2007

A Pay-as-Bid Mechanism for Pricing Utility Computing

Philipp Bodenbenner
Institute of Information Systems and Management, Universität Karlsruhe, bodenbenner@iism.uni-karlsruhe.de

Jochen Stößer
Institute of Information Systems and Management, Universität Karlsruhe, stoesser@iism.uni-karlsruhe.de

Dirk Neumann
Institute of Information Systems and Management, Universität Karlsruhe, neumann@iism.uni-karlsruhe.de

20th Bled eConference
eMergence:
Merging and Emerging Technologies, Processes, and Institutions
June 4 - 6, 2007; Bled, Slovenia

A Pay-as-Bid Mechanism for Pricing Utility Computing

Philipp Bodenbenner, Jochen Stößer, Dirk Neumann
Institute of Information Systems and Management (IISM)
Universität Karlsruhe (TH)
Englerstr. 14, 76131 Karlsruhe, Germany
{bodenbenner, stoesser, neumann}@iism.uni-karlsruhe.de

Abstract

Encountering the increasing demand for high-performance computational resources in academic as well as commercial organisations, utility computing offers a solution by providing users with on-demand availability of requested computing services. Approaches to the fundamental issue of resource allocation include the use of technical scheduling mechanisms as well as introducing economic ideas into the allocation schemes. Technical scheduling mechanisms are often very simple (such as first-in-first-out) but suffer under the shortcoming to adequately prioritize jobs in times when demand exceeds supply. As empirical studies show, Grids (such as PlanetLab) are frequently characterized by huge excess demand for resources. This is where economic models such as markets come into play. Hitherto, market mechanisms are either (too) simple or too complex for usage in Grids.

The contribution of this paper is threefold. Firstly, a mechanism for Grids is proposed, which is still simple but geared up for use in the Grid. Secondly the mechanism is embedded in state-of-the-art Grid middleware Sun N1 Grid Engine 6. Thirdly, it is shown by means of a numerical case study that this mechanism is superior to other commonly used mechanisms.

Keywords: Utility Computing, Grid Computing, Market-based Scheduling, Pay-as-Bid, Sun N1 Grid Engine

1 Introduction

The number of complex and elaborate calculations which require increasingly powerful and faster computing resources has been constantly growing. At the same time, resource owners intend to keep their resource inventory at a minimum level to avoid high total cost of ownership. A promising solution to these diverging trends is to pool together distributed, computational resources to large clusters, so called "Grids", that can provide users with sufficient computing resources on-demand (Foster et al. 2001). Grid computing is often denoted as utility computing, as computing resource are utilities like water or electricity and can be accessed dynamically (Rappa 2004). Recently, mainly two utility computing initiatives have made waves in the community: Sun's One-Dollar-Per-CPU-Hour[1] and Amazon's Elastic Compute Cloud[2] are prominent examples for the industry take-up of utility computing offerings.

Concerning their size and build-up, Grid systems can be classified into three types: Cluster Grids are sets of cooperating computer hosts in a cluster that offer a single point of access to users within a project or department. Enterprise or Campus Grids enable the sharing of computer resources for members of different projects or departments within an organisation. Global Grids realize the Enterprise Grid idea across organisational boundaries and therefore allow for the creation of large virtual resource sharing systems (Joseph et al. 2004).

The key problem in distributed resource sharing environments, be it Cluster, Enterprise or Global Grids, is allocation, i.e. how to distribute the scarce resources to requesters at what time. The common approach to this problem is to use technical schedulers (such as first-in-first-out) to determine the allocation. Those technical schedulers are hampered by the fact that they cannot define reasonable priorities in cases when there is excess demand for resources (Stößer et al. 2007).

In recent years, the idea of employing market based mechanisms attracted more and more interest due their ability to base the allocation on the real demand and supply situation (Smidt 1968; Sutherland 1968). As a side effect, sophisticated pricing models are facilitated.

In this paper, we will focus on Enterprise Grids and in particular on Sun N1 Grid Engine (N1GE). The results we achieve here, whether the market mechanism can be used for N1GE as well, can be generalized to other Enterprise Grids. The reason why we refer to N1GE is to assure that our market solution is not just purely theoretical but solidly founded in the real world. N1GE is a distributed resource management and scheduling system from Sun Microsystems which operates on the Enterprise Grid level (N1GE User's Guide, 2005). Being an extension of the Solaris operating system, it administers and dynamically allocates the shared pool of heterogeneous resources such as computing power, memory and licensed software within an organisation. The usage of these resources is managed in a way to best achieve the goals of the organisation, such as productivity, timeliness and level of service. Concerning the IT infrastructure itself, a more efficient utilisation provides the basis for the reduction of total cost of ownership and the increase of return on assets for the organisation's computing facilities. N1GE has been employed for setting up Grids within organisations like companies and universities, each comprising a cluster size of around 500-2,000 CPUs.

N1GE provides means of submitting requests for execution of computationally demanding tasks, so called "jobs", to the users associated with the system. A technical scheduler orchestrates the allocation of jobs to the available shared resources by arranging the jobs in a central queue, using a variation of configurable

[1] http://www.sun.com/service/sungrid/overview.jsp, February 13, 2007
[2] http://www.amazon.com/gp/browse.html?node=201590011, February 13, 2007

policies. These policies represent the prevailing precedence structure among single users, departments and projects within the organisation and thus conduce to the appropriate entitlement in the competition for computational resources. Administrators of N1GE are provided with tools for policy adaptation as well as system monitoring, controlling and reporting.

This paper is novel and unique as it proposes a mechanism for Grids which is fairly simple but specifically designed for usage in the Grid. The proposed mechanism is embedded in state-of-the-art Grid middleware Sun N1 Grid Engine 6. Furthermore, we show in a numerical case study that this mechanism is superior to other commonly used mechanisms.

The remainder of this paper is structured as follows: Section 2 presents a motivational scenario which shows how the market mechanism can be integrated in the current scheduler of N1GE. Section 3 covers related efforts in developing market-based market mechanisms for scheduling in the Grid. Section 4 introduces the base model of allocation algorithms. Section 5 presents an extension to this base model which tailors the mechanism to the Grid. A numerical case study is given in section 6. Section 7 concludes the paper with a brief summary and gives an outlook for future research opportunities.

2 Motivational Scenario

The N1GE scheduler consists of a waiting queue with pending jobs and a technical scheduler that assigns waiting jobs to idle resources (see figure 1). The user submits a job combined with a specification of requirements of the job. There are two different groups of specifications, namely hard and soft state. Hard state requirements are essential for a job to run. If no resource fits the required specifications, the job will be ignored by the scheduler and remains pending. Soft state requirements are tried to be considered, but the job's processing does not depend on them. If no further specification is given by the user, a job is assigned to a random resource which fits the requirements. If the job's requirements cannot be accomplished by the assigned resource, e.g. the memory size does not suffice, the job fails and the user is notified. In the N1GE scenario, a system administrator is usually responsible for submitting the resource specification. End users do not need to care about the resource requirements of their jobs.

After receiving the job requests, the scheduler places the jobs on the waiting list of pending jobs. The position of a job in the waiting list is determined by the job's priority value. This priority value is calculated by the scheduler using a predefined mix of different policies. There is one global waiting list under a centralized administration for all pending jobs in the Grid environment.

An excerpt of different policies is given next (Chaubal 2005):

- *Entitlement policy* (Ticket Policy, Share Based): Fair share (resp. proportional share) with manually (by the administrator) set shares for individual users, user groups, a department or a project.
- *Urgency policy*: Policy with deadline contribution (increase of dispatch priority for jobs which will reach their deadline soon), wait-time contribution (increase of dispatch priority for jobs that have been waiting for a long time) and resource requirement contribution (change of dispatch priority for jobs based on the resources they requested).

- *Custom policy* (POSIX): Standard users can sort their jobs according to their importance by assigning different priority values. These ratings apply only to the user's own jobs.
- *Override policy*: The administrator can manually intervene and modify the dispatch priorities.

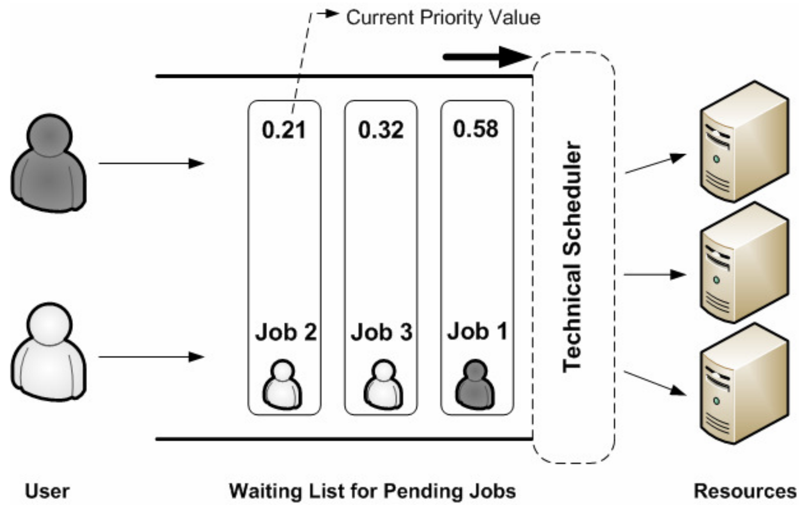

Figure 1: Scheduling in the Sun N1 Grid Engine 6

An example policy mix can look like this

$$P_{mix} = W_e * N_e + W_u * N_u + W_c * N_c$$

where P_{mix} is the dispatch priority, N_e is the normalized entitlement priority (on an interval between 0 and 1) and W_e is the entitlement weighting factor. N_u, W_u, N_c and W_c are defined accordingly for the urgency and custom priorities.

The dispatch priority for each pending job is reevaluated periodically. The time interval for this reevaluation is defined by the system administrator. The job length is not taken into account for the scheduling. Once a job is allocated, it can use the resource until it is finished. This leads to a possibility of manipulation. A user can submit a job with e.g. an endless loop in order to block resources. In this case the administrator can stop the job manually or a limit of maximum processing time can be defined. If a job exceeds this limit it is automatically terminated.

Market mechanism can be attached to the system by incorporating them as a new policy to the priority values. The administrator defines weights for the rules of the market mechanism in the policy mix and can hence influence the impact the new policy has on the dispatch priorities. Introducing a market-based mechanism to the Grid Engine scheduler allows users to directly influence their dispatch priority by submitting bids along with their jobs. To be useful for N1GE, the market mechanism must meet the following requirements (Stößer et al. 2007, cf. Table 1).

Requirement	*Scope*	*Description*
Allocative efficiency	Economic	Allocative efficiency is the overall goal of market mechanisms for Grid resource allocation. A mechanism is allocative efficient, if it maximizes the utility across all participating users (welfare or overall "happiness").

Budget-balance	Economic	A mechanism is budget-balanced if it does not need to be subsidized by outside payments.
Computational tractability	Functional	The market mechanism needs to be computed in polynomial run time in the size of the number of resource requests and offers.
Online Mechanism	Functional	The allocation of the mechanism needs to be made instantaneously, as the market assumes the role of operating system schedulers.
Simplicity	Functional	The mechanism needs to be simple such that the participants understand the mechanism and the bidding strategy.

Table 1: Requirements for the market mechanism

3 Related Work

There are two other basic groups of allocation schemes dealt with in Grid-related literature which aim at distributing fractions of a resource among requesters: fair share and proportional share mechanisms.

- **Fair share mechanism**
  Fair share mechanisms belong to the group of technical schedulers. One example is the SHARE scheduler proposed by Kay and Lauder (1988). Opposed to other technical schedulers, the idea behind fair share is to be fair to users rather than to processes. It is a scheduling strategy in which the usage of a certain resource (mainly CPU time) is equally distributed among system users (as opposed to equal distribution among processes). In a group of n users, everyone will receive a fraction of $1/n$ of the available resource. Different fair share implementations allow the administrator to partition users into groups and apply fair share to these groups as well. The most common way of implementing the fair share scheduling strategy is to recursively apply the round-robin scheduling strategy. The drawback of the fair share strategy is that all parameters are pre-specified and set by the system administrator. The users have no influence on the allocation. From an economic perspective, the fair share strategy reaches only a very low level of allocative efficiency; except in the case that all users have the same utility for a share of the resource. On the downside, fair share is a simple, online, budget-balanced mechanism.

- **Proportional share**
  To alleviate the problems encountered with fair share and to increase efficiency, proportional share mechanisms were introduced. Proportional share initially allows for resource distribution with shares of unequal size for different users accounting for varying importance among them. Whereas scheduling according to pre-set, fixed shares for different users remains technical, market-based proportional share mechanisms dynamically base the resource share on the users' reported valuations, their "bids". The total amount of available resources is distributed among the requesters according to the fraction their reported valuation amounts to in the sum of reported valuation across all resource requesters: a user i with

reported valuation v_i will receive a fraction of $v_i \big/ \sum_{j=1}^{n} v_j$ of the available resource when a group of n users is competing for resource access. Systems using proportional share as allocation scheme were proposed by Chun and Culler (2000), Stoica, Abdel-Wahab et al. (1996) and Lai et al. (2004). Chun and Culler (2000) employ the idea of proportional share on a market where multiple requesters compete for computing time. The system aims at optimizing for user value and utilizes a scheduler which assures that each requester receives a share of the resource which corresponds to the fraction of its reported valuation for the resource. The problem with market-based proportional share is that it does generally not support Quality of Service assertions. The mechanism holds the risk that requesters are not able to obtain the necessary resources (Lai et al. 2004); the share of resources provided to a requester, i.e. the actual service level returned to the requester, will generally be below or above this requester's required service level. More importantly, the basic form of market-based proportional share as suggested by Chun and Culler (2000) does not support advance reservation and may result in high latency which is fatal for interactive applications (Lai et al. 2004). For their Tycoon system, Lai et al. (2004) tackle latency problems and incorporate advanced reservation for resources and an incentivising payment scheme in the basic market-based proportional share mechanism. In summary, proportional share mechanisms satisfy all requirements except the most important one, allocative efficiency.

In the following we will explore an alternative mechanism – a so-called pay-as-bid mechanism (Sanghavi and Hajek 2004) – that may improve on these present mechanisms.

4 Basic Model

Let τ be an allocation mechanism that splits up one unit of a perfectly divisible good among n users. The vector $w = [w_1, ..., w_n]$ comprehends the non-negative bids of the users. These bids equal the payments the users have to make for the share of the resource that is assigned to them. This share is referred to by $x = [x_1, ..., x_n]$ and is calculated according to the pre-specified τ. Thus $x_i = \tau_i(w)$ is the quantity user i is allocated as a result, given a payment vector w.

The allocation mechanism τ is considered *valid* if it satisfies the following four properties (Sanghavi and Hajek 2004):

1. It is an allocation: $\tau_i \geq 0$ and $\sum_i \tau_i(w) = 1$ for all values of w such that $\sum_i w_i > 0$; a zero bid will always get zero allocation.
2. It is smooth: $\tau_i(w_i, w_{-i})$ is differentiable, increasing and concave in w_i for all w_{-i}.
3. It is symmetric in the user indices, such that $\tau_i(w) = \tau_{\sigma(i)}(\sigma(w))$ for all permutations σ of the indices $i = 1, ..., n$.
4. It is scale free, such that $\tau_i(\gamma w) = \tau_i(w)$ for all real $\gamma > 0$ and $0 \leq i \leq n$.

Given this problem formulation, Sanghavi and Hajek (2004) propose an allocation mechanism τ which is shown to be optimal for two users. The two users, referred to as l ("low bidding") and h ("high bidding"), have a payment vector (w_l, w_h) with $w_l \leq w_h$. The optimal mechanism τ^*, guaranteeing the best worst case fractional efficiency[3] and can be devised as follows:

$$\tau_l^*(w_l, w_h) = \frac{w_l}{2w_h} \qquad \text{and} \qquad \tau_h^*(w_l, w_h) = 1 - \frac{w_l}{2w_h}$$

It can be shown that a Nash Equilibrium exists for any scenario with valid user utility functions and a valid allocation mechanism τ, i.e. it satisfies the properties given in chapter 2. A utility function is considered valid if it is differentiable, concave and strictly increasing. The Nash equilibrium point can be regarded as the result of a hypothetical repeated game where users give "myopic best responses": bids are continuously adjusted based on the market information generated by previous iterations and finally converge to the values which constitute the equilibrium (Sanghavi and Hajek 2004).

Compared to any other valid mechanism, the basic model generates at least an equally high social welfare in its Nash Equilibrium point. Furthermore, the uniqueness of this Nash Equilibrium can be guaranteed. Hence, the mechanism is not only optimal for the worst case, but even for any pair of valid value functions. For two buyers, the worst case fractional efficiency of τ^* adds up to 87 %. The mechanism ends up in this worst case scenario when both buyers have linear value functions[4].

This difference in the levels of efficiency is caused by the different pricing schemes. While proportional share uses homogeneous pricing, Sanghavi and Hajek (2004) introduce a discriminatory pricing mechanism. The buyer with a lower bid pays a higher price per share than the high bidder. This volume discount encourages high bidders to bid higher, and thus closer to their true valuation, compared to a scenario with a uniform pricing scheme where users can potentially benefit from shading their bids downwards.

In general, the pricing is given by $p_i(w) = \frac{amount\_paid}{quantity\_received}$. This equals the price the user would be paying for the entire unit given his bid. Thus, for the Sanghavi/Hajek mechanism, in the two buyer case the following pricing scheme is applied:

$$p_l(w) = 2w_h \qquad \text{and} \qquad p_h(w) = \frac{2w_h^2}{2w_h - w_l} \qquad \text{with } p_l \geq p_h.$$

Sanghavi and Hajek (2004) developed an extension of the above mechanism from two to n buyers. This mechanism still has the property of a "volume discount", i.e. higher bidders pay lower prices.

[3] The fractional efficiency is defined as $\frac{\sum_i U_i(x_i)}{\sum_i U_i(x_i^*)}$, with $\sum_i U_i(x_i^*) \geq \sum_i U_i(x_i) \;\; \forall x$

[4] In addition, the slope of the value function of one buyer has to be half the slope of the other buyer. As a comparison the worst case efficiency of the proportional share mechanism is 82.84% (for a proof see Johari and Tsitsiklis 2004).

For n buyers and a given payment vector $w = (w_1, ..., w_n)$, the following mechanism is proposed:

$$\tau_i^* = \frac{w_i}{w_{\max}} \int_0^1 \prod_{j \neq i} (1 - s \frac{w_j}{w_{\max}}) ds$$

with at least two $w_i \geq 0$ and $w_{\max}$ being the maximum bid.

Since it satisfies the four previously given requirements, the mechanism τ^* is valid. The given mechanism simplifies to the optimal mechanism proposed for two buyers when n is set to 2.

In contrast to the case for two buyers, it is hard to determine an exact value for the worst case efficiency for an unlimited number of buyers. Instead, Sanghavi and Hajek calculate an interval as an approximation for the worst case efficiency:

$$0.8703 \leq \frac{\sum_i U_i(\tau^*(\tilde{w}_i))}{\sum_i U_i(x_i^*)} \leq 0.875$$

Obviously, the proposed mechanism is still close to the theoretical maximum worst case efficiency, i.e. 87.5%. But a guarantee that the mechanism τ^* is the optimal one can no longer be given.

5 Extended Model

To adapt the basic model to the domain of scheduling in a Grid environment, with a particular focus on the N1GE scenario, further considerations have to be done. The following section presents a number of extensions to the basic model which are required to make the scheduling mechanism applicable for large-scale resource clusters retaining the economic features:

- **Re-evaluation of the Priority Value**
  Recalculation of the priority values of all pending jobs is necessary whenever a new job enters the waiting list. This is required since every new bid would change the allocated share of each waiting job. The re-evaluation of all jobs would put quite some load on the scheduler if a large number of jobs enter the waiting queue within a short time interval. This problem can be prevented if the recalculation is not done for every new job, but according to a pre-specified time window. All jobs arriving at the waiting queue during such a window are gathered and the recalculation is done for all of the jobs at once. A problem that could arise when time windows are employed is that jobs arriving at the beginning of such a window are delayed before being put on the waiting queue. This is especially undesirable for time critical and urgent jobs. Therefore the time window has to be specified accordingly (e.g. re-evaluation is done every 2 seconds) to minimize the delay of the jobs.

- **Feedback**
  For a convergence to the Nash equilibrium point, the users need feedback on their bids to see how much share of the good they received. Using this partial market information, they will adjust their bids in myopic best responses to finally reach the Nash equilibrium point. The N1GE scheduler does not support direct feedback yet. The reporting tool for the waiting list, named *qstat* in N1GE, shows the priority value for each job, but this is the accumulated value for all policies that are part of the policy mix in the

specific scenario. To ensure a convergence to the Nash equilibrium it might be necessary to extend the tool in such a way that it reports the fraction of the priority value that was allocated to each job according to the submitted bid. The "won" priority value should only be reported to the user that submitted the respective job. Otherwise it might exhibit the possibility for a user to draw conclusions on the other users' valuations and consequently influence his future bids in such a way that the Nash equilibrium is no longer reached.

- **Bid Updating**
  Related to the issues of re-evaluation of the priority value and feedback is bid updating: Shall the mechanism allow bidders to update their bids after submitting the jobs to the waiting queue? Considering a periodic re-evaluation of the priority value, this would not increase the computational effort since priorities are updated anyway. It still needs to be analysed if bid updating has any impacts on the economic properties of the mechanism. But it probably would not change these properties since bid updating can be interpreted as part of the "myopic best response" bidding which leads to the Nash equilibrium. It would allow bidders to directly react on the positioning of their jobs in the queue.
- **Job Starvation**
  The current Sanghavi/Hajek mechanism entails the problem of job starvation. If a user submits a job with a very low bid the job will be displaced by all following jobs with a higher bid. Thus this job will probably never proceed to be executed. To diminish this setback a waiting time bonus could be introduced. This instrument is already incorporated in the N1GE urgency policy and allows increasing the priority value of a job according to the time it has already waited. Another possibility would be a deadline for pending jobs. After a certain specified time of waiting in the queue the job will be "killed" and taken out of the queue. The user who submitted the job will be informed. This solution can be combined with the possibility for users to update their bids. Should the previous bids not suffice for the job to advance in the queue, the user can increase the bid value. If this is not done, the job will eventually be dismissed by the system.
- **Job Length**
  The current disregard of the job length when calculating the priority values is another topic to be discussed. Currently the job length is not taken into account at all for determination of the job's order in the queue. Consequently, the mechanism is not "merge-proof" (Moulin 2004): It might be unattractive for users to submit short jobs with high urgency. These kinds of jobs are very expensive compared to longer jobs. Hence users might tend to merge smaller jobs to one big job to pay the "entrance fee" to the waiting list only once. Submitting a merged job with a bid that equals the sum of the single bids of the merged bids could lead to a much better position in the waiting queue compared to submission of the single jobs. It might thus be useful to base the allocation on the lengths of jobs, which however creates the need for determining a job's length beforehand. Though there are tools for roughly estimating the duration of a specific job, incorporating them will doubtless have a negative effect on the ease of use of the mechanism.

- **Payment**
  It has to be specified at what time of the allocation process the user has to make his payment. It might happen that the job is rejected because none of the machines fits the job's required specifications or that it fails during execution. If the user has to pay directly after submitting his job to the waiting list, he has to be refunded in these cases where the execution of the job fails. A payment that is made in the first stage of allocation is also problematic when bid updating is allowed. For every modified bid that he submits he has to adjust his payment. A solution would be to account only for jobs which passed the execution phase successfully.

6 Case Study

In the following case study, fair share, proportional share and the extended model are compared concerning their allocation and the resulting efficiencies. They are assumed as being incorporated as policies in the N1GE scheduler and the only policies to determine the dispatch priority; all other available policies are weighted with 0.

Suppose two users are competing for resources in this N1GE environment. Currently all resources are blocked with running jobs. The waiting list for pending jobs is empty. Now the two users submit their jobs J_i, with $i = 1,2$, along with the resource specifications required for the job's execution. For the proportional share and the Sanghavi/Hajek pay-as-bid mechanism each user additionally submits his bid w_i.

6.1. Allocation and Pricing

The scheduler computes the priority values according to the pre-specified scheduling mechanism, here: fair share, proportional share and extended mechanism. The share for each user is denoted by x_i. The users' quasi-linear utility functions are given by $U_1(x) = 1.5x_1 - p_1x_1$ and $U_2(x) = 4x_2 - p_2x_2$ where p_i is user i^{th} *unit price*, i.e. the price user *i* would have to pay if he got the whole resource unit. Furthermore, assume the centralized resource provider to have a quasi-linear utility function $U_P(x) = \sum_i p_i x_i$ with no reservation prices. This is a quite realistic assumption in Enterprise and Campus Grid settings. Consequently, social welfare can be computed as $U(x) = U_1(x) + U_2(x) + U_P(x) = 1.5x_i + 4x_i$.

Fair Share

Applying the fair share scheduling strategy, in which each user receives an equal share and no payments arise from this allocation, the priority value for all users can be calculated as $x_i = \frac{1}{n}$. In the current example with two users, x_1 and x_2 are consequently set to 0.5 and $p_1 = p_2 = 0$. This allocation results in the valuations $U_1(x) = 0.75$, $U_2(x) = 2$ and $U_P(x) = 0$. Summing up the individual valuations, the fair share mechanism creates welfare of $U(x) = 2.75$, which corresponds to an efficiency ratio of 68.75% compared to the optimal allocation; In this optimal allocation, the high bidding user is given a priority value of 1.0 whereas the low bidder receives nothing. This allocation would create the maximum social welfare of 4.

Proportional Share

The example applies a dynamic type of proportional share mechanism in which the users submit bids to the provider. After receiving the bids, the provider distributes fractions of the priority value according to the pre-set allocation rule $\tilde{\tau}_i\ (w) = \frac{w_i}{\sum_i w_i}$ to the users.

Concerning the pricing, it is assumed that a pay-as-bid rule is applied: Each user is required to finally pay a total sum which equals his submitted bid. For the pay-as-bid-rule, the unit price p_i for user i can consequently be computed as $w_i = p_i x_i \Leftrightarrow p_i = \frac{w_i}{x_i}$. With proportional share, however, $x_i = \frac{w_i}{\sum_j w_j}$ and thus $p_i = \sum_j w_j$, that is user 1 and user 2 have to pay the same unit price.

For two users, the given utility and pricing functions, the mechanism arrives at the Nash equilibrium[5] bid vector $w^* = (w_1^*, w_2^*)$=(0.2975, 0.7934). In this Nash equilibrium, $x_1^* = \tilde{\tau}_1\ (w^*) = \frac{3}{11}$ is allocated to user 1 and $x_2^* = \tilde{\tau}_2\ (w^*) = \frac{8}{11}$ to user 2 and no user i has an incentive to unilaterally deviate from its bid w_i^*. The unit prices are $p_1^* = p_2^* = 1.1$.

Thus the proportional share mechanism generates an overall social welfare of $U(x) = 1.5 \cdot \frac{3}{11} + 4 \cdot \frac{8}{11} = 3.32$ and an efficiency ratio of 82.95%.

Extended Model

According to the optimal allocation mechanism τ^* for the two users given above, the Nash equilibrium point is reached for the bid vector (w_1^*, w_2^*) = (0.28125, 0.75) and the following shares are allocated to the users:

User 1: $$x_1^* = \tau_1(w^*) = \frac{w_1^*}{2w_2^*} = 0.1875$$

User 2: $$x_2^* = \tau_2(w^*) = 1 - \frac{w_1^*}{2w_2^*} = 0.8125$$

In the Nash equilibrium, the extended model results in an overall social welfare of $U(x) = 1.5 \cdot 0.1875 + 4 \cdot 0.8125 = 3.53125$. Hence, the allocation according to the extended model reaches an efficiency ratio of 88.28% for this setting, which is significantly higher than the results of the fair share and proportional share allocations.

Applying the pay-as-bid-rule to this allocation exemplifies the “volume discount” for the high bidding user 2. He pays a unit price of $p_2^* = \frac{w_2}{x_2^*} = 0.9231$ whereas user 1 has to pay a notably higher unit price of $p_1^* = \frac{w_1}{x_1^*} = 1.5$.

[5] For a derivation and proof of uniqueness of the Nash equilibrium for the proportional share mechanism see Maheswaran and Basar (2005).

	Fair Share	Proportional Share	Extended Model
Bid vector	---	$(w_1^*, w_2^*) = (0.30, 0.79)$	$(w_1^*, w_2^*) = (0.28, 0.75)$
Allocation	$x_1 = x_2 = 0.5$	$x_1^* = 0.27$ $x_2^* = 0.73$	$x_1^* = 0.19$ $x_2^* = 0.81$
Unit prices	---	$p_1^* = p_2^* = 1.09$	$p_1^* = 1.5$ $p_2^* = 0.92$
Utilities	$U_1(x) = 0.75$ $U_2(x) = 2$ $U_P(x) = 0$	$U_1(x^*) = 0.11$ $U_2(x^*) = 2.12$ $U_P(x^*) = 1.09$	$U_1(x^*) = 0$ $U_2(x^*) = 2.5$ $U_P(x^*) = 1.03$
Social Welfare	$U(x) = 2.75$	$U(x^*) = 3.32$	$U(x^*) = 3.53$
Efficiency ratio	68.75%	82.95%	88.28%

Table 2: Comparison of the Scheduling Mechanisms

7 Conclusion

The extended model being inspired by Sanghavi/Hajek pay-as-bid mechanism is a promising addition for the N1GE scheduler. Employing a market-based mechanism for resource allocation in Grids offers new possibilities on both sides, for providers as well as for buyers. Current technical schedulers require an administrator to specify user weights based on these users' relative importance, regardless of the dynamic demand and supply situation, opening up possible inefficiencies. To this end, the Sanghavi/Hajek pay-as-bid mechanism allows flexible reactions to changes in the demand and supply situation. Moreover, it offers an elaborated pricing scheme where prices reflect the current market situation and induce users to report their true valuations to the system. The administrator no longer needs to adjust the weights manually and the users can directly express urgency of their jobs by submitting a high bid without being dependent on the administrator as a "mediator". Furthermore, this usage-based pricing scheme opens up new avenues for both external (Inter-enterprise and Utility Computing) Grids and internal (Enterprise and Campus) Grids.

The implementation of the mechanism within N1GE can be done fairly easy as a new scheduling policy, which then would be part of the policy mix. This integration can be done without major changes in the existing architecture. As mentioned, an extension of the current waiting list reporting tool might be necessary to enable a more sophisticated feedback functionality for the users. Very important for a successful integration of a market-based scheduler will be the enforcement of the payment. Thus some kind of payment system, either based on real money or virtual credits, has to be integrated in the N1GE architecture which is conjunct with a high implementation effort.

Comparing the extended model to other market-based mechanisms, it scores with its ease-of-use and an allocation mechanism that is transparent to the user. In addition, it has a noticeably increased worst case fractional efficiency in comparison to the proportional share mechanisms and is close to the theoretical maximum regarding the worst case efficiency of pay-as-bid mechanisms. Above all, the extended model imposes a very low additional communicational and computational effort on the scheduling process. It allows real-time allocations, even if the recalculation of the priority has to be done periodically.

Further work has to be done on analyzing the extensions and their impact on the economic features of the mechanism. For this purpose it would be very helpful to actually implement the mechanism within the N1GE scheduler. This would allow for running simulations to examine the mechanism's behaviour and performance, especially regarding execution time, in large scale clusters.

In addition, the focus is on other scenarios where the mechanism can be employed along with the N1GE. A very interesting and challenging approach is to establish a decentralized version of the mechanism to support decentralized waiting queues as well. This might be necessary to keep the N1GE scheduler applicable for very large clusters (20,000+ cores), which will be demanded in the near future.

A second scenario would change the current allocation per job to a reservation of timeslots of whole machines (resp. CPUs). This would enable users to book a machine for a certain time span and use it to run as many jobs as possible. The problem with this scenario is a restriction of N1GE, which allows only one task per CPU at a time. A possible solution to this would be virtualization, where virtual machines are utilized on a layer between scheduler and resources to make single CPUs "divisible".

References

Chaubal, C., (2005): Scheduler Policies for Job Prioritization in the Sun N1 Grid Engine 6 System. Sun BluePrints Online, Sun Microsystems, Inc., Santa Clara, CA, USA, http://www.sun.com/blueprints/1005/819-4325.pdf.

Chun, B. N. and D.E. Culler (2000): Market-based Proportional Resource Sharing for Clusters. Technical Report, Berkeley, CA, USA.

Foster, I., C. Kesselman and S. Tuecke (2001): The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing, Vol. 15, No. 3, pp. 200-222.

Johari, R. and J. Tsitsiklis (2004): Efficiency Loss in a Network Resource Allocation Game. Mathematics of Operations Research, Vol. 29, No. 3, pp. 407–435.

Kay, J. and P. Lauder (1988): A Fair Share Scheduler. Communications of the ACM, Vol. 31, No. 1, pp. 44-56.

Kelly, F., (1997): Charging and Rate Control for Elastic Traffic. European Transactions on Telecommunications, Vol. 8, No. 1, pp. 33-37.

Lai, K., B.A. Huberman and L. Fine (2004): Tycoon: A Distributed Market-based Resource Allocation System. Technical Report, HP labs, Palo Alto, CA, USA.

Maheswaran, R. and T. Basar (2005): On Revenue Generation When Auctioning Network Resources. Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference.

Moulin, H. (2004): Proportional Scheduling, Split-proofness, and Merge-proofness. Games and Economic Behavior, forthcoming.

Rappa, M.A. (2004): The Utility Business Model and the Future of Computing Services. IBM Systems Journal, Vol. 43, No. 1, pp. 32-42.

Sanghavi, S. and B. Hajek (2004): Optimal Allocation of a Divisible Good to Strategic Buyers. Proceedings of the 43rd IEEE Conference on Decision and Control.

Smidt, S. (1968): Flexible Pricing of Computer Services. Management Science, Vol. 14, No. 10.

Stößer, J., D. Neumann and A. Anandasivam (2007): A Truthful Heuristic for Efficient Scheduling in Network-Centric Grid OS. Proceedings of the 15th European Conference on Information Systems (ECIS), June 7-9, 2007, St. Gallen, Switzerland, forthcoming.

Stoica, I., H. Abdel-Wahab, et al. (1996): A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. IEEE Real-Time Systems Symposium.

Sun N1GE 6 User's Guide (2005), Sun Microsystems, Inc., Santa Clara, CA, U-SA, http://docs.sun.com/app/docs/doc/817-6117/.

Sutherland, I. E. (1968): A Futures Market in Computer Time. Communications of the ACM, Vol. 11, No. 6, pp. 449-451.