A commodity market algorithm for pricing substitutable Grid resources

Gunther Stuer*, Kurt Vanmechelen, Jan Broeckhove

Department of Mathematics and Computer Sciences, University of Antwerp, Middelheimlaan 1, Antwerp, Belgium

Received 5 September 2006; received in revised form 19 November 2006; accepted 21 November 2006
Available online 11 January 2007

Abstract

A crucial goal for future Grid systems is to strive towards user-centric service provisioning. A way to achieve this is through the use of economics-based resource management. Currently, several models exist from among which auction- and commodity-based models are the most popular. This contribution will focus on the latter, and in particular on commodity markets, where the value of a Grid resource is determined by supply and demand. We propose some refinements to the application of Smale’s method for finding price equilibria in such a Grid market. We also extend the approach to substitutable goods. That is, we introduce ‘slow’ and ‘fast’ CPUs, two categories of the same type of good that are priced separately, but are strongly coupled with potentially strong shifts in demand. We show that Smale’s method can be adapted to handle this type of Grid resources market, and that price stability, allocative efficiency, and fairness are realized.

Keywords: Grid; Dynamic pricing; Grid economics

1. Introduction

In recent years, the need for computational resources has grown significantly. This has led to a situation where local administrative domains can no longer provide enough resources. Due to this, a vision of distributed computing has been articulated in which applications can plug into a vast, worldwide, pool of available resources which are no longer bound by administrative or geographical boundaries. The most important realization of this vision is the computational Grid [1].

An important aspect of Grid middleware is resource management. Traditionally, the emphasis has been on system-centric resource management, where a scheduling component decides which jobs are to be executed at which resource, based on cost functions driven by system oriented metrics such as utilization and system throughput. If one wants to muster user support for Grids, it is important that this emphasis shifts to a more user-centric approach, where the focus is on delivering maximum utility to the individual user of the Grid. In this way, allocation decisions are steered by the user’s valuation of their results.

---

* Corresponding author. Tel.: +32 3 218 39 06.
E-mail address: gunther.stuer@ua.ac.be (G. Stuer).
Likewise, in calm periods, prices should drop. Such a price-based model is fair to both providers and consumers.

In addition to a pricing scheme, agreements have to be made on the type and granularity of the goods that are traded in the market. Furthermore, an agreement has to be made on the currency used. Indeed, according to [5], virtual economies are worthless unless the currency used can somehow be converted to real world money. While this is certainly our long term goal, we have currently chosen not to make this link, as it would introduce its own set of problems such as exchange rates, security, management systems, and so on.

It is clear that framing the resource allocation problem in economic terms is attractive for several reasons [6]: resource usage is not free, the dynamics of Grid performance response are hard to model, and it provides an incentive for resource owners to bring their resources to the Grid. For the above reasons, resource management based on economic principles will play an important part in the next generation of Grid-related challenges, which will be dominated by economical aspects instead of technical ones [7].

A vision of the Grid where applications can treat computational, network, and storage resources as interchangeable, and not as specific machines, networks, and disk systems, corresponds closely to a commodity market. In such a market, a number of resource categories are defined for which the market entity suggests a price. Consumers and providers then respond to these price levels by creating demand and supply for the different categories. The market optimizes the price to achieve market equilibrium, in which demand equals supply. When the market is brought to equilibrium, we achieve a Pareto optimal (or Pareto efficient) allocation. This means that a change in price cannot increase the utility of one participant in the market without worsening the utility of at least one of the other market participants.

An attractive feature of commodity markets is that the price is optimized for achieving a Pareto optimal allocation through a simple interaction between the market and its participants. Participants only have to respond with their demand and supply values given a single price vector for all of the market’s commodities. In non-commodity market organizations, wherein specific resources are sold in separate markets (e.g. single unit auctions that sell access rights on a specific CPU of a provider), participants need to deal with the extra complexity of deciding in which market to participate. Consumers who decide on participating in multiple markets at the same time need to decide on how they divide their available budget across these markets. If these decisions are taken in a non-optimal way, this can lead to inefficiencies. An example of such an inefficiency is that of a consumer who needs a minimum value in utility data centers [18], and more recently to Grid computing [19–28]. An overview can be found in [2].

According to the classification in [29], our work uses a state-based, non-pre-emptive strategy. State-based means that the allocations are based on a current snapshot of the system state as opposed to model-based, where the allocations are based on a predictive model. Non-pre-emptive means that tasks are assigned to hosts once and stay there, as opposed to pre-emptive strategies where tasks are allowed to migrate between hosts.

In a commodity market, participants are encouraged to report their true valuations (in terms of demand and supply levels) because they have no information on the demand levels of the other participants, and do not know how close the suggested price is to the equilibrium. This limits the options for strategic behavior of the participants in such markets and the associated complexity of strategic participation.

Nevertheless, there are some problems with the commodity market approach. First, the notion of locality is not taken into consideration. In a commodity market, all resources that fall under the same commodity category are deemed equal, and the added cost for network usage is not incorporated in the price. This issue is still an open research question.

Secondly, as soon as one has to start paying for resource usage, one can no longer deem all resources equal. Besides performance, aspects such as quality of service, available (cache) memory, network bandwidth, . . . contribute to the value of a compute node. At first sight, this would be an argument against the use of commodity markets in Grids, but a technique known as fuzzy clustering [9] offers a solution. Here one tries to group all the compute nodes in x categories based on a, fuzzy, combination of their characteristic properties. As a result, one can come up with n clusters of compute nodes, e.g. low-end, average, and high-end systems. Once this is done, the price can be determined using commodity market principles. This time however, one has three substitutable commodities, which makes decision-making more challenging. Is it preferable to buy A nodes of type X, or B nodes of type Y, or maybe some other combination? This illustrates clearly that pricing of substitutable goods, using commodity market principles, is an important step in the construction of Grid commodity markets.

2. Related work and classification

Cost-based resource management and market-oriented models have proven to be useful for resource management in the context of cluster computing [10–12]. Furthermore, economic models for resource sharing have been applied to agent systems [13,14], telecommunication networks [15], databases [16] and data mining [17], optimization of business value in utility data centers [18], and more recently to Grid computing [19–28]. An overview can be found in [2].

According to the classification in [29], our work uses a state-based, non-pre-emptive strategy. State-based means that the allocations are based on a current snapshot of the system state as opposed to model-based, where the allocations are based on a predictive model. Non-pre-emptive means that tasks are assigned to hosts once and stay there, as opposed to pre-emptive strategies where tasks are allowed to migrate between hosts.

When one classifies based on the underlying economic model [30], our work belongs to the category of commodity markets. Some other possible models are bargaining [31], posted price models [20], contract based models [32,33], bid-based proportional resource sharing models [34,35], bartering [36], and various forms of auctions [37–39]. An overview can be found in [40].
The use of a commodity market model for obtaining global equilibrium prices for resources in a Grid context has already been proposed in [3,6]. We take this approach further and extend it to allow for trading and pricing of substitutable goods. This more closely models Grid markets, where multiple resources are available to process the jobs, but with varying performance characteristics. We also introduce significant improvements to the optimization algorithms used to compute the equilibrium prices.

After outlining our modeling decisions in Section 3, we describe our pricing algorithm in Section 4 and simulation process in Section 5. Next, the algorithm is evaluated in Section 6. We conclude with a study of our market’s behavior as a function of price correctness and stability, fairness, and allocative efficiency.

3. Commodity market model

Our market model consists of three parties: consumers, providers, and the market itself. Consumers buy resources according to a buying strategy, providers sell resources following a selling strategy, and the market establishes an optimal price, matching demand and supply, using a pricing strategy. Consumer and provider strategies, which are loosely based on [6], are described in later subsections. The pricing strategy is detailed in Section 4. While providers and consumers do not necessarily have to be different entities, we have chosen to treat them as being distinct for clarity.

3.1. Resource and job model

In this work, we limit ourselves to modeling a commodity market of two substitutable goods: low-end and high-end compute nodes (fastCPU and slowCPU), which we will refer to as the market in the following. As all results presented in this contribution are based on simulations and not on a real-world Grid setup, the problem of clustering nodes into a finite set of resource categories was not an issue. However, once the proposed Grid economics engine is integrated into real grids, fuzzy clustering of available resources will be an important issue, for which further research is needed.

In case of substitutable goods, one needs a measure by which goods in the substitutable categories can be compared. For compute nodes, we express this as a speedup factor \( \text{PerfRatio}_i \), which denotes the performance ratio between a CPU from category \( i \) and one from category \( r \), the reference category. In this way, two substitutable good categories are introduced into the market model. As a consequence, consumers will be faced with the problem of deciding which of the two categories to buy for a particular job. This differs from the work presented in [3,6], wherein a commodity market model is set up with two commodity goods in the form of CPUs and disk storage.

In accordance with the limitation of our resource model to CPUs, we model the jobs as CPU-bound computational tasks. Every job has a nominal running time \( T \), determined by the time it takes for the job to finish on a reference CPU \( r \). When jobs are allocated to a CPU in category \( i \), their actual running time is determined by \( \frac{T}{\text{PerfRatio}_i} \). Jobs are also taken to be atomic, in the sense that they are always allocated to a single CPU.

3.2. Consumer model

Consumers formulate demand in the market by expressing their willingness to buy CPUs from providers in order to run their jobs. Because providers only bring free CPU resources to the market, jobs start running immediately when allocated to a provider’s CPU resource. In order to buy these resources, consumers need credits.

Each consumer is endowed with a limited budget, which is periodically replenished according to a budget refresh period, known as the AllowancePeriod. At the start of a new allowance period, the leftover budget from the previous period is discarded.

Our consumer model assumes that the determination of the consumers’ budget levels is performed by some external process. This process could be governed by a controlling authority that implements the sharing agreements supported by the different Grid stakeholders. Another possibility is that consumers inject real money into the system to obtain a budget for running Grid jobs.

To prevent a scenario where every consumer tries to schedule all of his jobs in the beginning, resulting in an extreme increase in price, expenditures are spread out evenly across the allowance period. Furthermore, we do not assume that consumers know the running times of their jobs. Therefore, we need to prevent consumers from agreeing on a price level that would not be sustainable for them over the entire allowance period.

Each consumer has a queue of jobs it wishes to execute. At certain time instances, called peaks, a large number of jobs will be added to this queue. At other times, there is a certain probability that a new job will be inserted. In every simulation step, consumers are charged the usage rate prices for all Grid resources that are currently in force for their jobs.

Given a price vector \( \mathbf{p} \), consumers have to express their demand for each resource type [41,42]. First, each consumer calculates the average rate, \( ar \), at which it would have spent credits for the jobs it has run so far, if it had been charged the current price. It then computes the capable rate, \( cr \), i.e. the rate of spending it can sustain until the next budget refresh. If the capable rate is greater than the average rate, a consumer will express demand. The demand volume depends on the available budget at the time of demand formulation, and is given by \( cr - cp \), where \( cp \) denotes the amount of credits needed for currently processing jobs.

For each CPU category \( i \), every consumer determines, depending on the job mix it has to schedule, a preference factor \( \text{Pref}_i \). Consumers show preference for a particular CPU category \( i \) according to formula (1). The lower the consumer’s \( \text{Pref}_i \) value, the higher the consumer values a CPU of category \( i \).

\[
\text{Pref}_i = \frac{P_i}{(\text{PerfRatio}_i \times \text{PrefFactor}_i)}
\]  
(1)
with $P_i$ the price for CPU type $i$, $\text{PrefRatio}$, the performance ratio of type $i$ in relation to the reference CPU type $r$ and $\text{PrefFactor}$, the personal preference factor a consumer assigns to type $i$.

For each resource category, from low to high $\text{Pref}_i$, the expressed demand will be the maximum amount possible given the currently remaining budget. In cases where $\text{Pref}_i$ equals $\text{Pref}_j$, a random ordering is assigned to both CPU types.

The preference factor is a simple abstraction for the complex logic a consumer might follow to prefer one CPU category over another outside of pure cost considerations. An example of such a logic, wherein a consumer is willing to pay more than double the price for a CPU of category 1 (which is only twice as fast as one of category 2), is the following: suppose the consumer has a job graph which includes a critical path, and that an optimization strategy is used which optimizes for total turnaround time. Such a consumer would be willing to pay more than the nominal worth of a CPU of category 2 for allocating jobs on the critical path, as they have a potentially large effects on turnaround time.

In realistic situations, not all consumers exercise the same buying strategy. Due to this, the simulator allows for pluggable strategies which determine the consumer’s behavior when presented with a given price vector. Currently, however, only the strategy described above is implemented.

### 3.3. Provider model

Providers host a configurable number of resources of different types. In order to determine whether a provider will sell resources of a certain type, it first calculates the average price $P_m$ for which the provider has sold slots of this type in the past. If the current price $P_c$ is greater than $P_m$, all available resources of this type are sold. If the price is lower, only the fraction $P_c/P_m$ is sold. Providers are thus prepared to limit their supply of free CPUs to the market in order to keep prices high, thereby taking on the strategy of maximizing revenue instead of utilization.

However, the fewer resources that are sold, the lower the mean price becomes, which in turn leads to an increase in the number of resources offered at price level $P_c$. The speed of these downward price adjustments is controlled by the elasticity attribute $E$, which determines the length of the time period that is taken into consideration for obtaining $P_m$. If $E$ equals zero, the provider does not take past revenues into account when determining supply, and always brings every free CPU into the market. If $E$ equals infinity, the provider takes all of the revenue into account that was generated in the past. This way, $E$ determines the provider’s reluctance for adjusting their supply when confronted with downward price adjustments.

If a consumer decides to buy a CPU resource at a current price level $P_c$, the consumer will be charged with this fixed rate for the entire duration of the job. An alternative to a fixed rate is to allow a variability in the charged rate over the duration of the job based on the market’s price evolution. Another option is to allow variability on the performance a consumer receives for a given rate over the job’s execution period, an approach adopted in [22]. These alternatives allow for faster reallocation of resources according to the dynamic market situation, but make budgetary planning and resource usage planning more difficult for consumers.

### 4. Pricing strategy

As mentioned in the introduction, resource costs should be dynamic, depending on demand, supply, value, and general wealth levels. In a dynamic pricing model, prices communicate complex consumer and provider valuations of the different resource types. These markets have emergent behavior, in that they react to changes in valuations. They are a means for achieving equilibrium between supply and demand.

Consider a market of $n$ substitutable commodities and corresponding prices, each measured by non-negative real numbers. A price vector $p = (p_1, \ldots, p_n)$, $p_i \geq 0$, where $p_i$ represents the price of a unit of the $i$th commodity. Let $R^n_+$ denote an $n$-dimensional real Cartesian space, and $R^n_+ = \{ x \in R^n \mid x_i \geq 0 \}$. We consider an economic setting with supply and demand functions $S, D : R^n_+ \rightarrow R^n_+$, with $S, D$ functions of prices in $R^n_+$. The condition for economic equilibrium is that supply equals demand, or $D(p) = S(p)$ as a condition on the price system $p$. The derived excess demand is the function $\xi : R^n_+ \rightarrow R^n_+$ given by $\xi(p) = D(p) - S(p)$. Thus, $\xi_i(p)$ is the excess demand for the $i$th good at price vector $p$. As defined, it can be positive or negative; negative excess demand can be interpreted simply as excess supply. Three hypotheses on $\xi$ are:

1. $\xi(\alpha p) = \xi(p)$ for $\alpha > 0$ (homogeneity)
2. $p \cdot \xi(p) = 0$ (Walras’ Law)
3. If $p_i = 0$, then $\xi_i(p) > 0$.

It has to be pointed out that the excess demands for the different goods are interrelated. Some reasons for this are that (1) goods are interchangeable, so the more one buys from good $i$, the less one needs from the other goods, and (2) one only has a limited budget, so the more one buys from good $i$, the less credits one has left to buy other goods.

The market equilibrium is characterized by a price vector $p^*$ where supply and demand equal one another. This means that in practice, finding an equilibrium is equivalent to an $n$ dimensional minimization problem for the surface defined by the norm $||\xi(p)||$. For example, for a model with two substitutable goods, the search space is a two-dimensional surface, as shown in Fig. 1. Many fundamentally different techniques for solving such minimization problems exist. In our work, we have chosen to use a well-known technique from quantitative economics, namely Smale’s method [43].

#### 4.1. Smale’s method

Smale’s theorem says that given a market consisting of $n$ interrelated commodities with price vector $p$ and associated excess demand vector $\xi(p)$, an equilibrium point $p^*$ exists such
The slope of the search space should never be 0. Assuming a continuous search space, $p^*$ is found by solving the differential equation

$$D_{\xi}(p) \frac{dp}{dt} = -\lambda \xi(p)$$

where the sign of $\lambda$ is $-\text{sgn}(\text{Det}D_{\xi}(p))$. Here, $D_{\xi}(p)$ is the linear transformation with matrix representation

$$D_{\xi}(p) = \left( \frac{\partial \xi_i}{\partial p_j} \right) \approx \left( \frac{\Delta \xi_i}{\Delta p_j} \right).$$

In practical applications of Smale’s method, the partial derivatives are approximated by discretization, as indicated above. A limitation of $\Delta p$ to 1, as in [6], however, leads to problems, as will be described later on.

4.2. Smoothing the search space

Using Smale’s method (Eq. (2)) to solve $p^*$ assumes a number of conditions, some of which are no longer guaranteed when operating on a Grid market. The problematic assumptions are:

1. If $p_i = 0$, then $\xi(p) > 0$. If, at a certain time step, no consumer needs to schedule a job, this condition is violated and cannot be corrected.
2. The search space is continuous at every point. This condition is violated whenever, at price $p_i$, a provider sells $n$ units of good $i$, but at price $p_i - \epsilon$, only $n - 1$ goods are sold. Comparably the condition holds for consumer behavior.
3. The slope of the search space should never be 0. This condition is violated in situations where a provider/consumer wants to sell/buy an equal amount of goods for prices $p$ and $p + \epsilon$.
4. Negative prices are not allowed.

Except for the first, these problems can be addressed by introducing a series of modifications to the search space and some extensions to the search algorithm.

4.2.1. Search space modifications

The first modification is that the excess demand value for negative prices is artificially heightened. The more negative the price, the higher the excess demand. Since we are dealing with a minimization algorithm, this modification will guide the algorithm away from negative prices.

Secondly, for all price vectors which lead to structural over-supply, the excess demand surface will be flat. This is the case whenever prices are so high that no one buys anything anymore. To guide the algorithm back to lower prices, we have introduced a slope on this part of the excess demand surface.

Thirdly, during price formation, providers and consumers will express their demand and supply in fractional units. For providers, the fractional supply for good $i$, $s_{f,i}$, is formulated as $s_{f,i} = r_a \times (p_{w}/p_c)$, with $r_a$ the amount of available resource $s$ of type $i$, $p_{w}$ the wanted price and $p_c$ the current price. The actual supply for good $i$, $s_{a,i}$, which will be traded during the scheduling activity, is the rounded part of the fractional supply. Averaging ensures that the total fractional supply approximately equals the total actual supply.

By introducing fractional resources on the consumer side, the search surface will be slightly elevated and as such, the minimum will no longer be 0. In the worst case, when (1) consumers have a leftover budget which is infinitesimally less than the price of the cheapest resource, and (2) both resources are equally expensive, the elevation is given by Eq. (4), with $n$ being the number of commodities.

$$\text{Max elevation} = \sqrt{\sum_{i=1}^{n} \left( \frac{1}{n} \right)^2} = \frac{1}{\sqrt{n}}.$$

Since the remaining budget, which is slightly less than the price of the cheapest resource, is evenly distributed among all commodities, which are all equally expensive, the excess demand for each commodity will increase by $1/n$. As a result, the search space will be elevated by $1/\sqrt{n}$. From this, we can conclude that we can no longer demand that the search continue until the norm equals 0: it should stop as soon as it hits the limit calculated by Eq. (4).

4.2.2. Search algorithm extensions

Even after the modifications presented previously, there still are some scenarios which would prevent the original algorithm from finding an optimum. Starting from Smale’s algorithm, we have incorporated three extensions in order to allow it to operate on the typical search spaces generated by market based Grid economics with interchangeable goods.

The first of these is plain rejection of negative prices. We have found that sometimes the algorithm, as presented in [44], returns negative prices. For this reason, whenever a price vector contains a negative component, the new price is rejected and the previous one is re-used.

A second scenario involves the algorithm stranding on a ridge, meaning that a price change for some good does not trigger a change in excess demand. This can occur when $p_j$ is so high that no demand is formulated under this price level. Price $p_j + 1$ then results in a $\Delta \xi$ of 0. In this situation, we remove all 0-columns and corresponding rows from $D_{\xi}(p)$, and as a consequence, the price for these goods will not change.

Finally, we have observed that if $\Delta p$ is fixed to 1, as is the case in [6], the algorithm can get stuck in a local minimum. To solve this, we have introduced a step size $SS$ with $\Delta p = 2^{SS}$. Now, when a $\Delta p$ of 1 does not yield a better norm, the
4.3. The algorithm

The algorithm used to solve Eq. (2) is a combination of the algorithm described in [44] and the extensions described above. An activity diagram is shown in Fig. 2. For the implementation, we need a definition for $|\xi(p)|$. We use the familiar Euclidian vector norm.

$$|\xi(p)| = \sqrt{n \sum_{i=1}^{n} (D_i - S_i)^2}.$$ 

With $n$ being the number of commodities, and for a given price, $S_i$ being the supply of resource $i$, and $D_i$ being the demand for resource $i$.

The algorithm starts by assigning the previous price vector $p_0$ to the current price vector $p$. The recursion depth $RD$, the price step $PS$, and the step size $SS$ are all initialized to 0. The base norm $n_0$ is calculated using the base price vector $p_0$. A new price vector $p'$ will only be accepted if its norm is better than $n_0$. Finally, $\Delta p$ is initialized to $2SS$, which comes down to 1.

Next, $J(p)$, the determinant of $D_{\xi}(p)$ is calculated. If $J(p)$ equals 0, $p$ is located on a ridge and an order reduction is necessary. Otherwise, $\lambda$, a factor which influences the extent of the price step, is set to 1 and the new price vector $p'$ is calculated using the algorithm described in [44]. If $p'$ contains a negative price, or if the change in the norm is too big, the value of $\lambda$ is divided by two and the price-setting part of the algorithm is restarted. This iterates until all prices are positive and the change in norm is within range, or until $\lambda$ is so small that the magnitude of the price changes would fall below the precision of the computer.

If there are still negative prices, or if the new norm is worse than the base norm and $SS$ did not reach its maximum value yet, $SS$ is increased by one and the algorithm is iterated. Currently, $SS_{max}$ is limited to 10, which results in a price step of 1024. If no better norm can be found within this range, the algorithm assumes it is in the global minimum.

If, at this point, there are still negative prices, the algorithm is restarted from its initial price vector $p_{default}$. This recursion is allowed only once, to prevent an infinite loop. We have observed this phenomenon in situations where (1) prices are very close to 0 because there was a long period of over supply, (2) there is a sudden and strong spike in demand due to a job peak, and (3) the demand for one good is much higher than for the others.

If the price vector still contains negative prices or it has a norm which is worse than the base norm, the algorithm ends by reusing the old price vector. On the other hand, if the new price vector produces a better norm, it will become the current price vector. Now, one step in finding the minimum is finished. The algorithm starts over using this price vector as a starting point. This continues until the norm is less than $\epsilon$ or the algorithm has iterated more than $PS_{max}$ times.

5. Simulation process

We have implemented the market model discussed in the previous sections in our own Java-based discrete event simulator [45]. This enables us to efficiently study the dynamics of commodity markets and the state of the Grid environment in a controlled manner, and under differing factors such as market size, resource categorizations, budget allocations, and participant strategies.

The simulator’s configuration framework allows the consumer to build up a scenario configuration for an experiment that can be persisted to permanent storage for repeatability. This allows for tuning the configurable aspects of the different domain objects that implement the market model, such as
the strategies for providers, consumers, and pricing, or the consumers’ job generation patterns. An output management framework allows for the extraction and inventarization of output metrics defined on the different classes that make up the simulated environment.

A graphical user interface, shown in Fig. 3, hooks up with the output management layer and the configuration layer. This enables the user to configure the scenario before launch, and to monitor different aspects of the market environment as the scenario advances. The output visualization panel shown on the right of Fig. 3 allows the user to flexibly set up graphs to which monitorable metrics from the simulation environment can be attached as data sources. For ease of use, the configuration of the visualization panel can also be persisted to disk.

To launch a scenario, its configuration is first instantiated to obtain a live instance of the simulated market environment. The environment’s controller can then be instructed to advance the environment’s state in discrete simulated steps. Each stimulation step consists of the following:

1. The budget of each consumer is updated. If, in this first step, or during another step a consumer’s allowance period \( ap \) is reached, the budget is reset to the allowance \( a \). Both parameters are bounded random numbers.

2. Each consumer’s job queue is updated. For all consumers who have reached their job peak step \( ps \), a configurable number of jobs is added to the queue. For all other consumers, a job is added to the queue according to the consumer’s job creation probability. The peak period, the number of jobs added, and the job creation probability are bounded random numbers.

3. If there are non-occupied resources in the market, equilibrium price levels are established for these resources.

4. Actual trading occurs: in random sequence, every consumer allocates resources on the market at the price level determined in the previous step.

5. Charge consumers for all of their running jobs, and transfer associated revenue to the provider accounts.

6. Adjust the remaining running time of all active jobs and remove all finished jobs from the system, updating the resource availability states of the providers’ resources.

When, in an over-demand market situation, an equilibrium price is not found by our search algorithm, allocation is performed on a first-come-first-serve basis. Therefore, it is important that we randomly iterate over the consumer population in step (4), so no consumer is favored when resource allocation takes place in such a situation.

6. Algorithm evaluation

The evaluation of the ideas presented above can be broken down into two parts. The stability and efficiency of the algorithm, discussed here, and the fairness and correctness of the market mechanism, discussed in Section 7.

To test the stability and efficiency of the algorithm, we have constructed a turbulent market situation. The simulation parameters are presented in Table 1. There are 100 consumers, each submitting new jobs in accordance with their job creation probabilities. Besides this, they submit a random number of jobs every 100 time steps. We have opted to make each consumer’s peak period coincide, so that the market instability
Table 1
Parameters common for all performed simulations

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation steps</td>
<td>2000</td>
</tr>
<tr>
<td># Consumers</td>
<td>100</td>
</tr>
<tr>
<td># Providers</td>
<td>50</td>
</tr>
<tr>
<td># fastCPU per provider</td>
<td>{1,2, … ,8}</td>
</tr>
<tr>
<td># slowCPU per provider</td>
<td>{2,3, … ,15}</td>
</tr>
<tr>
<td>$E$ attribute for all providers</td>
<td>$\infty$</td>
</tr>
<tr>
<td>Performance ratio</td>
<td>2.0</td>
</tr>
<tr>
<td>Job length</td>
<td>{2, … ,10}</td>
</tr>
<tr>
<td>Allowance period</td>
<td>500</td>
</tr>
<tr>
<td>Base allowance</td>
<td>[500,000, … 1,250,000]</td>
</tr>
<tr>
<td>Peak period</td>
<td>100</td>
</tr>
<tr>
<td>Jobs submitted at peak</td>
<td>{1,2, … ,200}</td>
</tr>
<tr>
<td>New job probability per step</td>
<td>10%</td>
</tr>
</tbody>
</table>

Fig. 4. Price evolution.

at these steps would be maximum. Furthermore, each consumer receives a random allowance every 500 steps. The allowance period is a multiple of the peak period in order to show the interplay between them. Jobs have a random nominal running time. There are 50 providers in the simulation, each supplying a random number of fastCPUs and slowCPUs. The simulation is run for 2000 steps. The price evolution in this simulation is shown in Fig. 4. The dotted vertical lines represent the peak steps. One can see that prices rise sharply whenever a demand peak has occurred. Furthermore, the price of a slowCPU follows the same trend as the price of a fastCPU, but is always about 65% lower. This is due to the fact that, on average, consumers prefer to buy a fastCPU until it is more than three times more expensive than a slowCPU. As the demand diminishes, prices drop. They do not drop suddenly, because of the large value of the elasticity attribute $E$ (cfr Section 3.3) and the gradually diminishing nature of the consumer job queues. One can also observe that the consumer behavior is very defensive. The further consumers are away from the next allowance period, the less money they are willing to spend. We would like to note that in periods of excess demand, more than 600 resources are traded per step.

The stability and efficiency of the pricing strategy can be analyzed by looking at $|\xi(p)|$, found through minimization. Fig. 5 shows the norm for each simulation step and its relative frequency distribution over the whole simulation. Table 2 lists some key statistical figures. From this, we conclude that in general the norm is low: more than 50% of all steps find an optimal norm and 95% has a norm less than 5.13. The spikes occur at the steps where the price drop ends and a new market equilibrium is found. After a couple of steps, the market stabilizes and the absolute norm drops once again to near 0 level.

Eq. (6) defines a second metric, which we refer to as the mismatch factor.

$$\text{relNorm} = \frac{\sum_{i=1}^{n} |S_i - D_i|}{\sum_{i=1}^{n} |S_i| + |D_i|}.$$  

(6)

It is a relative indication of the number of goods for which supply and demand did not match. It varies between 0 and 1, where 0 means a perfect allocation and 1 means that none of the resources could be matched.

Fig. 6 shows the mismatch factor for each simulation step and its relative frequency distribution over the whole simulation. Table 2 lists some key statistical figures. From these sources, we can derive that in more than half of the simulation steps, the mismatch is less than 0.5%, and in 90% of all cases, the mismatch is less than 5.82%. The peaks in the mismatch factor coincide with the area of near-zero prices. This is logical, because in these simulation steps there is a structural oversupply. Providers are trying to sell resources, but there just

Table 2
Statistical analysis of the norm and mismatch factor

<table>
<thead>
<tr>
<th>Norm</th>
<th>Mismatch (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>0.00</td>
</tr>
<tr>
<td>25th percentile</td>
<td>0.43</td>
</tr>
<tr>
<td>50th percentile</td>
<td>0.68</td>
</tr>
<tr>
<td>75th percentile</td>
<td>1.31</td>
</tr>
<tr>
<td>90th percentile</td>
<td>2.87</td>
</tr>
<tr>
<td>95th percentile</td>
<td>5.13</td>
</tr>
<tr>
<td>Max</td>
<td>21.12</td>
</tr>
</tbody>
</table>
is no demand for them due to job shortages in the consumer queues.

To conclude, we would like to remark that the norm and the mismatch factor should be considered together for a complete interpretation. A peak in the norm of 25 may seem bad, but if the mismatch factor indicates that this accounts for only 0.1% of all traded goods, we actually have a very good match. Conversely, a mismatch of 50% looks unacceptable, but if the norm is very small, then this indicates that, during this simulation step, there were only very few goods traded anyway.

7. Evaluation of the market mechanism

In this section, we present an evaluation of the workings of our resource market. We focus on the following desirable market properties:

1. Prices should be set in such a way that market equilibrium
   is reached.
2. The ratio between price levels for different goods should reflect the mean valuation ratios of those goods by the consumer population as a whole.
3. The resource market should lead to fair allocations.
   a. In an oversupply situation, this means that every consumer should be able to acquire an equal share of the infrastructure if the job spawning rates over all consumers are equal.
   b. In an overdemand scenario, wherein consumers are not limited by a shortage of jobs in their queues, this means that a consumer in, with budget share BS, should be able to allocate a share of the total utility US with

\[ US_i = BS_i. \]

(c) In a market where consumers value the computational resources according to their nominal worth, this means that for every consumer, BS = IS must hold. Here IS is the share of compute resources allocated by the consumer during the simulation.
4. The measured resource utilization levels should be equal or close to the maximum achievable utilization.
5. Prices should be stable in the sense that limited, short-term changes in supply and demand lead to limited, short-term price responses in the market.

In the context of point 4, note that all of our providers share the strategy of maximizing personal revenue instead of maximizing the utilization of their infrastructure.

Experimental data on the real-world execution of economic Grid resource management systems are scarce, as such systems still need to find their way to the Grid middlewares that are deployed on a large scale. In the absence of such reference data for configuring our market, we investigate aforementioned properties under two distinct and typical scenarios. The first represents a market in oversupply, which means that, on average, the total available CPU time in the market is greater than the total computational time requested by all consumer jobs. Following [6], we inject a diurnal characteristic into the job flow in order to evaluate market efficiency and stability under sudden demand changes. The second scenario simulates a market in constant overdemand.

In order to evaluate the fairness of our market, we introduce four consumer groups with distinct allowance levels. A consumer C in allowance group AG is allocated a budget \( B_C = A \cdot AF \), with A the base allowance for all consumers and AF the allowance factor for AG. We evaluate market fairness by establishing that the average budget share of consumers in AG equals their average infrastructural share IS. In the context of our current market model, IS denotes the share of CPUS which have been allocated by allowance group AG over the course of the simulation.

7.1. Oversupply scenario

The simulation parameters for the oversupply scenario are given in Table 3. For parameters that are specified with a range, we use a uniform random distribution. This scenario simulates a market with 100 consumers and 50 providers. Every provider hosts between one and eight fast CPUs, and between two and fifteen slow CPUs. Jobs have a nominal running time of between 2 and 10 simulation steps, and every consumer has a 10% probability of spawning a new job during each simulation step. At the job induction steps, between 1 and 200 jobs are spawned per consumer. In our instance of this scenario, the market contained 456 slow CPUs and 222 fast CPUs. The jobs that were spawned during the 450 simulation steps represented a total nominal compute time of 256,098 simulation steps. The infrastructure provides 900 nominal computation steps per simulation step. Therefore, the highest achievable utilization rate is 63.3%, which corresponds to an oversupply situation.

As shown in Fig. 7, utilization levels start at 100% during the first job peak and slowly decrease as we move into oversupply. At the following job peak, the utilization levels rise again. The utilization at the end of the simulation is 61.7% for fast CPUs and 64.6% for slow CPUs. This means that our average utilization for the infrastructure is 63.15%. Our market thus exhibits a very high allocative efficiency, even in scenarios where our provider population strives for maximization of revenue instead of infrastructural utilization. This means that property (4) is fulfilled for this oversupply scenario. Fig. 7 also shows that prices follow supply and demand closely. At the
Table 3
Parameters specific to the oversupply simulations

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation steps</td>
<td>450</td>
</tr>
<tr>
<td>Pref\textsubscript{fast}</td>
<td>[1.0,2.0]</td>
</tr>
<tr>
<td>Pref\textsubscript{slow}</td>
<td>1.0</td>
</tr>
<tr>
<td>AF\textsubscript{1}</td>
<td>1.0</td>
</tr>
<tr>
<td>AF\textsubscript{2}</td>
<td>1.5</td>
</tr>
<tr>
<td>AF\textsubscript{3}</td>
<td>2.0</td>
</tr>
<tr>
<td>AF\textsubscript{4}</td>
<td>2.5</td>
</tr>
<tr>
<td>Allowance period</td>
<td>100</td>
</tr>
<tr>
<td>Base allowance</td>
<td>500,000</td>
</tr>
</tbody>
</table>

All other parameters are as in Table 1.

job induction steps, the prices of both goods are immediately adjusted to bring the market to equilibrium. After a price peak, prices decline as demand drops below the supply level due to a shortage of jobs, and the market returns to the steady state in which there is oversupply. Since providers do not set minimum prices for resource usage, prices converge to zero at that point. This indicates that prices are stable, in the sense that after disturbances due to the job induction, price levels gradually return to their steady state. This fulfills property (5).

As shown in Table 4, the median of the norm has a value of 0.64 with a 95th percentile of 5.8. The prices set by our pricing scheme thus approach the market equilibrium very closely, thereby fulfilling property (1).

The graphs in Fig. 8 show the effect of our price alterations on the excess demand levels. The grey graph displays the excess demand levels that would arise if we used the price calculated in step $i - 1$, for step $i$. The black graph displays the excess demand levels for the prices calculated by our pricing scheme. We clearly see that the peaks in excess demand at the job induction steps are neutralized by our price adjustments. At step zero, the pricing algorithm is bootstrapped with a default price vector of (100,000, 100,000). As we can see from the graph, the default price vector needed correction by our pricing scheme, because it would otherwise have induced a full oversupply situation, leaving all of the 222 fast CPUs unallocated. Note that the peaks in excess demand are not fully shown in the graph, as this would require an $Y$ axis range of $[-250, 5000]$.

The mean preference factor for fast CPUs over the entire population is 1.5, and the performance factor is 2. Therefore, fast CPUs should be valued at three times the price of slow CPUs. Table 4 shows that prices approximate this relative valuation, but are not in perfect correspondence with it. This can be explained by the fact that consumers favoring fast CPUs for a particular price vector can still acquire slow CPUs with the budget resources they have left after buying their fast CPUs. The acquisition of these slow CPUs however, takes place regardless of their price. This elevates the aggregate demand for slow CPUs and their price level. If we disable this behavior in the consumer strategy, the mean valuation factor is correctly reflected in the price levels through a median price of 85.05 for fast CPUs and 28.13 for slow CPUs. We therefore conclude that property (2) is fulfilled.

Fig. 9 shows the budget share and allocation share of the four consumer groups in the simulation. We notice that the allocation shares oscillate between the job induction pulses. This behavior is to be expected; a consumer group with a high budget share is only able to allocate its affordable resource share fully when enough jobs are available in the consumer queues. As the job queues shrink, prices drop and other consumer groups are able to allocate resources. As prices converge to zero, the allocation shares remain approximately constant. When a new price peak arises, the allocation shares for wealthy consumers gradually rise again, at the expense of the poorer consumers. The oscillatory effects are damped by the effect of averaging the allocation shares over all simulation steps. We see that allocation shares converge to the 25% mark. This corresponds...
with the notion that in a market with oversupply, differences in consumer budgets do not affect their allocation shares in the long term, i.e., every consumer is able to allocate an equal share of the Grid’s resources. This fulfills property (3)(a).

Although the market’s behavior can be intuitively analyzed when the demand peaks of the consumers are synchronized, such synchronization cannot be expected in a real-world scenario. Fig. 10 shows the evolution of the market prices when the demand peak period for each consumer is uniformly distributed between 90 and 100 simulation steps. As the simulation advances, the demand spikes of the consumers drift farther apart from each other, resulting in price peaks earlier on in each 100 time-step period.

### 7.2. Overdemand scenario

For the over-demand scenario, we keep the number of jobs in the consumer queues at the level obtained during the first job induction at simulation step zero. All other parameters are equal to those of the oversupply scenario. Fig. 11 shows the price and utilization levels for fast and slow CPUs in the over-demand scenario. Although one might expect prices to remain constant, they rise as we approach the allowance replenishment steps (at intervals of 100). This can be explained by the fact that consumers always have an expenditure rate within the limits of their allowed spending rate per simulation step. Because of the unit price per CPU, consumers are not generally able to spend all of their budget. This way consumers “save” up budget leftovers from the previous step within one allowance period. Since the available spending rate for a particular simulation step is calculated as a function of the remaining steps until the next allowance period, higher spending rates are possible near the end of the period. As we reach the allowance replenishment step, all leftovers are discarded and prices drop, after which this price evolution repeats.

The graph shows that our utilization level remains constant at approximately 100%. The mean utilization level for fast CPUs was 99.6% and 99.7% for slow CPUs. Property (4) is thus fulfilled in the over-demand scenario.

Again, the data in Table 4 shows that we achieve price correctness with the median of the norm at 0.29 and a 95th percentile of 1.21, thereby fulfilling property (1). Although supply and demand remain fairly constant in the over-demand scenario, Fig. 12 demonstrates that it is still crucial to perform small price adjustments in order to keep excess demand levels low. This time, we show the excess demand levels for the slow CPU category, both when using the price level of the previous simulation step (grey graph) and when using the adjusted prices (black graph) for the current simulation step.

If we increase the allowance period to 1000 simulation steps and compensate with a tenfold increase of allowance for all consumers, we obtain stable and fairly constant price levels, fulfilling property (5). This is shown in Fig. 13.

Figs. 11 and 13, as well as Table 4 show that the price levels of both CPU types approximate the relative valuation of these types by the consumer population. The relation does not perfectly reflect the valuation factor of 3, and the same explanation holds as in the oversupply scenario.

In the over-demand scenario, all consumers are able to fully exert their budgetary capabilities in order to allocate their share of the Grid infrastructure. As shown in the graph in Fig. 14, the mean allocation share for a particular budget group corresponds to its budget share. This fulfills property (3)(b).

Two important premises exist for obtaining this fair allocation, that are outside the control of our pricing scheme.
and market operation. The first is that a consumer’s spending rate should not be limited by a shortage of jobs in the consumer queue. As we keep the number of jobs in the consumer queues constant at a high enough level, this premise is fulfilled. The second premise is that consumers have to value the CPU types according to their real nominal worth. This is because we compare the budget share to the share of allocated resources \( I_s \), and not to the share of total perceived utility \( U_s \). Note that in the context of an individual consumer, this premise does not necessarily hold, as the consumer may add extra relative value to a CPU type through its \( \text{PrefFactor} \). However, because the preference factors are uniformly distributed over the consumer population, the premise does hold when we approach the fairness issue from the perspective of the individual budget groups.

8. Future work

First, some aspects of our algorithm require further study. Specifically, we need to investigate its behavior in extreme market conditions such as a very limited number of participants. Another point of research will be to improve the search algorithm in an attempt to remove the remaining mismatches. Furthermore, we will work on expanding the dimensions of the search space. This is important for extending our market model to more types of complementary, or substitutable goods.

Secondly, we have observed that the number of excess demand queries can be very high in some states of the market environment. This could be a significant drawback for an implementation in real life Grids. As such, a scalability study is in order. We are also considering modeling techniques to reduce the number of queries [46], and tree-based topologies to reduce the network load.

Thirdly, our model has to be extended in order to handle the locality of resources and their associated transport costs.

Finally, we realize that for economics-inspired resource sharing models to truly gain influence in contemporary Grid technology, we must transfer our ideas and techniques from a simulation environment to real-world environments.

9. Summary and conclusions

In this contribution, we have proposed a market model which uses a state-based, non-pre-emptive strategy belonging to the category of commodity markets. Our market model consists of three parties: consumers, providers, and the market itself. Consumers buy resources according to a buying strategy, providers sell resources following a selling strategy, and the market establishes an optimal price, matching demand and supply, using a pricing strategy. As such, we have presented a pricing scheme that responds to the dynamics of demand and supply in the market.

A vision of the Grid where applications can treat computational, network, and storage resources as interchangeable, and not as specific machines, networks and disk systems, corresponds closely to a commodity market. However, there are some problems with this analogy. First, the notion of locality is not taken into consideration, and secondly, besides performance, other aspects, such as quality of service, available (cache) memory and network bandwidth, contribute to the value of a compute node. Although these would be arguments against the use of commodity markets in Grids, fuzzy clustering can offer a solution where one tries to group all the compute nodes in a number of categories based on a fuzzy combination of their characteristic properties.
We have taken the state-of-the-art in commodity-based Grid economic models one step further, by extending it to allow for trading and pricing of substitutable goods. This more closely models Grid markets, where multiple resources are available to process the jobs, but with varying performance characteristics. The proposed model is based on Smale’s algorithm, but has a number of enhancements in order to allow it to operate in typical Grid economies settings.

First, the search space has been tweaked in the following ways: (1) the excess demand value for negative prices is artificially heightened; (2) a slope has been introduced in the structural oversupply region of the search space; and (3) during price formation, providers and consumers will express their demand and supply in fractional units, which results in a smoother search space.

Second, we introduced improvements to the optimization algorithms used to compute the equilibrium prices: (1) sometimes, the algorithm suggests negative prices, which should be immediately rejected instead of letting the algorithm gradually return to positive prices; (2) sometimes, the algorithm can get stuck on a ridge in the search space, requiring a temporary order reduction of the search space; and (3) we have observed that if the price deltas are fixed, the algorithm can get stuck in a local minimum. An adaptive price adjustment scheme can solve this issue.

We have evaluated the proposed model by building a Grid economics simulator, which allows us to study the emergent behavior of our market model and investigate the impact of different simulation parameters, such as the number of participants and their strategies, job arrival rates, and budget distributions.

The evaluation of the proposed model has been broken down into two parts — the stability and efficiency of the algorithm, and the correctness of the market mechanism. Simulations have shown that the proposed algorithm is stable and efficient under the presented conditions. Furthermore, we have demonstrated that the proposed market model achieves the desirable properties of price correctness and stability, allocative efficiency, and fairness.

References


Dr. Gunther Stuer is a post-doctoral fellow in the Department of Mathematics and Computer Science at the University of Antwerp (UA), Belgium. His research interests include Grid Economics. In 2003 he received his Ph.D. in Computer Science, at the University of Antwerp (UA), Belgium.

Kurt Vanmechelen is a Ph.D.-student at the Department of Mathematics and Computer Science at the University of Antwerp (UA), Belgium. His research interests include Grid economics and Grid resource management in general. He is a member of the steering board of the national Belgian research Grid initiative BEgrid.

Dr. Jan Broeckhove is a professor in the Department of Mathematics and Computer Science at the University of Antwerp (UA), Belgium. His research interests include computational science and distributed computing. He received his Ph.D. in Physics in 1982 from the Free University of Brussels (VUB), Belgium.