Market Mechanisms for Trading Grid Resources

Costas Courcoubetis[1], Manos Dramitinos[1], Thierry Rayna[2],
Sergios Soursos[1], and George D. Stamoulis[1]

[1] Network Economics and Services Group (N.E.S.), Department of Informatics,
Athens University of Economics and Business (AUEB),
76 Patision Street, Athens, GR 10434, Greece
{courcou,mdramit,sns,gstamoul}@aueb.gr
[2] Internet Centre, Imperial College London, South Kensington Campus,
180 Queen's Gate, London SW7 2AZ, United Kingdom
t.rayna@imperial.ac.uk

Abstract. There has been recently an increasing interest in Grid services and economic-aware Grid systems both in the industry and the academia. In this paper we specify a market for hardware providers and consumers interested in leasing Grid resources for a time period. Our approach comprises a stock-market like mechanism that enables the trading of computational power on the basis of a spot and a futures market. The spot market comprises a pair of bid and ask queues. This grid market is more complicated than the standard spot/futures markets of storable commodities, because the computational service traded in our case comprises of resources that are perishable, and has both quantity and duration specified in terms of a time interval. This is an important feature of our market mechanism, complicating considerably the trading algorithms that we develop and assess in this paper.

Keywords: Market mechanism, grid market, bid and ask, spot, futures.

1 Introduction

Motivated by the electrical power grids and the time-sharing computational systems of the past, there has been an increasing interest in Grid services over the past years. In order to materialize the virtualization and wide-scale sharing of computational resources, a variety of related business models regarding utility computing and software on demand have been developed, while economic-aware Grid systems have become increasingly popular both in the industry and the academia [1]. A wide variety of related market systems have been proposed; these are based on fixed prices, bartering, negotiations or auction models for leasing "Grid contracts". A detailed overview and presentation of these economic-aware grid systems and architectures can be found in [2] and [3]. However, despite the various economic mechanisms that have been proposed as candidates to be adopted in a Grid market, very few efforts have been made to fully specify the design of a market that is tailored to the Grid products and services. Indeed most proposals neglect taking into account the fact that both the resource type and the time dimension are of significant importance for the perishable Grid resources.

J. Altmann, D. Neumann, and T. Fahringer (Eds.): GECON 2008, LNCS 5206, pp. 58–72, 2008.

In this paper, we specify a marketplace comprising all functionality for the leasing of computational service for a time period. It can serve as the core of the Grid economy. Customers (and hardware providers) interact through the marketplace, possibly by means of brokers, in order to lease (respectively offer) Grid resources. In Sect. 2 we outline a stock-market like mechanism and a corresponding system architecture that enables the trading of computational power on a spot market basis [4], as well as for selling futures contracts [5]. The underlying principle for this mechanism is that of a standard spot and futures market: All parties announce the maximum price they are willing to buy for and the minimum price they are willing to sell for. The spot bids (resp. asks) are put in the spot queue for bids (resp. asks). The futures are listed in the directory service of the futures market. All the compatible trades, i.e. when a bid is matched with a set of asks, are immediately executed. Note that matters are more complicated for our system's spot market than in standard spot markets of storable commodities, because in our case a the computational service is traded. This service is non-storable, because its specification includes both the quantity of resources and the relevant time interval. These matters are clarified in Sect. 3, 4 and 5, where we also introduce and assess an economically sound algorithm for matching bids and asks. Some additional important issues regarding matching that are left for future work, including the outline of a more sophisticated matching algorithm, are presented in Sect. 6.

2 The Marketplace and Its Architecture

So far, we have outlined the core functionality of the marketplace, i.e. the Grid Market, which is the main focus of this paper. It is worth noting that in order to be feasible for a *realistic* Grid marketplace to fully support the market mechanisms, a set of additional subsystems must be implemented as well. These subsystems are common among all existing (e.g. e-commerce) marketplaces, are not Grid-specific and both complement and support the core functionality of the Grid marketplace. Their detailed description is beyond the scope of this paper, which focuses on the presentation of a simple, fast and applicable, yet economically sound, Grid marketplace mechanism and the set of algorithms it comprises. However, we outline the marketplace system architecture (also depicted as Fig. 1) and highlight the subsystems functionality below for completeness reasons.

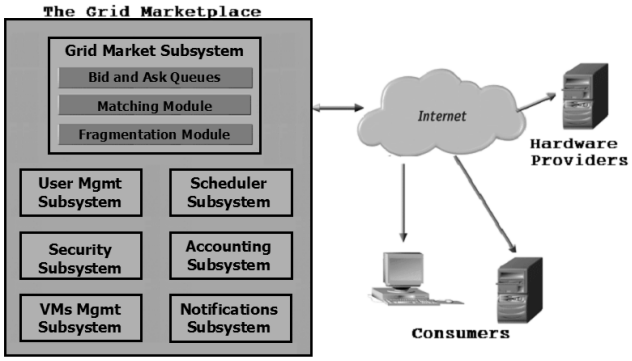

Fig. 1. The marketplace system architecture

User Management Subsystem: This subsystem is responsible for admitting users and providers into the marketplace. It uniquely authenticates and admits users/providers into the system, stores/checks their respective credentials and also interacts with the accounting/logistics and notifications subsystem.

Resource Management Subsystem: This subsystem is responsible for the management of the computational elements of the Grid marketplace and for the binding of the resources offered with a certain economic mechanism.

Security Subsystem: This subsystem enforces the marketplace's rules in the market transactions. It performs a wide variety of checks. For instance, it checks that the resources offered by the providers are indeed idle. It also interfaces with the Accounting and Logistics subsystems described below.

Accounting/Logistics Subsystems: Perform accounting and logistics management.

Directory Services Subsystem: This subsystem allows the organization and the advertisement of the leasing of resources. It is complemented with a search.

Notifications Subsystem: This subsystem is responsible for sending notifications to users. There are plenty of cases where this is desirable, such as: inform a bidder whether his bids are winning or not, or send reports to the virtual machines providers about their resource usage status and the respective revenue attained.

Scheduler Subsystem: This subsystem allows the execution of tasks at certain epochs, possibly periodically.

3 The Grid Market

First, we need to define the service that is to be traded in the grid market. Obviously, this must be suitable for the types of Grid applications currently existing or emerging. Hardware providers offer for leasing virtual machines (VMs) of different types that can be traded by means of different mechanisms [6], [7], [8]. It is expected that these resources be offered for a minimum desirable price and for certain time duration within a specific time interval, depending on the providers' supply constraints. Note that a virtual machine does not just correspond to a certain computational speed, but rather to an entire configuration of the hardware. This configuration is henceforth and throughout the remainder of this paper referred to as VM or unit of computation; these terms are used interchangeably.

An additional assumption of our model throughout the paper is that time is discretized in time slots. For simplicity of presentation reasons, the duration of this time slot in all the examples presented in the remainder of the paper is taken to be 1 hour, though in practice it would be set to a different value.

Customers are interested in accommodating their needs for computational power from the Grid market. They can achieve this by leasing some of the virtual machines that the providers make available in the Grid market. Depending on the nature of the tasks consumers may wish to execute, their demand can be expressed in a multitude of ways. A general type of "contract" is specified by means of a certain rate of computation for a specific time interval. For instance, this could be the case for a

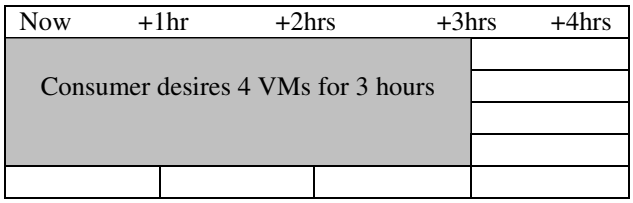

Fig. 2. A consumer's demand for 4 VMs for 3 hours is depicted by means of a rectangle

company's web server that leases Grid resources when it is critically loaded. This type of consumer need can be also graphically depicted by means of a rectangle (see Fig. 2): The height of the rectangle denotes the number of virtual machines required at any time of the interval, while width of rectangle denotes the amount of time for which these machines are needed.

Another type of contract could be specified by means of computational volume, i.e. a total number of VMs must be made available up to a maximum deadline constraint, so that a certain computationally intensive task is executed in time. As opposed to the previous case, only the total quantity of computational power is of interest, while the rate of computation provided at the various time epochs is not. This could be the case for a weather prediction program or a stock market data mining application that must be executed up to a deadline, i.e. the announcement of the weather forecast and the prediction of stock market prices before the markets open respectively. Note that the consumer needs in this case no longer correspond to rectangles, but rather to areas of rectangles, possibly with a maximum width (i.e. deadline) constraint. An extension of the Grid market mechanisms for this type of contracts is provided later in the penultimate section of this paper.

3.1 Bids

A bid in our system prescribes the resources required, which are specified by means of: a) the type and quantity of resources required, b) the starting time of the interval for using the resources, and c) the time duration of using the resources. It also specifies d) the price, expressed in €/min/unit and e) the time limit for which the bid applies. The latter is the maximum time at which the bid is considered to be valid. If this time is reached without the bid being matched, the bid must be removed.

The bid definition could also be complemented by a definition of whether or not the bid is atomic, i.e., it should be fully served by resources of a single provider. Atomicity may be the result of technological constraints on the possibility to switch execution environments. If such constraints are absent, economic theory suggests that the market should refrain from supporting atomic bids, due to the market power that large providers would obtain, which is not compatible with a bid and ask mechanism. In fact, even if the market would only support atomic bids, then it is likely that consumers would post-sale combine the resources of multiple such bids to obtain a more extended service Therefore, it is henceforth assumed that bids are non-atomic; more on this issue will be discussed later.

There are two types of bids in our system, namely future and spot bids. Future bids are the bids for which the starting and ending times are fixed instants in the future.

For instance, a future bid is the following: “User X bids for 5 processing power units (i.e. VMs) of type A to be used for 5 hrs, starting at time 13:00, with bid price 0.5 €/min/unit”. Note that in practice the time contains also the Date, but this is omitted for brevity reasons throughout this paper. Note also that different providers may offer the resources required or even a subset thereof; e.g. Provider Y1 offers 2 units from 13:00 to 14:00, while Provider Y2 offers 1 unit from 14:00 to 18:00 and Provider Y3 offers 4 units from 15:00 to 18:00. As opposed to future bids, spot bids demand to utilize resources as soon as they are available. Such bids are distinguished from futures by setting the starting time at a special value (e.g. 0) and by the fact that their start and end time are continuously moving as time passes and the bid is not matched (up to the maximum time allowed by the expiry of the bid). Therefore, spot bids are more flexible than future bids, since they allow users to express demand for service of a certain duration over a larger time interval, as opposed to futures. Note that the actual time of the service of the consumer in this case is a priori unknown, since this depends on when these bids will be matched by asks. For instance, a spot bid is the following: “User X bids for 5 VMs of type A to be used for 5 hrs, starting at time 0, with bid price 0.5 €/min/unit, and time limit 20:00”. In this example the bid could end up be executed the latest starting at 20:00.

3.2 Asks

An ask in our system prescribes the resources offered, which are specified by means of: a) the type and quantity of resources offered, b) the starting time and the end time of the interval when the resources are made available, and c) the total time duration of using the resources. It also specifies d) the price, expressed in €/min/unit and e) the time limit for which the ask is valid and can be used for matching bids. The latter is the maximum time at which the ask is considered to be valid, i.e., the provider of the ask will remove the ask or any remainder of it from the system after the above time. That is, this is the expiry time of the offer, and can be earlier than the maximum time deadline for which the resources offered in the ask can be made available to users.

Similarly with bids, there are also two types of asks, namely future and spot asks. Future asks are those for which the starting and ending times are fixed instants in the future. For asks, the ending time equals the starting time plus the duration, while the time limit also has the same value by default. For instance, a future ask is the following: “Provider Y offers for leasing 2 VMs of type A to be used for 8 hrs, starting at time 15:00, with ask price 0.2 €/min/unit”. On the contrary, spot asks offer resources that can be utilized as soon as there is demand for them. Such asks are distinguished from future asks by setting the starting time at a special value (e.g. 0) and because they are more flexible than future asks, since they offer service of a certain duration over a larger time interval. For instance, a spot ask is the following: “Provider Y offers 2 VMs of type A to be used for 3 hrs, starting at time 0 and until 22:00, with ask price 0.2 €/min/unit, and time limit 19:00”. The semantics is that up to two machines can be used each for up to three time slots (hours), not necessarily consecutive, during the next time slots; this ask will be removed from the system when the time is 19:00 if it has not been matched until then.

3.3 Bid and Ask Queues

Trading is performed by means of a continuous double auction mechanism. This is an extension of the standard spot market mechanism so as to provide for the trading of computation service that can be fully specified only when the associated time is also defined. Similarly to the standard mechanism, the spot bids and asks submitted by traders are placed in the *bid queue* and the *ask queue* respectively. Each queue is ordered according to the price and time of issuance, with the *bid* queue being sorted in *decreasing* order of price, and the *ask* queue being sorted in *increasing* order of price. In the case of the futures market, the bids and asks are listed in a directory service that enables searching and matching.

If two or more orders at the same price appear in a spot queue, then they are entered by time with older orders appearing ahead of newer orders. Since price is discretized in our model, then an equivalent representation of this queue is an ordered (per price) list of queues, one per price asked/bidded, where the asks/bids are sorted by time. The prices displayed to traders when they log into the market are the highest bid price in the bid queue and the lowest ask price in the ask queue. If no price is displayed it is because the corresponding queue is empty.

Orders remain in the queues until they are removed by the system due to expiration, or until they are accepted by other traders (a matching occurred) and result in trades. Expirations are determined according to the terms of the order. In particular, a spot bid expires at its time limit, and the same applies for spot asks.

3.4 Matching

The matching module is invoked when a new bid or ask is submitted. The rationale behind the matching module is that bids are completely satisfied, i.e. there are never remainder bids (parts of a bid that may be satisfied in the future). For simplicity reasons and in order to reduce the potential communication overhead that occurs for customers being served simultaneously by multiple providers, we assume that each bid is served by one provider at any time instance, while multiple providers can only be involved in different times during the servicing of the bid. This assumption will be henceforth referred to as *vertical atomicity*. This assumption is adopted for one more reason: the possibility of serving an application at a certain time with resources belonging to multiple providers depends on the parallelizability of the application, which would then have to be input to the market mechanism and taken into account thereby. Thus, we assume that the matching algorithm considers as candidate matches of a bid only asks whose height is greater than or equal to that of the bid.

Furthermore, we assume that the matching algorithm considers as candidate matches of a bid only asks whose price (per unit) does not exceed that of the bid. Therefore, we omit examining higher price asks and try combine them with lower price asks, even if such combinations could in fact serve the bid with the bidder attaining positive net benefit from the overall charge of the service. This is justified from an economic point of view, since serving customers using higher price asks than the price of the bid would be misleading and distorting for the information signals regarding the actual market price.

Note also that the bid and the ask should be matching in both time and quantity, i.e. all the bid constraints should be satisfied using the existing state of the ask queue. If the bid price and the ask price are different, then the price used is the one of the oldest order. Although this idea is similar to that of a double auction, matters are considerably more complicated because we are dealing with perishable resources with a time dimension. Finally, the matching module must also periodically check for expired bids and asks, which should be removed. In Sect. 4 we present in detail the matching procedure that is to be executed when a new bid/ask is submitted, while in Sect. 5 and Sect. 6 we specify two matching algorithms that could be adopted.

4 Matching and Remainder Asks

The rationale of the matching procedure is to provide the required coverage of the bid with the cheapest matching asks (asks overlapping in time whose price is at most as high as that of the bid) by means of a matching algorithm. If a bid is matched *fully* then reservation of resources, accounting and computation of remainder asks that replace the original asks in the ask queue are performed and the bid is withdrawn from the bid queue and subsequently serviced. Though a bid is always fully matched (fully satisfied), this is not the case for asks. Therefore, in general a fraction of an ask may be used to (partly) match and serve a bid, thus generating a remainder ask.

Specifying the remainders in the futures (forward) market is much simpler than the spot market. Since both future bids and asks are fixed in time, the remainder is a valid ask and can remain in the futures market. This remainder in general corresponds to a non-rectangular shape, in the sense that the amount of VMs offered in not the same for the entire time interval spanned. Such a remainder ask can also be equivalently represented as a collection of at most three rectangular shapes. We henceforth adopt this representation, to clarify the presentation of the algorithms to follow. A related example is depicted in Fig. 3.

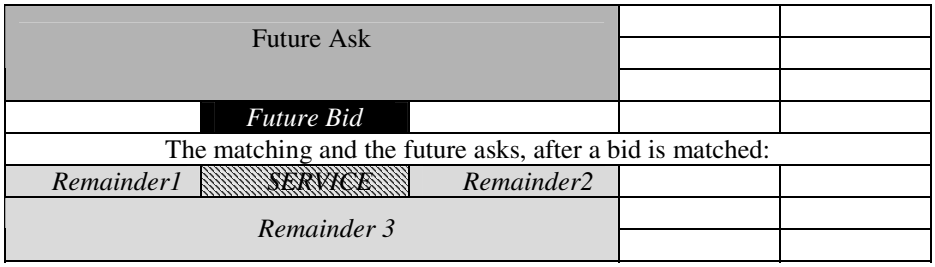

Fig. 3. Matching and remainder asks in the futures market

Things are more complicated for the spot asks, since some of the remainders generated may not be able to offer resource immediately, as opposed to others. The matching procedure and the respective remainders, which are considered as individual rectangles by our matching algorithm, are depicted in Fig. 4 and Fig. 5.

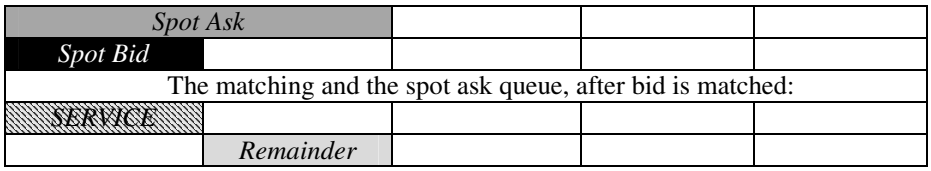

Fig. 4. Matching and remainder of a spot bid with an ask offering same number of VMs

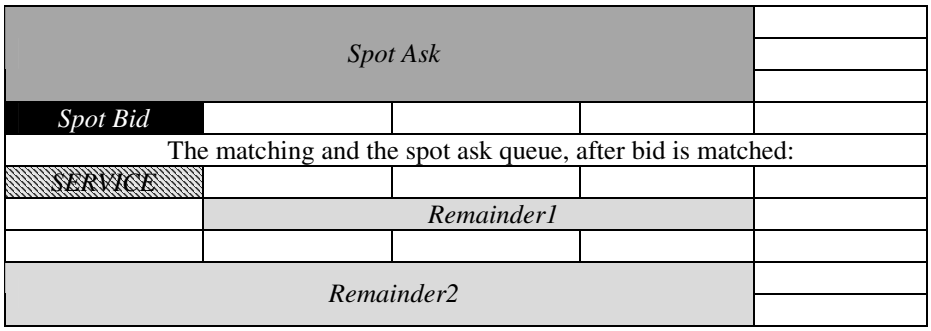

Fig. 5. Matching and remainder of a spot bid with an ask offering different number of VMs

An interesting issue here is that a matching of a spot bid with a much larger spot ask creates remainders that offer resources "as soon as possible" but not immediately, namely *Remainder* of Fig. 4 and *Remainder1* of Fig. 5. There are three options on the treatment of these remainders: a) transfer them to a "waiting queue" and reinsert them into the spot queue when the system time is such that they can indeed offer resources immediately, or b) cancel them and notify the provider, or c) treat all the remainders of a spot ask as valid spot asks which remain in the spot ask queue and are considered for matching, but tagged with additional constraints on when they can be used. Option a) is not economically sound because the market should be kept simple and refrain from making any "brokering" decisions on users' behalf. Indeed, the automated re-insertion of an ask after some time where the market conditions and prices may be completely different than those at the moment, would be confusing for providers who would face uncertainty regarding their strategy. Options b) and c) are both economically sound, with the first being the simplest one, yet resulting in overhead for the hardware providers. On the contrary, option c) does not suffer from this problem, yet it complicates significantly both the representation of asks since now a task description must also include the time slots where the machines of the ask are not free to be used by other bids, and the matching algorithm. In this paper, we investigate both approaches. In particular, option b) is the fundamental assumption for the matching algorithm of Sect. 5. Option c) and its implications on matching are investigated in the penultimate section of this paper, where the outline of a more sophisticated matching algorithm is also presented.

5 The Matching Algorithm

We begin the presentation by focusing on the forward market. For simplicity reasons, it suffices to adopt a one-pass of the queues matching algorithm. The matching algorithm in the futures market is much simpler than that of the spot queue, since the timespan of all bids and asks is fully specified, i.e. their start and end times are decided upon their submission and cannot be changed subsequently, as opposed to spot bids/asks. A meaningful matching procedure for the futures market is to try to match a bid with the cheapest matching asks.

The algorithm for the spot market has to make the same decision but in light of the feature that spot asks may start contributing resources to the matching at some later time, due to the flexibility associated with the provision of their resources. Note that we refrain from adopting a combinatorial approach due to the high computational complexity. The algorithm presented in the remainder of this section is in line with the sorting and treatment of the ask queue in terms of price and time of arrival (in case of equal price for two or more asks), despite the fact that it uses some temporary data structures with a different sorting. Its fundamental property is that if an ask is of lower price than another, then the latter cannot "steal" time of match of the former cheaper ask, i.e. an ask can influence only the quantity of resources that will be provided by higher price asks, as opposed to that of lower asks. This property is very important, since it ensures that the matching algorithm does not violate the rationale of the bid and ask spot market. Also, as mentioned in the previous section, it relies on the assumption that all the spot asks of the queue can offer their resources immediately. This implies that if a spot ask can offer service at some time *t*, then it can also provide service at any time *t'* prior to *t*.

The matching algorithm of this section examines how to cover a particular bid, and produces as the matching solution an ordered list of asks matching the bid in terms of price; the list is ordered with respect to the deadlines of the asks (i.e. latest time to start providing resources). This list would then be passed to the scheduler, who could serve the bid accordingly. However, the algorithm matches the bid with resources taken as much as possible from cheapest asks in the list, which are considered first. That is, the ordering of the list yields the order in time according to which the bid will be served by the various matching asks (or parts thereof).

This code is run from scratch every time a new matching is to be performed, either because a new bid arrived, or because a new ask arrived. Note that when we encounter an ask that could provide some service because its price does not exceed that of the bid we need to decide a) where to place it in the order of asks to serve this bid, b) how much of it to use. The solution we adopt is a) to order the matching asks according to the time constraints b) use as much as possible of cheapest asks. In particular, for any matching ask we use the part of it that does not render any of those asks invalid[1] by any part in time. Indeed if we use less of the specific ask considered than this part, we leave a part of service that could be provided unfulfilled. Yet, if we use a larger portion of it, then we would actually replace service that could be provided by cheaper asks. This would increase the customer's charge and violate the ordering and treatment of asks of the queue with respect to prices. An example of this matching algorithm is provided below:

[1] I.e. it does not cause any time deadline violation due to the "shifting" in time of the service start of the respective ask.

Assume that a spot bid is received, requesting 1 VM for 3 hours for a price of 4€/VM/hr. Assume that there are the following matching asks in the queue, which for simplicity are taken as offering each as many resources as those required by the bid: a) *Ask1*: Offer 1 VM for 1 hour, time deadline: now + 0,5 hour p: 1€/VM/hr. b) *Ask2*: Offer 1 VM for 1 hour, time deadline: now + 1 min p: 2€/VM/hr. c) *Ask3*: Offer 1 VM for 2 hours, time deadline: now + 6 hours p: 3.8€/VM/hr. Note that both for this example and throughout the paper, "now" denotes the start of the next time slot, due to the fact that in our model time is not continuous but discretized in slots.

The algorithm would initially partly match the bid with the **cheapest** ask of the queue, namely *Ask1*. Therefore, the outcome of the execution of the algorithm after examining the first ask in the queue is as follows:

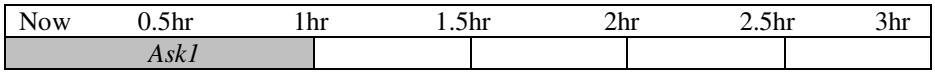

Fig. 6. The algorithm initially partly matches the bid with the cheapest ask

The algorithm would subsequently examine the second cheapest ask, namely *Ask2*. *Ask2* is inserted prior to *Ask1*, due to its shorter deadline. This means that *Ask1* would be shifted in time and then *Ask1* would violate its time constraint by 0.5hr.

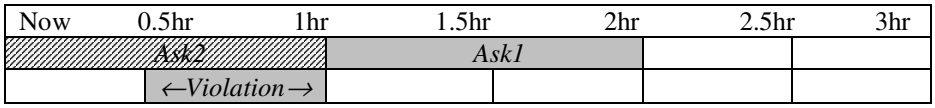

Fig. 7. Ask1 is shifted in time due to the selection of Ask2 as part of the matching solution

This time violation means that only 0.5 hr of service will be provided by *Ask2*, since an ask cannot influence the quantity of resources that will be utilized by any lower price ask, namely *Ask1* in this example. Since we can have in total 1.5 hour of service, our algorithm opts to get as much as possible from the cheapest provider. This means that *Ask2* will provide only 0.5 hr of service, as depicted below:

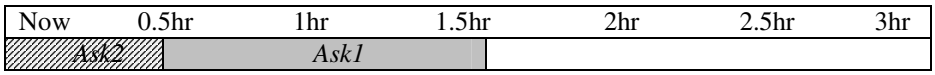

Fig. 8. A fraction of Ask2 is used for the matching since Ask1 must be fully used

Subsequently Ask3 is examined and it provides the remaining 1.5 hr of service. Therefore, this bid will be served as follows:

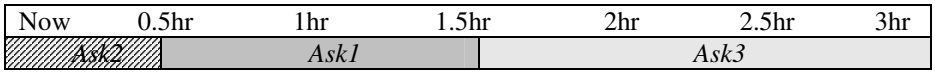

Fig. 9. The solution of the matching algorithm

It is easy to prove that the aforementioned algorithm clearly favors low price asks. By construction, if an ask is of lower price than another, then the latter cannot "steal" time of match of the former cheaper ask, although it can influence its position in the order of providing service to the bids. Therefore, this matching procedure provides nice incentives for providers to submit low price asks. Also, this procedure attempts to match a bid with a low-cost coverage of matching asks.

Yet, this algorithm fails either to always discover matching of a bid with the asks in the queue whenever such a matching is feasible, or guarantee that when it finds a match that this is the lowest-cost match of the bid. This algorithm does not guarantee these properties because its objective is to match the bid completely without violating the queue order. In fact, we have also developed an algorithm that always produces one matching whenever there does exist one. However, the latter is not economically sound because it violates the queue order principle of the bid and ask mechanism and can only serve as a benchmark in order to assess the ratio of matches that the present algorithm misses. To illustrate these shortcomings of the present algorithm, it suffices to modify *Ask2* as follows: Offer 1 VM for 6 hours, time deadline: now + 1 min p: 2€/VM/hr. It is now obvious that the lowest-cost matching for the bid is to match it for its entire duration with *Ask2*; this is depicted as Fig. 10. However, the matching algorithm still returns the same solution, which is depicted as Fig. 11.

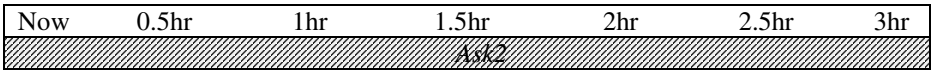

Fig. 10. The cheapest matching solution for the bid submitted

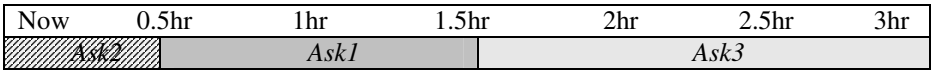

Fig. 11. The matching solution computed by the algorithm

Note that the solution that the matching procedure provides is in fact more expensive, due to the much higher cost of *Ask3*. It is also worth noting that if *Ask3* were absent from the queue, the algorithm would not find the matching with *Ask2* and the bidder would not be served, although this is actually feasible.

Last but not least, we remind the reader that this algorithm relies on the assumption that all spot asks can offer their resources immediately; (remainder) spot asks which could offer resources from some time in the future have been removed from the queue and their providers have been notified accordingly. In addition to the overhead for the hardware providers, the fact that this algorithm works with a subset of the spot asks that could be used for matching bids, further limits the number of matches computed. This is in contrast with the algorithm outlined in the next section, which also favors low price asks and also treats all the remainders of a spot ask as one non-rectangular spot ask which remains in the spot ask queue and is considered for matching.

6 Extensions and Future Work

As opposed to the algorithm of the precious section, we proceed to outline an algorithm, which considers spot asks whose resources are not necessarily available from the current system time (such asks are the *Remainder* of Fig. 4 and *Remainder1* of Fig. 5) as candidate matches. In order to allow such asks in the spot queue, we need to generalize the definition of spot asks (see Sect. 3) so as to be the asks that prescribe that a certain quantity of resources (e.g. 1 VM) for a certain duration (e.g. 2 hours) is made available as soon as possible within certain time intervals (e.g. from 13:00 till 20:00 today, except the intervals [14:00-15:00] and [16:00-17:00] where this VM has already being previously reserved to service some bid) and the ask is valid and present in the queue up to a maximum time deadline, e.g. 18:00 today. Note that this ask is still a spot ask since the starting and ending times are not fixed instants in the future, as opposed to futures.

Note also that this spot ask is different at different times, due to the fact that prior reservations that keep the resources busy are fixed in time. Therefore, the matching of a bid with a set of such asks that are also changing in time is more complicated, in the sense that the algorithm should first specify the current state of the ask. In particular, solving this scheduling problem is a well-known NP-complete problem. Due to the problem's high complexity, we outline an algorithm which is fast enough to be adopted in a realistic market, performs well in terms of the matches computed and does not violate the fundamental rationale of the spot ask queue, i.e. prioritization of cheap asks. Nevertheless, this algorithm is a heuristic approach that does not claim to solve the scheduling problem, i.e. it cannot always compute a set of matching asks for a bid if there is indeed one. Its formal definition and assessment are beyond the scope of this paper and are left for future work. However, the rationale of the algorithm is presented below.

The algorithm initially computes the candidate matches for the ask (i.e. asks of the demanded quantity of VMs) that can offer service from time *Now* (denoting the start of the next time slot) and for a service duration equal to that specified in the bid. This is performed by means of creating a matrix. Such an example matrix is depicted as Fig. 12. Each column of the matrix corresponds to a slot of the time interval where service will be provided. Each row corresponds to a provider that can offer service within this time interval, with the cheapest being on the top row. The cells where each provider can offer service are marked, as well as the total availability of each provider's (i.e. number of slots where they can offer the desired amount of VMs). For instance, Provider2 in Fig. 12 can offer three hours of computation anywhere within the 4-hour time interval, i.e. provider's availability is 3. If there is a slot where it is not possible to provide service for any provider, then the algorithm fails and proceeds to find a match for the time interval [Now + 1 slot, Now + 1 slot + service duration]. The algorithm then detects the slots where there is only one provider offering service; these providers are matched for those slots and their total availability for service is subsequently reduced. Then the algorithm attempts to fill the slots where there are multiple candidate providers, regardless of their total availability: For these slots, the algorithm attempts to do a probabilistic matching. In particular, the algorithm starts with the cheapest ask and according to the provider's availability randomly fills some slots, so that the provider's availability becomes zero. I.e. the cheapest ask is fully

Now	+1hr	+2hr	+3hr	+4hr	Availability
Unavailable	Provider1	Provider1	Provider1		2
Provider2	Provider2	Provider2	Provider2		3
Provider3	Provider3	Unavailable	Provider3		3

Fig. 12. The matrix used from Algorithm 2 to match a spot bid demanding 4 hours of service

utilized. It then proceeds with the next cheapest ask and does the same. Note however that after the second step, there might be slots allocated to two candidate providers. For these slots, each provider is assigned a probability of moving from this slot, depending on his total availability. A dice is thrown and a provider is moved to an empty slot according to a transition probability, which is larger for slots where the number of providers that could serve this slot is small. The algorithm terminates when all the slots are assigned to some provider and thus a match is found. In case there are slots where there is no provider serving it, while there are not any slots with more than one provider, the algorithm has failed to compute a match. Due to the matching algorithm's probabilistic nature, it can be repeated for a maximum prespecified number of times until it computes a match. If it fails, then it attempts to compute a match at a next time window, i.e. at the second time for the time interval [Now + 1 slot, Now + 1 slot + service duration]. This is performed until a match is indeed found or the algorithm fails for the entire duration where the bid is valid.

Note that for some services, e.g. non-parallel distributed applications, such as a company's web server, it might be meaningful to enforce *horizontal atomicity* instead of *vertical atomicity*. This means that the user should be assigned a provider's VM for the entire duration of service demanded. However, multiple providers may offer the total number of VMs requested. If this is indeed the case, the matching algorithm is greatly simplified. The reason is that under this assumption, candidate asks are only the asks of providers that can offer VMs for the entire duration of service demanded by the bid. Therefore, the algorithm sorts the asks providing a VM for the entire duration within each time interval, starting from [Now, Now + service duration]. If the number of matching asks in this interval is at least that demanded, then the cheapest VMs are selected and provided as match. If the number of matching asks is less, there is no possible match and the algorithm proceeds to compute a match at a next time window, i.e. at the second time for the time interval [Now + 1 slot, Now + 1 slot + service duration]. This is performed until a match is indeed found or the algorithm fails for the entire duration where the bid is valid. It is trivial to prove that this algorithm never fails to detect a match if any and that it also always computes the cheapest matching ask that can be provided as soon as possible to the user.

Throughout the paper we have assumed that customers are interested in rate of computation in a certain time interval. Replacing the "rectangles" of this market with a total quantity of computation greatly simplifies the matching algorithms presented earlier applicable for this market as well. Thus, instead of trying to match a bid with a rectangle constructed by a set of asks with proper height, the matching algorithm simply picks the cheapest asks that can provide the desired computation within the specified deadline.

As already mentioned it is possible that a bid be satisfied by the asks submitted by multiple providers. This clearly increases the switching costs of users and reduces the value of the allocations of the market. Therefore, this problem should be mitigated by means of a special algorithm. Such an algorithm could prescribe that units allocated to different users should be "swapped" if possible, thus resulting in a less fragmented with respect to number of providers per user, outcome. It is worth emphasizing that though units of allocation can be swapped between consumers, prices and quantities are not. A preliminary idea for such an algorithm is to swap units between two users, if and only if for some performance index (e.g. total number of different asks matched) the post-swap value is better for one user while being non-worse for the other. The formal definition of such an algorithm, as well as conducting simulations for the evaluation of the algorithms presented in this paper, is left for future work.

7 Conclusions

In this paper, we have specified a market where hardware providers can interact with users interested in leasing Grid resources for a price and a time period. Our approach comprises a stock-market like mechanism that enables the trading of computational power on the basis of a spot and a futures market. The spot market comprises a pair of bid and ask queues. This grid market is more complicated than the standard spot/futures markets of storable commodities, because the computational service traded in our case comprises of resources that are perishable, and has both quantity and duration specified in terms of a time interval. This is an important feature of our market mechanism that has been taken into account by both the market mechanism and the related matching algorithms that operate on the spot bid/ask queues and futures market. Finally, we have briefly addressed the issue of post-sale optimization in order to mitigate the switching cost of consumers being served by multiple providers over time. The formal definition of such an algorithm is left for future work. Another direction for future research is to formally specify and evaluate the algorithm that provides matchings according to which service can start with a delay, due to the fact that remainder asks that do not provide readily available resources are employed.

Acknowledgments. This work has been supported in part by the European Commission within the Framework Programm FP7, ICT, through the project GridEcon. The authors would like to thank all consortium members for useful discussions on the subject of this paper.

References

1. Buyya, R., Abramson, D., Venugopal, S.: The Grid Economy. Proceedings of the IEEE 93(3), 698–714 (2005)
2. The Gridecon Project Consortium: Deliverable D2.1. Economic Modelling Requirement Report (2008), http://www.gridecon.eu/html/deliverables.shtml
3. Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic Models for Resource Management and Scheduling in Grid Computing. The Journal of Concurrency and Computation Practice and Experience (CCPE) (2002)

4. Denton, M.J., Rassenti, S.J., Smith, V.L.: Spot market mechanism design and competitivity issues in electric power. In: Proceedings of the Thirty First HICS
5. Lien, D., Quirk, J.: Measuring the Benefits from Futures Markets: Conceptual Issues. International Journal of Business and Economics 1(1), 53–58 (2002)
6. http://www.amazon.com/gp/browse.html?node=201590011
7. http://www.sun.com/service/sungrid/index.jsp
8. http://h20338.www2.hp.com/enterprise/cache/250417-0-0-225-121.html