that our physical machines could handle to get a better understanding of its scalability properties. The user task importance was chosen to vary from 0 to 90%, as a higher importance variance simply would yield the same results for the optimal and the Tycoon proportional share mechanism, due to low-bid users not getting a high enough CPU share to run their jobs, as Experiment I below shows. The host preference variance configuration was chosen to allow a differentiation in the experiment results. So to summarize, the rational behind choosing these configurations was rather to optimize the number of properties we could investigate in our limited test environment as opposed to trying to mimic some imagined application load. This also stresses the point that the Tycoon infrastructure is application independent, and can serve any kind of jobs ranging from continuous service provisioning applications to large-scale task farming applications. Currently ongoing work includes analyzing the feasibility of Tycoon with real application and user loads from different domains, such as for Scientific HPC Grid applications.

5.2.2 Efficiency Evaluation

We define the average throughput of all requests for user i in interval t as:

$$\bar{h}_{i,t} = \sum_{j=1}^{n_{i,t}} (l_{i,j})^{-1}$$

where $n_{i,t}$ is the number of requests submitted in interval t by user i and $l_{i,j}$ the latency of request j of user i.

Now, the utility of a user i in time interval t can be calculated as:

$$U_{i,t}(m_{i,t}) = m_{i,t}\bar{h}_{i,t}$$

where $m_{i,t}$ is the importance of the requests of user i in time interval t. Chun [10] and others have argued for a utility function that allows some deviation in latency before decaying. The above simplified utility function was chosen for the experiments in order to react to minor changes in the test configuration more quickly. There is nothing in the Tycoon system prohibiting use of more advanced utility functions.

The social welfare of interval t is calculated as:

$$S_t = \sum_{i=1}^{k} U_{i,t}$$

where k is the number of users.

5.2.3 Fairness Evaluation

The overall envy does not increase when there are more users to envy for the same allocation. However, the overall envy does increase if there are more allocations where a user envies other users. To reflect this fact we first calculate the minimum envy-freeness value in an interval for each user, and then take the average of those values.

Envy-freeness in an inteval t for allocation scheme ω is thus given by:

$$\rho(\omega) = \frac{1}{k}\sum_{i=1}^{k}(\min_j \frac{h_{i,t}}{h_{j,t}}, 1)$$

If all users are given equal CPU shares in all intervals, ρ should be 1 and the system is then envy-free by definition.

5.3 Single Host Results

This set of experiments study the resource allocation to tasks submitted by competing users on a single host. Three different algorithms are compared focussing on CPU allocation. *Proportional Share* (PS) allocates the same share to all users within all time intervals. *Tycoon* (TY) places a bid proportional to the importance of tasks in a certain interval. *Optimal* (OP) allocates the resource to the user with the most important task in a certain interval. The importance of the tasks of the users in the different intervals are designed to vary with increasing deviation or fluctuations, where the aggregate task importance of all users across all intervals is kept constant.

5.3.1 *Experiment I: Varying Task Importance*

The goal of this experiment is to determine how efficiently the different algorithms handle high-priority tasks.

Two clients are launched against two virtual machines allocated on the same physical host. The clients run a remote CPU intensive task repeatedly for the full duration of each interval and the latencies and importance of the tasks for that interval and user are recorded.

The variance of the task importance between users and across intervals is increased. Figure 2 shows the average latency in the different intervals. Here we can see that the algorithms perform equally well. But the comparison cannot detect if some important tasks could not run at a desired performance level because of lower priority tasks occupying the resources. The point here, is that response time graphs typically used to evaluate the performance of a system may be insufficient in determining the overall efficiency of the infrastructure.

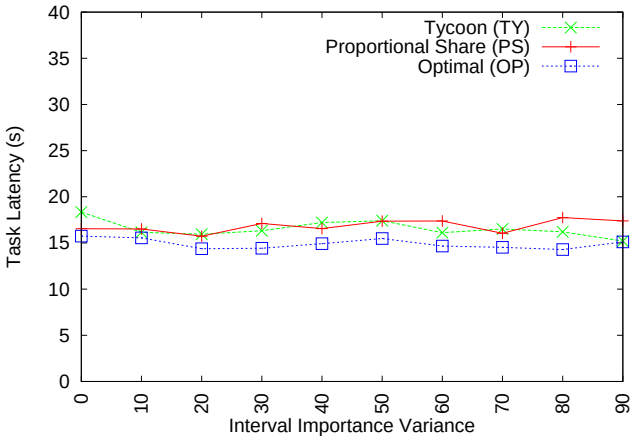

Figure 2: Experiment I - Latency with varying task importance

Figure 3 shows the social welfare of the system (as defined in §5.2.2) for different variances of task importance over time. TY increases steadily and is always within OP whereas PS remains constant with a linear increase in task importance variance.

The results of this benchmark can be verified intuitively by comparing the behavior of the social welfare functions when the importance variance increases. For PS we have (across all intervals):

$$S_{PS} = k\sum_{j=1}^{2n}(\bar{m} + (-1)^j\sigma)\frac{1}{k\bar{l}} = 2n\frac{\bar{m}}{\bar{l}}$$

where σ is the standard deviation of m, $\bar{m}$ the mean of m across all intervals in the experiment, $\bar{l}$ is the latency of a task in case of no competition. It should also be noted that we use a two-point distribution of importance ($m+\sigma$, $m-\sigma$) between competing users to make the results more easily tractable across the intervals. If a continuous distribution had been chosen a much larger number of intervals would have had to be investigated, so it was simply a time saving simplification. We also wanted to focus on how Tycoon handles different deltas in user task importance within individual intervals, as opposed to some average across all intervals, so a more complicated distribution across intervals would not have added much value to the results.

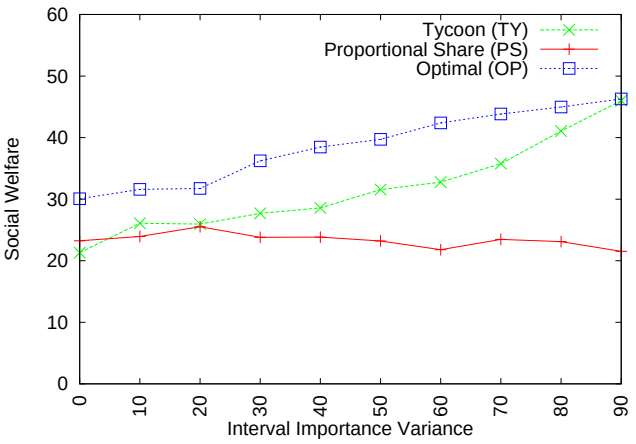

Figure 3: Experiment I - Social welfare with varying task importance

For example, $n = 2$, $\bar{m} = 50$, $\bar{l} = 9$ yields a constant social welfare of PS of approximately 22.2, which could be compared to the results in Figure 3.

For OP we have:

$$S_{OP} = \sum_{j=1}^{2n}\frac{(\bar{m}+\sigma)}{\bar{l}} = 2n\frac{(\bar{m}+\sigma)}{\bar{l}}$$

Finally, for TY we have:

$$S_{TY} = k\sum_{j=1}^{2n}(\bar{m} + (-1)^j\sigma)\frac{1}{\bar{l}}\frac{\bar{m} + (-1)^j\sigma}{\sum_{s=1}^{k}\bar{m} + (-1)^s\sigma} =$$

$$k\sum_{j=1}^{2n}\frac{(\bar{m} + (-1)^j\sigma)^2}{\bar{l}k\bar{m}} =$$

$$2n(\frac{(\bar{m}+\sigma)^2 + (\bar{m}-\sigma)^2}{\bar{l}\bar{m}}) = \frac{2n(\bar{m}^2+\sigma^2)}{\bar{l}\bar{m}}$$

$$\sigma = 0 \Rightarrow S_{TY} = \frac{2n\bar{m}}{\bar{l}} = S_{PS},$$

$$\sigma = \bar{m} \Rightarrow S_{TY} = \frac{2n2\bar{m}^2}{\bar{l}\bar{m}} = \frac{2n2\bar{m}}{\bar{l}} = \frac{2n(\bar{m}+\sigma)}{\bar{l}} = S_{OP},$$

$$0 \leq \sigma < \bar{m} \Rightarrow \mathrm{S_{PS}} \leq S_{TY} < S_{OP}$$

The last implication results because S_{TY} is a monotonically increasing function.

From these equations we see that the social welfare for PS should be constant across number of users and importance variance. TY increases quadratically with a linear increase in importance variance. OP increases linearly with a linear increase in importance variance. Furter TY is equivalent to PS if there is no variance in task importance and it is always less than or equal to the optimal OP allocation. These intuitive results correspond well with the outcome of the experiment depicted in Figure 3. The PS and TY algorithms are identical with 0 interval importance variance, so the fact that these two are slightly different in the graph is an artefact of running the experiments in a real live cluster at different time instances. The three algorithm benchmarks were run sequentially, one at a time.

We can also see from the equations that all algorithms should yield a constant social welfare across number of users. We will discuss this more in the next experiment.

To summarize, the experiment shows that the TY, and OP algorithms yield superior efficiency compared to PS in a system with high importance variance, whereas OP is slightly more efficient than TY and PS when the importance of user tasks does not vary much.

5.3.2 *Experiment II: Varying Number of Users*

We now look at how the algorithms behave, in terms of overall system efficiency and fairness, when we scale up the number of competing users on a machine.

This experiment is similar to the first, but we keep the importance variance the same across all intervals (50%) and instead let the number of users vary. The first data point (2 Users) of Figure 4 should be compared to the 50% data point in Figure 3.

We can see that the social welfare is increasing slightly for all algorithms with the number of users. This differs from the constant welfare expected from the equations in the previous section. We attribute this to the test setup and the fact that overheads like bandwidth and client side processing time compared to server side CPU processing time decreases with more users. However, we see that OP still has the highest welfare score closely followed by TY. PS remains the lowest across all user configurations. The fact that the welfare for four users appears to be slightly higher with TY than with OP is likely caused by external noise (experiement idependent host load) since the experiments ran on a live cluster albeit very lightly loaded. The main result of this experiment, though, is that the differences in system efficiency between PS, TY, and OP are, to a large degree, maintained as the number of competing users on a host is scaled up.

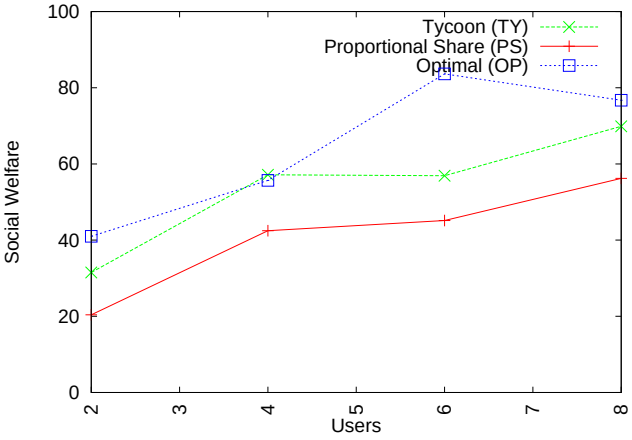

Figure 4: Experiment II - Social welfare with varying number of users

Next, we look at the fairness of the different algorithms using the same experiment result data. Our experience from analyzing envy-freeness is that it is very hard to make any kind of certain statements about the results due to noise in the data as the number of users increases. For example, contention for other resources than CPU becomes a factor. These external factors also contribute to a higher level of envy. In particular with 8 concurrent users, this noise tends to dominate the results. In lieu of this, the social welfare properties can still be maintained as seen in Figure 4, which seems to indicate white noise (zero mean). However, looking at the experimental results for 2-6 concurrently active users per physical machine in Figure 5, we can observe the behavior predicted from simulations and theoretical models. PS has the least amount of envy, since the resource shares are equally distributed, and thus should yield a theoretical envy-freeness value of 1. The decline of the PS curve can, thus, be seen as a measurement of the amount of noise in the experiment. The main reason not to use a winner-takes-it-all optimal allocation scheme is that high-bidding users can starve out others resulting in high envy. As Tycoon only lets users pay for the CPU cycles actually consumed and new auctions are hosted every 10 seconds, the difference in fairness between the OP algorithm and the TY algorithm is not as high as might be anticipated theoretically. For 2-6 concurrent VM users the fairness of TY is constantly higher, though. The limitation of number active of users on a host shown in this experiment is, as discussed in Section 5.2, more an artefact of the scalability of the local virtualization engine, Xen, than overall Tycoon scalability.

In summary, the TY algorithm exhibited higher system efficiency than PS and a higher degree of fairness than OP in multi-user experiments.

5.4 Multiple Hosts Results

In this set of experiments we compare a slightly different set of algorithms to determine what allocations to request in a cluster of hosts. *Best Response* (BR), see Section 4.6,

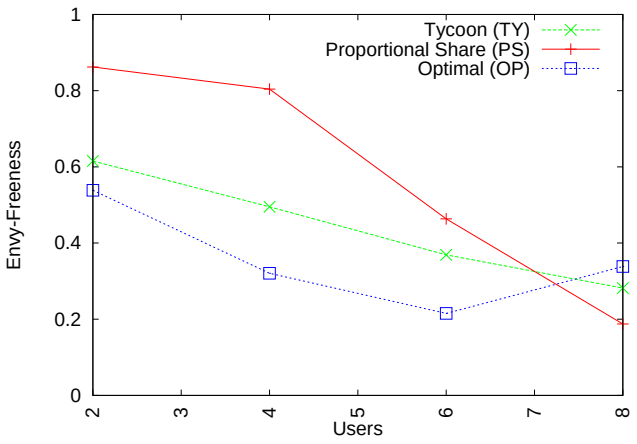

Figure 5: Experiment II - Envy-freeness with varying number of users

optimizes the overall utility by first calculating the marginal values of the hosts for each user and then determining how to divide a fixed budget into individual bids on the hosts with the highest marginal values. *Tycoon Uniform bids* (TU), distributes a fixed budget, based on task importance in a time interval, evenly across all hosts. PS gives the same allocation to all users on all hosts across all time intervals. The first experiment studes how an increasing number of hosts in the system affects the efficiency of the allocations. In the second experiment, we look at how the algorithms cope with hosts that deliver different performance levels by setting different user preferences on hosts that are considered to give a higher quality result.

When placing bids on hosts in the Best Response experiments, no margin threshold λ (as discussed in § 4.6) is set. The reason for this is that we use a continuous funding policy where more funds are given to the user in each time interval, which is a common model in large-scale Grid networks [36].

5.4.1 Experiment III: Varying Number of Hosts

The basic experiment involving multiple hosts investigates how the algorithms scale with a higher number of hosts running virtual machines on behalf of the same user. Eight user machines across 26 hosts were deployed to run the same application as in the single host tests. Clients belonging to a user send as many sequential requests as possible to all hosts in the pool concurrently. The share and performance obtained at each host is determined by the allocation algorithm, but the overall budget used to fund the processing on all hosts for a particular user remains constant across all intervals and users, as in the single host case. The BR algorithm takes advantage of both the interval task importance variance and the variance in host preferences. The TU algorithm only takes advantage of the interval importance variance whereas the PS algorithm distributes bids to hosts evenly independent of interval importance and host preferences. As a result of these properties, the social welfare of the system is highest for BR and lowest for PS, which can be seen in Figure 6.

We note that the average throughput for a host within an interval was used to evaluate the efficiency of the algorithms. So the fact that more hosts in the pool will result in a higher overall throughput for all algorithms was normalized away, to get a better indication of algorithm scalability.

The experiment shows that the system efficiency level obtained with PS as well as with TY declines slightly whereas BR maintains the same efficiency when the number of hosts is scaled up. A possible explanation is that BR can make better use of the heterogeneity of the quality of service offered on the different hosts when making allocation decisions, when the number of hosts increases.

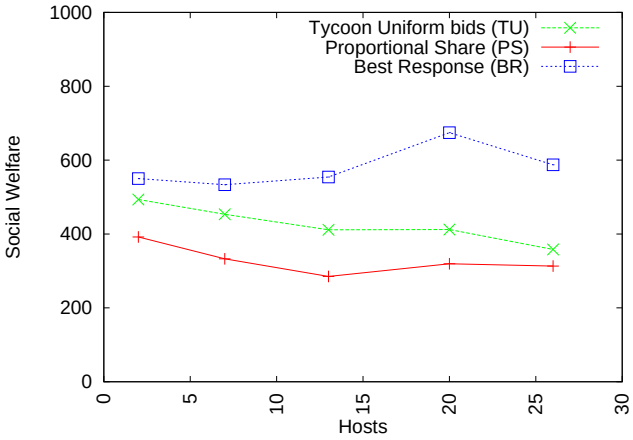

Figure 6: Experiment III - Social welfare with varying number of hosts

5.4.2 Experiment IV: Varying Host Preferences

In this final experiment, we study how the algorithms being analyzed handle different degrees of variance in host preferences. In Experiment III we used a constant host preference variance of 50% where the hosts are ordered from 1 to 26 in terms of value to the users. Here we vary this preference distribution from 0 (all hosts have the same value to the user) to about 30% (standard deviation per mean value ratio). There is then a trend in the social welfare graph in Figure 7 that BR increases with the variance whereas TR and PS remain constant. The effects are similar to those seen in Experiment III, and again support the theory that BR can make better use of QoS level differences among hosts when making allocation decisions.

To summarize the multi-host results, the BR algorithm efficiency scaled slightly better than the other algorithms, and was also able to take better advantage of differences in the level of performance that could be offered by the different hosts.

6. RELATED WORK

There are two main groups of related work in resource allocation: those that incorporate an economic mechanism, and those that do not.

One non-economic approach is scheduling (surveyed by Pinedo [32]). Examples of this approach are queuing in first-come,

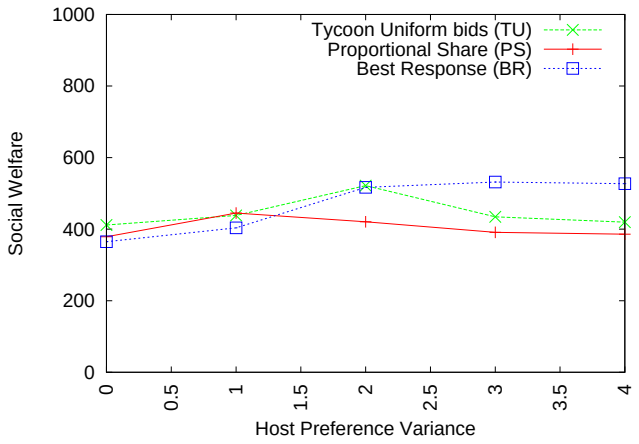

Figure 7: Experiment IV - Social welfare with varying host preference variances

first-served (FCFS) order, queueing using the resource consumption of tasks (e.g., [45]), and scheduling using combinatorial optimization [30]. These all assume that the values and resource consumption of tasks are reported accurately, which does not apply in the presence of strategic users. We view scheduling and resource allocation as two separate functions. Resource allocation divides a resource among different users while scheduling takes a given allocation and orders a user's jobs.

Examples of the economic approach are Spawn [43]), work by Stoica, et al. [38]., the Millennium resource allocator [10], work by Wellman, et al. [44], Bellagio [3]), GridBus [5], and Tycoon [24]). Spawn and the work by Wellman, et al. uses a reservation abstraction similar to the way airline seats are allocated. Unfortunately, reservations have a high latency to acquire resources, unlike the price-anticipating scheme we consider. The tradeoff of the price-anticipating schemes is that users have uncertainty about exactly how much of the resources they will receive.

Bellagio[9] uses the SHARE centralized allocator. SHARE allocates resources using a centralized combinatorial auction that allows users to express preferences with complementarities. Solving the NP-complete combinatorial auction problem provides an optimally efficient allocation. The price-anticipating scheme that we consider does not explicitly operate on complementarities, thereby possibly losing some efficiency, but it also avoids the complexity and overhead of combinatorial auctions.

In the GridBus [6] project economic schedulers and brokers have been investigated to allocate resources for task-farming applications in global Grid networks. Budget and deadline optimization algorithms have been implemented and validated through simulations and empirical studies. However, there is no mechanism support for dynamic price-setting nor personal user resource valuation.

There have been several analyses [18, 19, 20, 21, 37] of variations of price-anticipating allocation schemes in the context of allocation of network capacity for flows. Their methodology follows the study of congestion (potential) games [27, 34] by relating the Nash equilibrium to the solution of a (usually convex) global optimization problem. But those techniques no longer apply to our game because we model users as having fixed budgets and private preferences for machines. For example, unlike those games, there may exist multiple Nash equilibria in our game. Milchtaich [26] studied congestion games with private preferences but the technique in [26] is specific to the congestion game.

7. CONCLUSIONS

In this paper, we have analyzed the economic efficiency and fairness of different resource allocation algorithms, including proportional share, winner takes-it-all (socially optimal), and a set of market-based proportional share algorithms implemented in the Tycoon middleware. Benchmark experiments were set up using the Tycoon implementation to test the behavior and adaptability of the algorithms with varying number of hosts and users and with varying utilities and preferences over time. We found that the Tycoon algorithms for distributing CPU shares on a single host achieved an economic efficiency level that scaled quadratically from the proportional share level to the social optimum level with a linear increase in utility variance. When adding more users to the experiment we saw empirically that the ratios between the social welfare levels of the algorithms remained the same. The fairness of the optimal algorithm was also shown to be lower than both for proportional share and the Tycoon algorithm.

Running the same experiments across a set of hosts, we found that a best response algorithm implemented in Tycoon managed to preserve overall system efficiency better than a simple uniform bid allocation scheme, which in turn achieved a better result than the basic proportional share algorithm. This behavior was consistent across tests with varying number of hosts, users and preference degrees.

There are a variety of directions for future work. One is studying how the best response algoritm handles scientific task-farming applications in a Grid network. Another is extending the algorithm to handle both global host bid distribution and local host funding among many different resources such as CPU, disk, and bandwidth in a more optimal and consistent way. We are also interested in developing a scalable banking infrastructure. One possibility is to physically distribute the bank without administratively distributing it. The bank would consist of several servers with independent account databases. A user has accounts on some subset of the servers. A user's balance is split into separate balances on each server. To make a transfer, users find a server where both the payer and payee have an account and that contains enough funds. The transfer proceeds as with a centralized bank. Users should periodically redistribute their funds among the servers to ensure that one server failure will not prevent all payment.

8. REFERENCES

[1] http://www.vmware.com/.

[2] http://www.linux-vserver.org/.

[3] A. AuYoung, B. N. Chun, A. C. Snoeren, and A. Vahdat. Resource Allocation in Federated Distributed Computing Infrastructures. In *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure*, 2004.

[4] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *Symposium on Networked Systems Design and Implementation*, 2004.

[5] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. *Proceedings of the IEEE, Special Issue on Grid Computing*, 93(3):479–484, March 2005.

[6] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal. Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm. *Software: Practice and Experience (SPE) Journal*, 35(5):491–512, April 2005.

[7] V. Cerf and R. Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Computers*, 22(5):637–648, May 1974.

[8] Chaki Ng and Philip Buonadonna and Brent N. Chun and Alex C. Snoeren and Amin Vahdat. Addressing Strategic Behavior in a Deployed Microeconomic Resource Allocator. In *Proceedings of the 3rd Workshop on Economics of Peer-to-Peer Systems*, 2005.

[9] B. Chun, C. Ng, J. Albrecht, D. C. Parkes, and A. Vahdat. Computational Resource Exchanges for Distributed Resource Allocation. In *Unpublished*, 2004.

[10] B. N. Chun and D. E. Culler. Market-based Proportional Resource Sharing for Clusters. Technical Report CSD-1092, University of California at Berkeley, Computer Science Division, January 2000.

[11] D. D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *ACM SIGCOMM*, pages 106–114, 1988.

[12] David C. Parkes and Ruggiero Cavallo and Nick Elprin and Adam Juda and Sebastien Lahaie and Benjamin Lubin and Loizos Michael and Jeffrey Shneidman and Hassan Sultan. ICE: An Iterative Combinatorial Exchange. In *Proceedings of the ACM Conference on Electronic Commerce*, 2005.

[13] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, 2003.

[14] K. J. Duda and D. R. Cheriton. Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose schedular. In *Symposium on Operating Systems Principles*, pages 261–276, 1999.

[15] M. Feldman, K. Lai, and L. Zhang. A Price-Anticipating Resource Allocation Mechanism for Distributed Shared Clusters. In *Proceedings of the ACM Conference on Electronic Commerce*, 2005.

[16] D. Ferguson, Y. Yemimi, and C. Nikolaou. Microeconomic Algorithms for Load Balancing in Distributed Computer Systems. In *International Conference on Distributed Computer Systems*, pages 491–499, 1988.

[17] I. Foster and C. Kessleman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[18] B. Hajek and S. Yang. Strategic Buyers in a Sum Bid Game for Flat Networks. Manuscript, `http://tesla.csl.uiuc.edu/~hajek/Papers/HajekYang.pdf`, 2004.

[19] R. Johari and J. N. Tsitsiklis. Efficiency Loss in a Network Resource Allocation Game. *Mathematics of Operations Research*, 2004.

[20] F. P. Kelly. Charging and Rate Control for Elastic Traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[21] F. P. Kelly and A. K. Maulloo. Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Operational Research Society*, 49:237–252, 1998.

[22] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, 2003.

[23] K. Lai, B. A. Huberman, and L. Fine. Tycoon: A Distributed Market-based Resource Allocation System. Technical report, arXiv, 2004. http://arxiv.org/abs/cs.DC/0404013.

[24] K. Lai, L. Rasmusson, E. Adar, S. Sorkin, L. Zhang, and B. A. Huberman. Tycoon: an Implemention of a Distributed Market-Based Resource Allocation System. Technical Report arXiv:cs.DC/0412038, HP Labs, Palo Alto, CA, USA, Dec. 2004.

[25] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7), July 2004.

[26] I. Milchtaich. Congestion Games with Player-Specific Payoff Functions. *Games and Economic Behavior*, 13:111–124, 1996.

[27] D. Monderer and L. S. Sharpley. Potential Games. *Games and Economic Behavior*, 14:124–143, 1996.

[28] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable Wide-Area Resource Discovery. Technical report, U.C. Berkeley, July 2004.

[29] C. H. Papadimitriou. Algorithms, Games, and the Internet. In *Symposium on Theory of Computing*, 2001.

[30] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Dover Publications, Inc., 1982.

[31] L. Peterson, T. Anderson, D. Culler, , and T. Roscoe. Blueprint for Introducing Disruptive Technology into the Internet. In *First Workshop on Hot Topics in Networking*, 2002.

[32] M. Pinedo. *Scheduling*. Prentice Hall, 2002.

[33] O. Regev and N. Nisan. The Popcorn Market: Online Markets for Computational Resources. In *Proceedings of 1st International Conference on Information and Computation Economies*, pages 148–157, 1998.

[34] R. W. Rosenthal. A Class of Games Possessing Pure-Strategy Nash Equilibria. *Internation Journal of Game Theory*, 2:65–67, 1973.

[35] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[36] T. Sandholm, P. Gardfjell, E. Elmroth, L. Johnsson, and O. Mulmo. An ogsa-based accounting system for allocation enforcement across hpc centers. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 279–288, New York, NY, USA, 2004. ACM Press.

[37] S. Sanghavi and B. Hajek. Optimal Allocation of a Divisible Good to Strategic Buyers. Manuscript, `http://tesla.csl.uiuc.edu/~hajek/Papers/OptDivisible.pdf`, 2004.

[38] I. Stoica, H. Abdel-Wahab, and A. Pothen. A Microeconomic Scheduler for Parallel Computers. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 122–135, April 1995.

[39] R. Tijdeman. The Chairman Assignment Problem. *Discrete Mathematica*, 32, 1980.

[40] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *Proceedings of Operating Systems Design and Implementation*, December 2002.

[41] H. R. Varian. Equity, Envy, and Efficiency. *Journal of Economic Theory*, 9:63–91, 1974.

[42] H. R. Varian. Economic Mechanism Design for Computerized Agents. In *Proc. of Usenix Workshop on Electronic Commerce*, July 1995.

[43] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A Distributed Computational Economy. *Software Engineering*, 18(2):103–117, 1992.

[44] M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason. Auction Protocols for Decentralized Scheduling. *Games and Economic Behavior*, 35:271–303, 2001.

[45] A. Wierman and M. Harchol-Balter. Classifying Scheduling Policies with respect to Unfairness in an M/GI/1. In *Proceedings of the ACM SIGMETRICS 2003 Conference on Measurement and Modeling of Computer Systems*, 2003.

Market-Based Resource Allocation using Price Prediction in a High Performance Computing Grid for Scientific Applications

Thomas Sandholm
KTH – Royal Institute of Technology
Center for Parallel Computers
SE-100 44 Stockholm, Sweden
sandholm@pdc.kth.se

Kevin Lai
Hewlett-Packard Laboratories
Information Dynamics Laboratory
Palo Alto, CA 94304, USA
kevin.lai@hp.com

Jorge Andrade Ortíz and Jacob Odeberg*
KTH – Royal Institute of Technology
Dept. of Biotechnology
SE-100 44 Stockholm, Sweden
{jacob,andrade}@biotech.kth.se

Abstract

We present the implementation and analysis of a market-based resource allocation system for computational Grids. Although Grids provide a way to share resources and take advantage of statistical multiplexing, a variety of challenges remain. One is the economically efficient allocation of resources to users from disparate organizations who have their own and sometimes conflicting requirements for both the quantity and quality of services. Another is secure and scalable authorization despite rapidly changing allocations.

Our solution to both of these challenges is to use a market-based resource allocation system. This system allows users to express diverse quantity- and quality-of-service requirements, yet prevents them from denying service to other users. It does this by providing tools to the user to predict and tradeoff risk and expected return in the computational market. In addition, the system enables secure and scalable authorization by using signed money-transfer tokens instead of identity-based authorization. This removes the overhead of maintaining and updating access control lists, while restricting usage based on the amount of money transferred. We examine the performance of the system by running a bioinformatics application on a fully operational implementation of an integrated Grid market.

1. Introduction

The combination of decreasing cost for network bandwidth and CPU performance and the availability of open-source distributed computing middleware has led the high-performance computing community away from monolithic supercomputers to low-cost distributed cluster solutions. This model of computing allows users with bursty and computationally demanding tasks to share and use compute resources on demand. This usage model, aka utility computing [24], Grid computing [22], peer-to-peer compute farming [28], or Global Computing [23], has been applied to solve diverse technical computing problems in fields such as, bioinformatics [9], high-energy physics [10], graphics rendering [3], and economic simulation [2].

One common question remains: how to manage the allocation of resources to users? One challenge is that users are from disparate organizations and have their own and sometimes conflicting requirements for both the quantity and quality of services. For example, academic Grid projects [37, 4, 38, 8, 7, 6, 5, 10] typically require resources for throughput sensitive, long-running batch applications, while industrial utility computing offerings [14, 24, 26] typically require response-time sensitive, interactive, and continuous application server provisioning. One common solution is to have separate resource allocation mechanisms for different applications. However, this merely shifts the problem from reconciling the resource requirements of different applications to reconciling the resource requirements of different mechanisms.

Another challenge is that users have a web of relation-

*also affiliated with Karolinska Hospital, Gustav V Research Institute, Dept. of Medicine, Atherosclerosis Research Unit, Stockholm, Sweden

ships with regard to how they wish to share resources with each other. For example, one site may wish to share resources with a remote site, but only when demand from local users is low. Another example is that a site may wish to be *reciprocative*, where it only shares resources with sites that share resources with it. One common solution is to use access control lists (ACLs) to authorize access to resources. The problem is that managing ACLs is difficult because many users could potentially access a site, a site has many different types of resources, each of which may need a separate ACL, and the degree of access that each user has could change with each access. For example, as a user from site A uses host X at site B, site B would want to decrease the ability of other site A users from being able to consume resources at host X and possibly other hosts at the site. The dynamic updating of this amount of data not only increases overhead and development time, but could lead to inconsistencies that allow exploitation of the system.

Instead, we examine a market-based approach to resource allocation in Grids. A number of models and mechanisms for electronic markets, and computational economies have been proposed [40, 30, 36, 35, 33, 12, 27]. In previous work we have presented and analyzed Tycoon [31], a market-based resource allocation system for shared clusters. Here we focus on how market-based techniques address issues in a Grid environment. More specifically, our contributions are as follows:

• **A Tycoon controlled cluster scheduler for Grid execution managers.** The NorduGrid/ARC *meta-scheduler* [38] used by many academic Grid projects in Europe, such as SweGrid [37], to schedule large-scale scientific jobs across a collection of heterogeneous HPC sites using a uniform job submission and monitoring interface, was integrated with Tycoon. The integration allows the large existing user base of academic Grids to bid for and use resources controlled by economic markets seamlessly. We are also working on integrating our scheduler with Globus Toolkit 4 [21].

• **Pilot application experiments.** We use a bioinformatics pilot application currently deployed in an academic Grid to verify our model and implementation. A bioinformatics application scanning and analyzing the complete human proteome using a blast-based similarity search was run on a Tycoon cluster and evaluated.

• **Price prediction models and mechanisms.** We discuss, implement and analyze models and mechanisms to predict cost of resources and give guarantees of QoS (Quality of Service) levels based on job funding. In a spot market as implemented by Tycoon it can be hard to predict the future price of a resource and know how much money should be spent on funding a job with a specific set of requirements. To that end we provide a suite of lightweight prediction capabilities that can give users guidance regarding what performance levels to expect for different levels of funding of a job.

• **A security model combining identity and capability-based access control.** In academic Grid networks it is important to identify all users securely because a user's identity, and membership in virtual organizations, can automatically give access to shared resources. In electronic markets, however, the key question is whether you can present proof of payment for a resource. Our model allows Grid users to make use of electronic money transfer tokens, or checks, that can be used to pay for and gain access to resources to be used by a compute job, as well as to specify its priority and thereby 'buy' a certain level of QoS.

As a result, we believe that the combination of Grid and market mechanisms is a promising and viable approach to sharing resources in an economically sustainable way while ensuring fairness and overall system efficiency.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the design and discuss how it meets our goal. In Section 3, we delve more deeply into implementation specifics. Our price prediction analysis is presented in Section 4. Section 5 contains experimental results from running the bioinformatics application on our integration testbed. We describe related work in Section 6. We conclude by discussing some limits of our model and future work in Section 7.

2. Architecture Overview

2.1. A Case for Grid Markets

High-end compute resources, such as Grid-enabled High Performance Computing (HPC) clusters, are necessary for many scientific computing applications. These applications can use more resources to scan larger input data sets, provide a higher resolution for simulations, or simply complete the same amount of work faster. In other words, they exhibit a continuous utility curve where a larger resource share results in a higher utility.

Traditional queueing and batch scheduling algorithms assume that job priorities can simply be set by administrative means. In large-scale, cross-organizational, and potentially untrusted Grids, this can be expensive and slow, and allocations may not reflect the true relative priorities of jobs. Determining the relative priorities of applications requires a truth revelation mechanism like a market [12, 27, 41, 39]. This provides an incentive for uses to accurately prioritize their applications.

2.2. Tycoon

In this paper, we apply a previously described market-based resource allocation system (Tycoon) [29] in the Grid

context. In this section, we briefly review this system before describing the security and prediction extensions that are the focus of this paper. Please see the previous publication for additional details.

The main characteristics of Tycoon are its decentralized and continuous markets. Each host that contributes resources to a Tycoon network runs its own market. This reduces the dependency on centralized components, making the system more scalable, fault-tolerant, and agile in the allocation of resources. The continuous market allows users to bid and receive resources at a high frequency (10 seconds by default), which allows applications to start quickly and react to quickly changing load. This is more important for service-oriented applications like web servers and databases than the typical Grid applications that runs for days. Sharing the same infrastructure across these different types of applications allows better statistical multiplexing.

Another characteristic of Tycoon is the use of virtualized resources. This allows multiple applications to share the same host, which is useful both for applications that do not require a whole host and applications that have highly variable demand. Tycoon currently uses the Xen paravirtualization system [19], which imposes an overhead of 1%-5% for typical workloads.

The architecture of Tycoon is composed of the Bank, which maintains information on users like their credit balance and public keys, the Service Location Service, which maintains information on available resources, and Auctioneers, which run on each host and manage the market used to allocate resources on that host.

One of the main issues in this kind of distributed market is the efficiency relative to an ideal centralized market. Feldman, et al. [20] show that this market achieves both fairness and economic efficiency in the equilibrium when users use the Best Response optimization algorithm. Briefly, this algorithm solves the following optimization problem for a user:

$$\text{maximize } U_i = \sum_{j=1}^{n} w_{i,j} \frac{x_{ij}}{x_{ij}+y_j} \text{ subject to} \quad (1)$$

$$\sum_{j=1}^{n} x_{ij} = X_i, \text{ and } x_{ij} \geq 0. \quad (2)$$

where U_i is the utility of user i across a set of resources, $w_{i,j}$ is the preference of machine j as perceived by user i, for example the CPU capacity of the machine, $x_{i,j}$ is the bid user i puts on host j, y_j the total of all bids or the price of host j, and finally X_i is the total budget of user i.

One contribution in this work is to expose this Best Response algorithm to Grid HPC users, to allow them to easily use compute resources in a competitive market. Next we describe the architecture of this integration.

2.3. Grid Market Architecture

Our solution is novel in the sense that we maintain the overall Public Key Infrastructure (PKI) security model of the Grid, but introduce supply and demand driven dynamic pricing and resource share negotiation on a spot market within a virtual Grid cluster. Differentiated services are offered in an incentive compatible manner where the Grid user can attach a check-like token to jobs to pay for the resources to be consumed. The user needs to be fully authenticated using a Grid PKI handshake, but the authorization step is based on a capability composed of the funds transferred to the scheduling agent. The scheduling agent uses the Best Response algorithm described in the previous section to distribute and place bids on compute nodes. Virtual machines are created dynamically, with the appropriate QoS levels automatically configured in proportion to the bids placed. Job stage-in, execution, monitoring, performance boosting (by adding funds) and stage-out are all handled by the agent. Figure 1 depicts the overall architecture of the system. The Tycoon infrastructure is integrated with the Grid Job manager as a local scheduling system, fully transparent to the end-users. The next section describes the design and implementation in more detail.

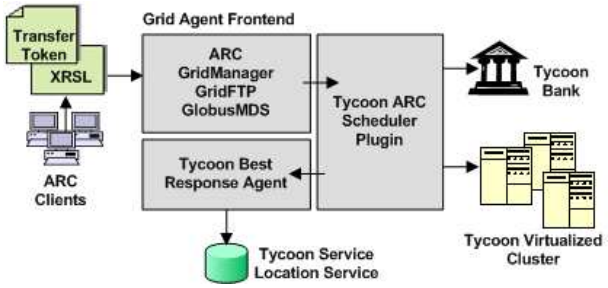

Figure 1. Tycoon Grid Market Architecture.

3. Grid Market Implementation

This section describes our integration of Tycoon with the Grid. Our approach is to integrate Tycoon with the infrastructure currently deployed in the SweGrid project, a national Grid in Sweden spanning six sites and a total of 600 compute nodes interconnected by the 10GB/s GigaSunet network. SweGrid has been in operation for a year and a half. The sites run the Nordugrid ARC middleware, which is based on the Globus toolkit, and currently connects about 50 HPC sites in 12 countries. Both ARC and Globus focus on providing an abstracted local cluster scheduler and execution manager so that the job submission mechanism and interface is the same regardless of whether PBS, LSF, Sun Grid Engine, etc., is used locally at the site.

As a result, to enable Tycoon for Grid users seamlessly, our approach was to write a Tycoon scheduler plugin for ARC to simplify the transition of SweGrid users to this new infrastructure. Integration directly as a Globus (GT4) plugin is work in progress. The Tycoon cluster is exposed as any other ARC cluster both for monitoring and job submission purposes, with the only difference being that the cluster is virtualized and thus reports number of virtual CPUs as opposed to physical compute node CPUs.

In ARC, the compute job requirements are describe in an XRSL (extended Globus Resource Specification Language) file. The CPU or WallTime XRSL attribute is mapped to the Tycoon resource bid deadline. The transfer token, described in more detail in the next section, is mapped to the total budget to be used within this deadline. Finally, the count attribute describes how many concurrent virtual machines or (virtual CPUs) the job requires. Mechanisms are provided to allow a job running in a virtual machine to access its ordinal number within such a batch of sub-jobs to, for example, process a different input data set. The bid distribution is determined by the Best Response algorithm of Tycoon. It calculates the optimal bids to maximize the overall utility gained from running a job on a set of host.

The ARC runtime environment allows users to specify what software needs to be installed in order to run the job. This software is installed automatically in the virtual machine using Yellow dog Updater, Modified (yum)[1]. The Tycoon ARC plugin also handles job stage-in, stage-out and output checkpoint monitoring. Jobs that have been submitted may be boosted with additional funding to complete sooner, and if the money deposited into host accounts has not been used (Tycoon only charges for resources actually used not bid for) the outstanding balance will be refunded to the user. Dynamic pricing, accounting and billing thus all happen automatically by means of the Tycoon infrastructure. Furthermore, the QoS specified in the XRSL description is automatically enforced by configuring custom virtual machines on demand when jobs require them. To limit the overhead of virtual machine creation, a user may reuse the same virtual machine between jobs submitted on the same physical host. However, no application data or scratch space is shared by different jobs.

Figure 2 shows a screenshot of the ARC Grid Monitor monitoring the Tycoon cluster. Note that since the number of CPUs are the number of virtual machines currently created, it can increase dynamically up to a maximum of about 15 times the number of physical nodes. Thus, for the current deployment of 40 hosts, a maximum of 600 virtual single CPU compute nodes can be offered concurrently to users by the Tycoon Grid. We could easily implement a virtual machine purging or hibernation model that could increase this number further if not all 600 machines need to be used at the same time, with the penalty of more overhead to setup a job on a virtual machine.

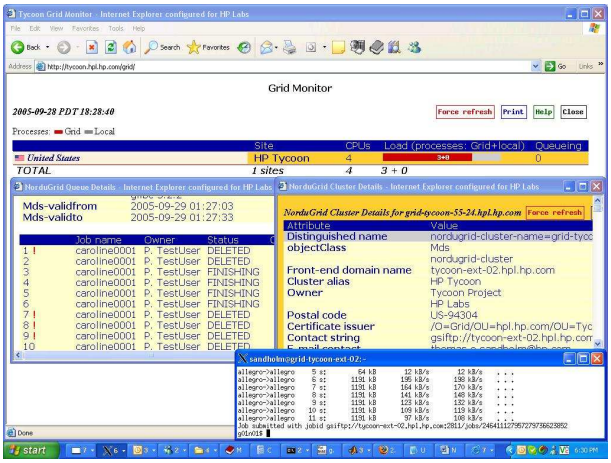

Figure 2. Screenshot of Nordugrid/ARC Tycoon monitor.

In addition to the virtualization of compute nodes, the cluster is also virtualized, in the sense that the Tycoon architecture is flat without hierarchies. This allows the submission agent to pick up compute nodes from any available physical cluster. Most of the current machines are at HP Labs in Palo Alto, California, but others are owned by Intel Research in Oregon, by Singapore, and by the Swedish Institute of Computer Science in Stockholm. The current limitation of 40 physical machines is only an artifact of the current state of the virtual cluster, and can grow dynamically as more clusters and nodes are added to the Tycoon network. Regardless of whether the compute node is local to the submission agent the host funding, job stage-in, job execution, job monitoring and job-stage out will all be done the same way. Finally, the agent itself can be replicated and partitioned to pick up a different set of compute nodes. The ARC meta-scheduler could then be used to load balance and do job to cluster matchmaking between the replicas. We therefore believe that this model will scale well as the number of compute nodes and virtual machines on these compute nodes increase.

3.1. Security Design

Our goal is to make the security integration as seamless as possible to the end-users, which means allowing Grid users to reuse their credentials when accessing a Tycoon cluster. To access a Tycoon cluster one needs a Tycoon bank account. So our task became equivalent to mapping a Grid certificate to a Tycoon bank account. Our approach is to transfer money from the user to the resource broker and map the proof of the transfer to a Grid identity, the Distinguished Name (DN) of the user.

This approach requires no manual access control list setup and it allows the user to keep both the Grid identity private key and Tycoon bank account private key on the local machine only. Furthermore, it does not require any changes to the existing Tycoon services. The user transfers money to the resource broker's bank account and then signs the receipt together with a Grid DN. The mapping can thus be decided independent of the transfer and can therefore also be used by arbitrary Grid users who do not have any Tycoon infrastructure deployed. On the resource side it is verified that the money transfer was indeed made into the broker account and that the transfer token has not been used before. The signature of the DN mapping is also verified to make sure that no middleman has added a fake mapping. Once the transfer token has been verified a new sub-account to the broker account is created and the money verified is transferred into this account by the broker. The new account is then used to fund and create host accounts used to run the job on behalf of the Grid user.

4. Price and Performance Prediction

4.1. Motivation

A challenge in any market-based resource allocation system is providing predictable performance. A variety of solutions exist, including reservations, future markets [11], options, and SLA contracts [42, 18]. However, all of these mechanisms require the ability to predict future demand and supply. Prediction is particularly important in spot-market systems like Tycoon that lack other mechanisms to ensure predictability. The failure to predict accurately either causes users to overspend on resources or prevents them from achieving their required quality of service.

Demand prediction requires a history of resource usage and a set of analysis algorithms. Our goal is to provide both a concise representation of historical prices on the Auctioneer and efficient client-side algorithms to analyze this data.

We represent the demand using the spot-market price on each host. In Tycoon, this reflects both the usage of and demand for a resource. This information is updated every 10 seconds, which is also the reallocation interval. In addition to the instantaneous demand, we also track the average, variation, distribution symmetry, and peak behavior of the price.

To make this information useful to a wide array of applications, periodization of the data is necessary. We implement this by presenting and scoping the statistics in moving, customizable time windows. By looking at smaller time windows we may be able to make simplifying assumptions such as the assumption of a symmetric normal distribution even though such a distribution may not be a good representation of a larger window, and vice versa. To track the price distribution dynamically we implement a self-adjusting slot table recording the proportion of prices that fall into certain ranges.

We now present three high-level prediction techniques to model our price data, (1) normal distribution prediction, chosen because it is simple to implement and rest only on fundamental statistical theory; (2) autoregression prediction of time series, a very common system identification approach; and (3) portfolio selection, a well-studied technique in economic theory to reduce risk. Finally, we present the theory behind our moving window approximation and smoothing implementation.

4.2. Lightweight Single-Host Stateless Price Prediction

In this model we assume a normal probability distribution of the spot market price and calculate the budgets needed to get a certain performance level or higher with a probability guarantee, which could be translated into the probability of a job completing within a certain deadline. Based on this information we would like to recommend a user to spend a certain amount of money given a capacity requirement and a deadline.

More formally, we first assume that the price y is an outcome of the normal random variable Y

$$Y \in N(\mu, \sigma^2), y \in Y \tag{3}$$

p is the probability given by the standard cumulative distribution function Φ, with μ as the measured mean of y and σ^2 as the measured variance. In other words, p is the probability that a resource offers a price less than or equal to y given its variance and mean.

$$p = P(Y \leq \frac{y-\mu}{\sigma}) = \Phi(\frac{y-\mu}{\sigma}) \tag{4}$$

The inverse cumulative distribution function, aka the probit quantile function of the normal distribution gives us the price y to expect with a given probability p.

$$\frac{y-\mu}{\sigma} \leq \Phi^{-1}(p) \Leftrightarrow y \leq \mu + \sigma\Phi^{-1}(p) \tag{5}$$

Combining (1) and (5) gives us the probability p to get the utility U given the budget X.

$$U_i(X_i, p) \geq \sum_{j=1}^{n} w_{i,j} \frac{x_{i,j}}{x_{i,j} + \mu_j + \sigma_j\Phi^{-1}(p)} \tag{6}$$

where $x_{i,j}$ is the bid picked by the best response algorithm in (1) with budget X_i on host j for user i.

If a user knows that the deadline d can be met if a utility greater than U is obtained, we can use (6) to recommend what budget to spend to meet that deadline, and conversely

what completion time to expect given a budget. For example, the budget X required to meet the deadline d with a certainty of p can be used as a recommendation for the extra cushion of funding needed to meet the deadline with a greater probability.

We call this model stateless, since we only need to keep track of running sums to report the mean and standard deviation of the price, and no data points need to be stored.

4.3. Single-Host Price Prediction Analyzing Time Series History Data

An autoregressive, $AR(k)$, [32] model based on a time series of CPU price snapshots was implemented using the following steps:

First, the unbiased autocorrelation with N sample snapshots of x and lag k is calculated as:

$$R(k) = \frac{1}{N-|k|} \sum_{n=0}^{N-|k|-1} x_{n+|k|} x_n$$

Then the following Yule-Walker linear equation system is solved using the Levinson reformulation:

$$L\alpha = r$$

where

$$L_{i,j} = R(i-j)$$

is the Toeplitz matrix with k rows and k columns, α is the column vector of k AR coefficients to be solved, and r is a column vector of size k where

$$r_i = R(i+1)$$

Now, future values of the time series x_i can be predicted using the coefficients in α as:

$$x_{N+1} = \mu + \sum_{j=0}^{k} \alpha_j (x_{i-j} - \mu)$$

where

$$\mu = \frac{1}{N} \sum_{n=0}^{N} x_n$$

Note that we omit the zero mean normal random white noise parameter here for simplicity.

4.4. Risk Management based Performance Prediction across Multiple Hosts

We now look at another prediction model for obtaining guidance in funding resources, portfolio theory. We need to obtain the return and plot that against the risk to calculate the efficient frontier where portfolios yield the most efficient trade-off between the two parameters. The fundamental rule of the frontier is that at a given risk value the return should be maximized and conversely at a given return value the risk should be minimized. We can then apply Morkowitz's mean-variance optimization [34]. As return we select the performance of the resource calculated as number of CPU cycles per second that are delivered per amount of money paid per second (inverse of spot market price).

Given the vectors of return and risk values for the resources, we used the matrix equations from [25] to calculate the risk free portfolio as well as the efficient frontier.

By looking at the efficient frontier we can, based on our degree of risk aversion, select a portfolio with an appropriate return. The advantage of the portfolio model is that we do not have to assume a normal probability distribution of the resource price. However, a symmetric distribution around the mean is assumed and it is also assumed that there is a variance in risk between resources that can be traded off with varying mean returns.

A similar approach focusing on Value-at-Risk analysis is presented in [16]. Their approach inherits the same strength and weaknesses as the general portfolio theory presented here, but extends it to give guarantees like, *within a given time horizon, the minimal performance will be a value V with a probability P* . In contrast, the approach presented here gives guidelines of the form, *given a certain level of risk aversion and expected performance, how should you distribute your budget across a set of hosts?*

4.5. Moving Window Smoothing Theory

We first look at the technique used to calculate moving windows for the price average (mean), variation (standard deviation), asymmetry of distribution (skewness), and peak behavior (kurtosis). A high value of skewness reflects a heavy-tailed (right-skewed) distribution, and a high value of kurtosis indicates that a large portion of the standard deviation is due to a few very high price peaks.

In terms of state information we only need to keep track of the previously calculated sample moments about the mean for the first (mean), second (standard deviation), third (skewness) and forth (kurtosis) moment about the mean. The linear smoothing function is determined by the window size, where a large window size results in the previously calculated moment having a very low impact on the next moment compared to the current snapshot, and vice versa. For window size 1, the previously calculated moments are ignored as expected.

$\mu_{i,p}$ is the pth sample moment about the mean at snapshot i, x_i is the price at snapshot i, n is the number of price samples in a window, σ_i is the standard deviation of price at snapshot i (for window n), $\gamma_{1,i}$ is the price skewness at

snapshot i (for window n), and $\gamma_{2,i}$ is the price kurtosis at snapshot i (for window n)

$$\mu_{0,p} = x_0^p$$

$$\mu_{i,p} = \alpha\mu_{i-1,p} + (1-\alpha)x_i^p$$

$$\alpha = 1 - \frac{1}{n}$$

$$\sum_{j=1}^{n} x_j^p = \mu_{i,p} n \Rightarrow \sigma_i = \sqrt{\mu_{i,2} - \mu_{i,1}^2}$$

equivalently,

$$\gamma_{1,i} = \frac{(\mu_{i,3} - 3\mu_{i,1}\mu_{i,2} + 2\mu_{i,1}^3)}{\sigma_i^3}$$

and,

$$\gamma_{2,i} = \frac{(\mu_{i,4} - 4\mu_{i,3}\mu_{i,1} + 6\mu_{i,2}\mu_{i,1}^2 - 3\mu_{i,1}^4)}{\sigma_i^4} - 3$$

We now look at the price distribution smoothing for moving time windows. The approach taken is to keep track of two price distributions for each window at all times. The distributions will contain twice as many snapshots as is required by the windows and have a time lag of the same size as the window. The merged window distribution to be retrieved at an arbitrary monitoring time is then calculated by using a share of both distributions proportional to how closely they are to the desired window size in terms of number of snapshots collected.

n is the total number of prices in a window, i is the snapshot time, $n_{k,i}$ is the number of prices in distribution array k at time i, $(0..2n)$, $s_{k,j}$ is the proportion of prices in slot j in distribution array k, $r_{i,j}$ is the proportion of prices to report in slot j at snapshot time i, and $w_{i,k}$ is the proportion of distribution array k to use in r at snapshot time i

$$w_{i,k} = 1 - \frac{|n_{1,i} - n|}{n}$$

$$|n_{1,\mathrm{i}} - n_{2,\mathrm{i}}| = n$$

$$\sum_{j=1}^{n_k} s_{k,j} = 1 \Rightarrow r_{i,j} = w_{1,k}\mathrm{s}_{1,\mathrm{j}} + (1 - w_{1,k})\mathrm{s}_{2,\mathrm{j}}$$

5. Grid Application Results

In this section we present some experimental results using a Bioinformatics application targeted for Grid environments, which was developed at the Bioinformatics laboratory at the Royal Institute of Technology in Stockholm [9]. It is a trivially parallelizable bag-of-task application, which is very typical for large-scale Grids. The experiments we present here, do not consider applications with more complicated workflow-like interactions among subtasks. However, none of the experiments depend in any way on the application-specific node processing performed by this application, more than the fact that it is CPU intensive.

5.1. Bioinformatics Application

The goal of the application is to identify protein regions with high or low similarity to the rest of the human proteome. A database of the complete human proteome is analyzed with a blast sequence alignment search tool performing stepwise similarity searches using a sliding window algorithm running in parallel on a distributed compute cluster. The reason for running this application in a compute farm is twofold, the proteome database is continuously evolving and the search is computationally hard. A search on a single machine takes about 8 weeks on a single node, and a run in the SweGrid compute farm utilizing 300 nodes out of 600 takes about 22 hours.

5.2. Experiment Setup

The proteome database is partitioned into chunks that can be analyzed in parallel. One of these chunks takes approximately 212 minutes to analyze on a single node in our cluster with a 100% share of a CPU. With 30 physical machines we can thus achieve a maximum performance of 35 hours/application run to be compared with 22 hours/run in SweGrid with 600 machines. In our experiment we are letting five competing users run the same application with different funding. The application makes use of a maximum of 15 nodes out of a total of 30 physical nodes. To have the users compete against each other but not between their own sub-jobs we restrict the setup to one virtual machine per user per physical machine. Hence, a maximum of 75 virtual machines may be used at any point in time. It should be noted that the physical machines have dual processors and there may thus not be competition for a CPU on a machine even though there are multiple users running there concurrently. The user jobs are launched in sequence with a slight delay to allow the best response selection to take the previous job funding into account. This is why users 1 and 2 tend to get to run on more nodes than the other users, as the price has not gone up as much at that point. Their shares will, however, be recomputed automatically and continuously within every 10s allocation interval.

5.3. Best Response Experiments

In this set of experiments we are interested in finding out whether an economically driven resource allocation mechanism would allow us to offer differentiated QoS levels to

Table 1. Equal Distribution of Funds

Users	Time(h)	Cost($/h)	Latency(min/job)	Nodes
$1-2$	7.16	4.19	28.66	15
$3-5$	6.36	4.28	45.49	8.7

Table 2. Two-Point Distribution of Funds

Users	Time(h)	Cost($/h)	Latency(min/job)	Nodes
$1-2$	7.07	5.10	29.31	14.5
$3-5$	4.16	10.9	23.46	11

Grid application users. We measure the **Time** defined as the wall-clock time as perceived by the user to complete the task of sub-jobs, the **Cost** as the money spent during this time, the **Latency** as the number of minutes it takes for each sub-job to complete (again in wall-clock time), and the number of **Nodes** or parallel sub-jobs used by the task. We start by gauging the environment and running the test with all users having the same funding for their jobs. They should hence expect an equal share of the CPUs. We, however, note from the results summarized in Table 1 that users 3-5 received a much lower quality of service, here defined as number of jobs that can be processed within a time unit, because the best response algorithm found it too expensive to fund more than a very low number of hosts. One possible solution to this issue would be to let the user hold back on submitting if not a threshold of minimum hosts to bid on is met.

The results from a two-point distribution with users funding their jobs with $100, 100, 500, 500, 500$ dollars with a deadline of 5.5 hours is summarized in Table 2.

Here we can see that the jobs with a budget of 500 dollars caused the earlier jobs to decrease their shares to allow the more highly funded jobs to complete within their deadline. We again see that fewer hosts were given to user 3-5 but this time the performance level (latency) is better. We also see that these users pay a higher price for their resource usage, as expected.

5.4. Price Prediction Experiments and Simulations

In this set of experiments we run the same Grid application job load as in the previous experiments with the difference that we let the total budget of the users be random with a normal distribution.

Using the normal distribution analysis presented in Section 4.2, we provide a graph visualization of the price and performance guarantees a user may expect from a host. Depending on what guarantee of average performance the user wants, different curves may be followed to decide on how much to spend. For example, looking at the graph in Figure 3 a user who wants 90% guarantee that the CPU performance will be greater than 1.6GHz should spend $22/day when funding the host. There is a certain point where the curves flatten out, and that point would be the recommended budget to spend on that host to get the best performance per funding unit. For the given example it would not make sense for the user to spend more than roughly $60/day. We can also see that to get any kind of feasible performance out of the machine with at least a 80% guarantee the user needs to spend at least $10/day. In this example, we based our prediction on a time window of one day.

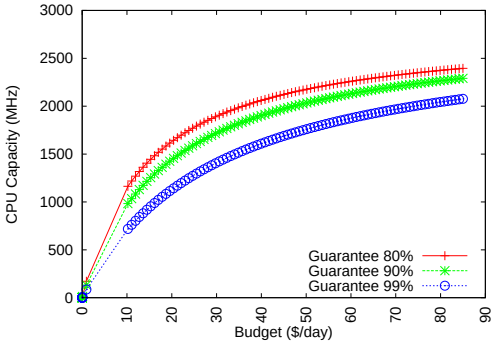

Figure 3. Normal distribution prediction with different guarantee levels

The basic AR model presented in Section 4.3 had problems predicting future prices due to sharp price drops when batch jobs completed. To overcome this issue we applied a smoothing function (cubic smoothing spline) before calculating the AR model. To verify the quality of the prediction we took a data sample of 40 hours of price history from our experimental run of Grid jobs described above. The first 20 hours were used to calculate the model and the last 20 hours were used to verify the model. The prediction error was then calculated as follows:

$$\varepsilon = \frac{1}{\mu_d n} \sum_{i=1}^{n} \sigma_i$$

where μ_d is the mean of the measured prices in the validation interval, n the number of data points in the validation interval, and σ_i the standard deviation of the prediction, measurement pair i. An AR(6) model with one hour forecasting (See Figure 4) yielded an ε of 8.96%, whereas a simple benchmark model always predicting the current price to remain for the next hour resulted in an ε of 9.44%.

Now, turning to portfolio theory (Section 4.4). There are some issues with this model concerning the definition

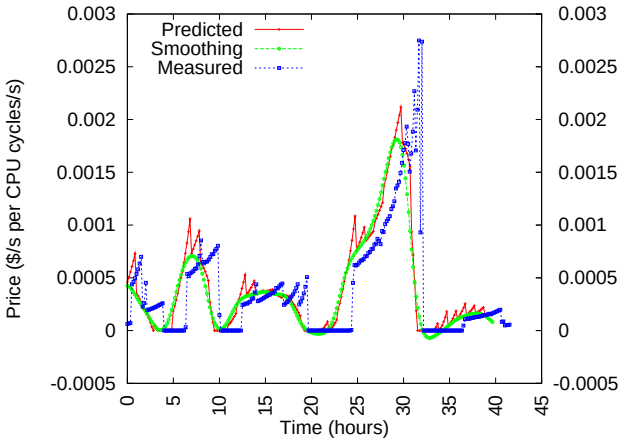

Figure 4. AR(6) prediction with a one hour forecast and smoothing function.

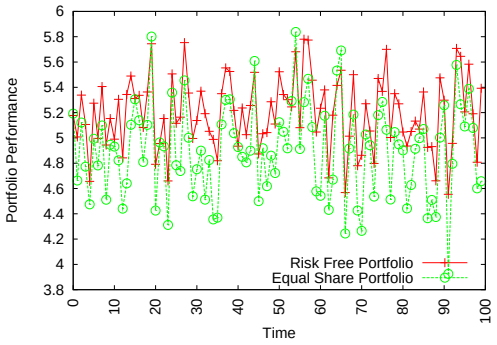

Figure 5. Risk free portfolio performance vs. equal share portfolio.

of risk and asymmetry of distributions as mentioned in Section 4.4, but we also noted in our experiments that a portfolio-based scheduler would not do as well in load balancing batches of user jobs coming in as the best response algorithm which bases its selection on the spot market price, and which could immediately move users away from high-bid machines. Portfolio theory may, however, prove useful for long term investments in hosts, e.g. when hosts should be bought to run a continuous application such as a web server. Another observation is that idle hosts tend to get 100% of the share in the portfolio, to avoid this behavior a larger time window needs to be used when collecting the mean and variance statistics from the hosts.

To test the risk hedging properties of the portfolios returned by our implementation we ran simulations where 10 hosts are picked either using the calculated risk free portfolio or equal shares. The aggregate performance over time is then measured. Individual mean host performance, performance variance, and variance of performance variances were all randomly generated with a normal distribution. The results, depicted in Figure 5 shows that downside risk could be improved by using the risk free portfolio.

Finally, we look at the distribution of prices over three time windows, a week, a day, and an hour. This data can be used to select an appropriate prediction model. For example, if the distribution resembles a normal distribution one could make use of the models described in Section 4.2, if the distribution is symmetric a portfolio analysis may be appropriate. A sample distribution graph is shown in Figure 6. It can be inferred from the graph that the prices exhibit signs of a heavy-tailed distribution (left-skewed) the last hour, mostly fall within the lowest price bracket, but are right-skewed, mostly in the most expensive bracket when considering a week or day-long window.

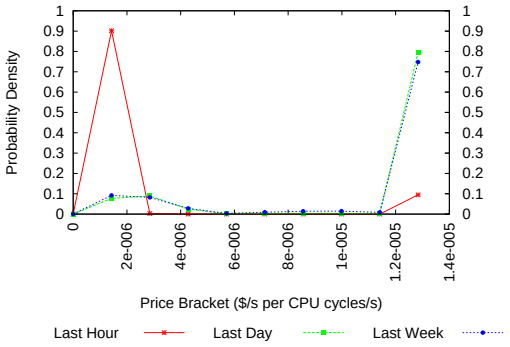

Figure 6. Price distribution within three different time windows.

To measure how accurate our window approximation is we ran a simulation of different distributions. Normal, Exponential and Beta Distributions were given a time lag of half the window size. At this point there is a maximum influence, or noise, from non-window data. The noise was generated using a uniform random distribution. We noted that normal distributions with a small standard deviation ($< 20\%$ of mean) could result in the approximation having its mean shifted slightly compared to the actual distribution. However, in general the approximations followed the actual distributions closely as seen in Figure 7 .

6. Related Work

Faucets [27] is a framework for providing market-driven selection of compute servers. Compute servers compete

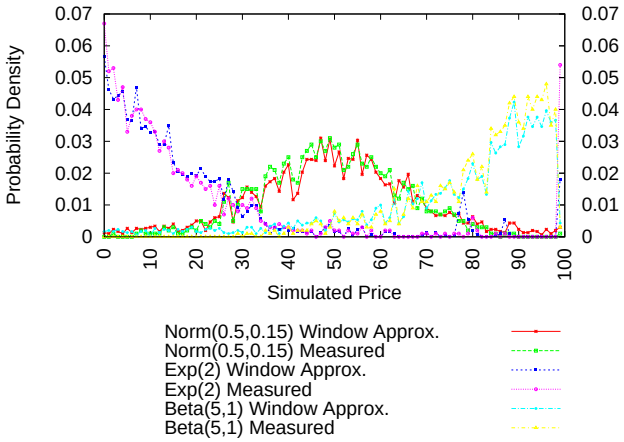

Figure 7. Window approximation of Normal, Exponential and Beta distributions.

for jobs by bidding out their resources. The bids are then matched with the requirements of the users by the Faucets schedulers. Adaptive jobs can shrink and grow depending on utilization and prioritization. QoS contracts decide how much a user is willing to pay for a job. The main difference to our work is that Faucet does not provide any mechanism for price setting. Further, it has no banking service, use central server based user-name password mechanisms, and does not virtualize resources.

Xiao et al. [43] suggest a model where users prioritize their jobs with different budgets and providers schedule jobs based on minimizing penalties from missing promised deadlines. It is argued that a user-initiated auction is more appropriate for lightly loaded system. From our experience with HPC projects like SweGrid, resources are scarce and their is competition for time slots, hence a seller-initiated auction is more appropriate for our work.

Chunlin and Layuan [17] propose a two-layered central market. In the first layer the users negotiate with services to meet deadline and budget constraints, in the second layer services purchase resources to meet the user demand. Service and resource prices are set by iteratively adjusting them up and down based on the measured demand and supply, until a market equilibrium is reached. In simulations they show that this model is more efficient in large Grids than a round-robin approach. Our work is less centralized, and thus more scalable and fail-safe, because all resource providers host their own markets.

G-commerce [41] is a Grid resource allocation system based on the commodity market model where providers decide the selling price after considering long-term profit and past performance. It is argued and shown in simulations that this model achieves better price predictability than auctions. However, the auctions used in the simulations are quite different from the ones we use. The simulated auctions are winner-takes-it-all auctions and not proportional share, leading to reduced fairness. Furthermore, the auctions are only performed locally and separately on all hosts leading to poor efficiency across a set of host. In Tycoon the Best Response algorithm ensures fair and efficient allocations across resources [20]. An interesting concept in G-commerce is that users get periodic budget allocations that may expire, which could be useful for controlling periodic resource allocations (as exemplified by NRAC and SNAC [37]) and to avoid price inflation. The price-setting and allocation model differs from our work in that resources are divided into static slots that are sold with a price based on expected revenue. The preemption and agile reallocation properties inherit in the bid-based proportional share allocation mechanism employed in our system to ensure work conservation and prevent starvation is, however, missing from the G-commerce model.

Buyya et al. [13] implement a completion time minimizing resource allocation algorithm for bag-of-task applications, utilizing an auctioneer infrastructure akin to the one deployed in Tycoon. The difference to the work presented here is that we use fixed budgets and the best response algorithm to place bids, as opposed to allowing bids to vary between a minimum and maximum value to meet deadlines. This allows us to make more precise statements about the fairness and efficiency of our solution in the equilibrium states.

Spawn [40], was one of the first implementations of a computational market, and Tycoon is an incarnation and evolution of many ideas presented in that work. Tycoon, in essence, extends Spawn by providing a Best Response agent for optimal and incentive-compatible bid distribution and host selection, and by virtualizing resources to give more fine-grained control over QoS enforcement. Tycoon also offers a more extensive price prediction infrastructure as presented in this paper. However, the general, continuous-bid and proportional share auction architecture is largely the same.

Other market based resource allocation systems, not focussing on Grid applications, have been presented in [39, 35, 15, 33]

7. Conclusions

We have presented an integrated Grid market of computational resources based on combining a market-based resource allocation system, Tycoon, and a Grid meta-scheduler and job submission framework, Nordugrid ARC.

One of the most challenging integration points was to map the Grid identity to an asserted capability. This prob-

lem was solved by introducing the concept of transfer tokens. This allowed both the private Grid credentials, and the bank account keys to remain local. It also makes it easy for resource users to give out 'gift certificates', to allow users without a Tycoon client installation to submit (and fund) jobs to the Tycoon cluster.

One of the first experiences gained from user feedback of the system was that it was hard to know how much money to use to fund a job. To aid the users in deciding how much funding their jobs would need to complete within a certain deadline, or conversely when a job would be expected to complete given a budget, we developed a suite of prediction models and tools. The accuracy of these predictions depends on the regularity of previous price snapshots and it is therefore crucial, for the results to be good, to pick a time window to study that exhibits these patterns. We therefore also implement a model that allows statistical data within a certain time window to be retrieved, using approximations based on linear smoothing functions.

Finally, our experimental results using a Bioinformatics application developed for the Grid, show that the level of performance delivered when submitting a large batch of jobs, can be customized by the incentive compatible use of transfer tokens. Thus the fairness and economically efficiency properties of Tycoon can be carried over to the Grid Market users.

Future work includes extending the lightweight prediction model presented here to handle arbitrary distributions and studying how higher-level reservation mechanisms, such as Service Level Agreements, Future Markets, Insurance Systems, and Swing Options can be built on top of the prediction infrastructure presented here to provide more user-oriented QoS guarantees.

8. Acknowledgments

We thank Bernardo Huberman, Lars Rasmusson, Fang Wu, Li Zhang, and Kate Keahey for fruitful discussions. The Tycoon Grid Market work would not have been possible without the funding from the HP/Intel Joint Innovation Program (JIP), our JIP liason, Rick McGeer, and our collaborators at Intel, Rob Knauerhase and Jeff Sedayao.

The work on the Bioinformatics application was funded by Wallenberg Consortium North Foundation and Vinnova.

References

[1] Yellow dog Updater, Modified. http://linux.duke.edu/projects/yum/.

[2] SUN, Queen's Univ, First Derivatives to Speed Bank Risk Analysis. *GRID today*, 3(30), July 2004.

[3] The Grid for the Video Industry. *GRIDSTART Business Newsletter*, (2):4, April 2004.

[4] EGEE. Enabling Grids for EsciencE. http://egee-intranet.web.cern.ch/egee-intranet/gateway.html, 2005.

[5] ESG. Earth System Grid. http://www.earthsystemgrid.org, 2005.

[6] NEESit. http://it.nees.org/, 2005.

[7] OSG. Open Science Grid. http://www.opensciencegrid.org, 2005.

[8] TeraGrid. http://www.teragrid.org, 2005.

[9] J. Andrade and J. Odeberg. HapGrid: a resource for haplotype reconstruction and analysis using the computational Grid power in Nordugrid. *HGM2004: New Technologies in Haplotyping and Genotyping*, April 2004.

[10] D. Bosio, J. Casey, A. Frohner, L. Guy, P. Kunszt, E. Laure, S. Lemaitre, L. Lucio, H. Stockinger, K. Stockinger, W. Bell, D. Cameron, G. McCance, P. Millar, J. Hahkala, N. Karlsson, V. Nenonen, M. Silander, O. Mulmo, G.-L. Volpato, G. Andronico, F. DiCarlo, L. Salconi, A. Domenici, R. Carvajal-Schiaffino, and F. Zini. Next-generation eu datagrid data management services. In *Proceedings of Computing in High Energy and Nuclear Physics*, La Jolla, CA, USA, March 2003.

[11] Brent N. Chun and Philip Buonadonna and Chaki Ng. Computational Risk Management for Building Highly Reliable Network Services. In *Proceedings of the 1st Workshop on Hot Topics in System Dependability*, 2005.

[12] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. *Proceedings of the IEEE, Special Issue on Grid Computing*, 93(3):479–484, March 2005.

[13] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal. Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm. *Software: Practice and Experience (SPE) Journal*, 35(5):491–512, April 2005.

[14] G. Caronni, T. Curry, P. S. Pierre, and G. Scott. Supernets and snHubs: A Foundation for Public Utility Computing. Technical Report TR-2004-129, Sun Microsystems, 2004.

[15] A. Chavez, A. Moukas, and P. Maes. Challenger: a multiagent system for distributed resource allocation. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 323–331, New York, NY, USA, 1997. ACM Press.

[16] B. N. Chun, P. Buonadonna, and C. Ng. Computational Risk Management for Building Highly Reliable Network Services. In *Proceedings of the IEEE First Workshop on Hot Topics in System Dependability*, 2005.

[17] L. ChunLin and L. Layuan. A two level market model for resource allocation optimization in computational grid. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 66–71, New York, NY, USA, 2005. ACM Press.

[18] S. Clearwater and B. A. Huberman. Swing Options. In *Proceedings of 11th International Conference on Computing in Economics*, June 2005.

[19] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, 2003.

[20] M. Feldman, K. Lai, and L. Zhang. A Price-Anticipating Resource Allocation Mechanism for Distributed Shared Clusters. In *Proceedings of the ACM Conference on Electronic Commerce*, 2005.

[21] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP'05: Proceedings of International Conference on Network and Parallel Computing*, volume 3799, pages 2–13. LNCS, Springer-Verlag GmbH, 2005.

[22] I. Foster and C. Kessleman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[23] The Future and Emerging Technologies Global Computing Initiative. Technical report, European Commision, DG Information Society, July 2005.

[24] S. Graupner, J. Pruyne, and S. Sherad. Making the Utility Data Center a Power Station for the Enterprise Grid. Technical Report HPL-2003-53, Hewlett-Packard Laboratories, 2003.

[25] L. J. Halliwell. *Mean-Variance Analysis and the Diversification of Risk*. Casualty Actuarial Society, St. Louis, Missouri, USA, May 1995.

[26] J. Hellerstein, K. Katricioglu, and M. Surendra. An Online, Business-Oriented Optimization of Performance and Availability for Utility Computing . Technical Report RC23325, IBM, December 2003.

[27] L. V. Kale, S. Kumar, M. Potnuru, J. DeSouza, and S. Bandhakavi. Faucets: Efficient resource allocation on the computational grid. In *ICPP '04: Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04)*, pages 396–405, Washington, DC, USA, 2004. IEEE Computer Society.

[28] G. Kan. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter Gnutella, pages 94–122. O'Reilly & Associates, Inc., 1st edition, March 2001.

[29] K. Lai. Markets are Dead, Long Live Markets. *SIGecom Exchanges*, 5(4):1–10, July 2005.

[30] K. Lai, B. A. Huberman, and L. Fine. Tycoon: A Distributed Market-based Resource Allocation System. Technical report, arXiv, 2004. http://arxiv.org/abs/cs.DC/0404013.

[31] K. Lai, L. Rasmusson, E. Adar, S. Sorkin, L. Zhang, and B. A. Huberman. Tycoon: an Implemention of a Distributed Market-Based Resource Allocation System. Technical Report arXiv:cs.DC/0412038, HP Labs, Palo Alto, CA, USA, Dec. 2004.

[32] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, December 1998.

[33] T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard. Enterprise: A Market-like Task Scheduler for Distributed Computing Environments. In B. A. Huberman, editor, *The Ecology of Computation*, number 2 in Studies in Computer Science and Artificial Intelligence, pages 177–205. Elsevier Science Publishers B.V., 1988.

[34] H. M. Markowitz. Portfolio Selection. *Journal of Finance*, 7(1), March 1952.

[35] O. Regev and N. Nisan. The Popcorn Market: Online Markets for Computational Resources. In *Proceedings of 1st International Conference on Information and Computation Economies*, pages 148–157, 1998.

[36] T. Sandholm. emediator: a next generation electronic commerce server. In *AGENTS '00: Proceedings of the fourth international conference on Autonomous agents*, pages 341–348, New York, NY, USA, 2000. ACM Press.

[37] T. Sandholm, P. Gardfjell, E. Elmroth, L. Johnsson, and O. Mulmo. An ogsa-based accounting system for allocation enforcement across hpc centers. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 279–288, New York, NY, USA, 2004. ACM Press.

[38] O. Smirnova, P. Erola, T. Ekelöf, M. Ellert, J. Hansen, A. Konsantinov, B. Konya, J. Nielsen, F. Ould-Saada, and A. Wäänänen. The NorduGrid Architecture and Middleware for Scientific Applications. *Lecture Notes in Computer Science*, 267:264–273, 2003.

[39] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):048–063, 1996.

[40] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A Distributed Computational Economy. *Software Engineering*, 18(2):103–117, 1992.

[41] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik. G-commerce: Market formulations controlling resource allocation on the computational grid. In *IPDPS '01: Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, page 10046.2, Washington, DC, USA, 2001. IEEE Computer Society.

[42] F. Wu, L. Zhang, and B. A. Huberman. Truth-telling Reservations. http://arxiv.org/abs/cs/0508028, 2005.

[43] L. Xiao, Y. Zhu, L. M. Ni, and Z. Xu. Gridis: An incentive-based grid scheduling. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, page 65.2, Washington, DC, USA, 2005. IEEE Computer Society.