Ann. Telecommun. (2010) 65:653–667
DOI 10.1007/s12243-010-0183-1

A marketplace and its market mechanism for trading commoditized computing resources

Jörn Altmann · Costas Courcoubetis · Marcel Risch

Received: 7 July 2009 / Accepted: 28 May 2010 / Published online: 10 June 2010
© Institut Télécom and Springer-Verlag 2010

Abstract This paper presents the design and implementation of the GridEcon Marketplace. In addition to supporting a market mechanism for trading computing resources on a pay-per-use basis, this marketplace also provides an environment for integrating value-added support services. These value-added services help consumers to use the utility computing market more efficiently. The GridEcon Market Mechanism for virtual machines specifies in detail the unit-of-trade, the bids and asks, as well as the matching algorithm. The marketplace and market mechanism are validated by using the GridEcon Platform, which is a service-oriented platform for composing market scenarios. Our validation results show that the GridEcon Marketplace fulfills all functional requirements and that the GridEcon Market Mechanism is computationally and economically efficient.

Keywords Grid economics · Cloud computing · Computing resource market · Market mechanism design · Utility computing · Grid computing · Simulation · Market scenario emulation

J. Altmann (✉) · M. Risch
Technology Management, Economics and Policy Program;
Department of Industrial Engineering,
College of Engineering, Seoul National University,
599 Gwanak-Ro, Gwanak-Gu,
Seoul 151-744, South Korea
e-mail: jorn.altmann@acm.org

M. Risch
e-mail: marcel.risch@temep.snu.ac.kr

C. Courcoubetis
Department of Computer Science,
Athens University of Economics and Business,
47A Evelpidon Str,
Athens 11362, Greece
e-mail: courcou@aueb.gr

1 Introduction

In recent years, many Grid platforms and Cloud computing solutions have been proposed. However, there has only been a lower-than-expected commercial uptake of these technologies. Only a handful of companies (e.g., Amazon) have entered the market successfully [1]. Nevertheless, even those companies provide basic services only. For example, Amazon's EC2 Cloud offering is a very simple platform, which does not provide users with an environment for combining and sharing computing resources on the fly, as originally envisioned by the Grid community. The EC2 offering simply follows a B2C-like approach, which allows a single provider to sell computing services to consumers.

By analyzing the incentives of users to access the Cloud and by investigating the structure of computing resource markets, four major reasons for the limited uptake of Cloud computing (note, in this paper, we use the terms "Grid computing", "utility computing", and "Cloud computing" interchangeably) beyond enterprise boundaries can be identified: (a) The risk of relying on the availability of external resources for running a business is perceived to be high [6]; (b) the required changes for integrating Cloud resources into the existing IT infrastructure are costly; (c) usage-based pricing schemes introduce a large uncertainty about the total charge for a resource; and (d) sustainable business models for Cloud resource provisioning do not exist.

The analysis of these issues has indicated that a solution (for achieving an uptake of Cloud technology in the next generation Internet) could be a market for trading computing resources. Such a market could reduce the risk of using external resources by becoming a trusted entity that mitigates the risk of using resources of unknown providers.

The market could also provide support services in order to help users to integrate Cloud computing services smoothly into their existing IT infrastructure. These support services could additionally help users to predict the usage-based charges incurred in the Cloud accurately and, last but not least, provide vital trading support for consumers.

Although many marketplaces and marketplace environments for commercial Grid computing have been suggested in recent years, none of these approaches has focused on the need for supporting the right types of services in such environments. In most approaches, computing power has been treated either as a traditional economic non-perishable good (i.e., no special attributes like duration and urgency were considered) or as a job scheduling problem (i.e., the whole system was considered to behave as a centrally controlled queue of jobs) [20].

Due to these shortcomings, the lack of an analysis of support services, and the lack of an understanding of their key economic parameters, the GridEcon project aimed at identifying the functionality of services required for the efficient operation of a computing resource market [4]. These services are value-added services, complementary to the market mechanism used. They require the careful definition of key economic attributes of purely computational services.

The result of the analysis led to the design of the GridEcon Marketplace and the GridEcon Market Mechanism for commoditized computing resources, i.e., standardized virtual machines (VM). The market mechanism for trading computing resources is an extension of a classic double auction design. The extension allows for extra attributes in the auction offers to include duration (i.e., a time interval), price, urgency (i.e., the point in time when the computing resource is needed), and maximum validity time of the offer. Matching of buy and sell offers requires not just a one-to-one but a many-to-many approach. The design of the mechanism took also into account issues like scalability and not favoring large providers of computation. An economic analysis of the equilibrium properties of the spot market that the GridEcon mechanism would generate can be found in [40].

The remainder of the paper is structured as follows: In Section 2, we provide an overview of the state-of-the-art in market mechanisms, market environments, and Cloud computing resource markets. Section 3 specifies the GridEcon Marketplace and the GridEcon Market Mechanism. The specification comprises the prerequisites of a market to be successful, the market environment, the unit-of-trade, the bids and asks, as well as the matching algorithm. After introducing briefly the GridEcon Platform that is being used for validating the GridEcon Marketplace and its market mechanism, Section 4 describes the validation scenarios and measurement results. Besides, Section 4 presents the simulation results and the discussion of the results. The paper concludes with Section 5.

2 State-of-the-art in computing resource markets

2.1 Market environment

Only a few Grid markets (e.g., Popcorn Market [8, 9], the Spawn system [11], the Grid Architecture for Computational Economy (GRACE) [2], and Tycoon [5]) have been proposed. They differ in the form of value-added services offered to consumers and the openness of the implemented systems.

GRACE is the most sophisticated among the Grid markets. It offers the consumer many services, e.g., the Grid Resource Broker. However, GRACE is a monolithic application, which would leave consumers without a choice but to use these services. Therefore, GRACE could use its market power to set prices for maximizing profit (e.g., personalized pricing), to lock in consumers to this system, and to making this approach less attractive.

The agents proposed by the Tycoon market attempt to perform many tasks for the consumer, such as capacity planning, portfolio management, and risk management. However, these agents cannot be substituted. Once a user has decided on an agent, the user has to use this agent for all his support service needs.

With respect to the software components developed by GRACE and Tycoon, it seems that they were built with a specific business model in mind. In the case of GRACE, the broker is an entity, which shields the consumer from the Grid market. It should ensure that the consumer is not aware of the fact that the Grid is a distributed, non-homogeneous environment. In the case of Tycoon, the agents should perform all the tasks that the consumer cannot perform on his own, such as resource discovery and ensuring resource availability.

Some additional research on Grid Markets, in particular on a Grid resource marketplace, comes from Pattanaik et al. [12]. The authors pay close attention to the fact that many resource markets are one-sided, i.e., that the providers have to volunteer resources and the customers then have the option to choose which services they wish to purchase. While the points raised in this paper are valid, the authors tacitly assume that resource consumers have sufficient expertise to evaluate the different offers. The idea that some consumers may require additional support services has not been included.

In addition, some work on Grid market services has been performed by the Gridbus [13] project. It has been the only project that has introduced a number of services for the user (e.g., a Workflow Engine [14], a Nimrod-G Broker [15],

and a Gridbus Data Broker [16]). While these technical services are useful for consumers with respect to executing an application on the Grid, they do not provide support for the economic-related issues of marketplaces. For example, these services do not answer questions as which resources to purchase and at what price they should be purchased. Furthermore, the Gridbus environment assumes knowledge on how to trade on a market.

The Globus project has also taken a look at the development of Grid services [35]. This work has been executed under the OGSA umbrella [36]. It has mainly focused on services which would make the Grid run more smoothly. The two major focuses were the development of an accounting system and the development of data access methods within a Grid environment [37–39]. However, these services do not take into account that users may require some help when working in a Cloud or Grid environment.

In addition to this, all these projects (Popcorn Market [8], Spawn system [11], GRACE [2], and Tycoon [5]) assume a very abstract model of a resource consumer. It seems that it was tacitly understood that consumers had already accepted the market environment, learned to deal with the risks involved, participated in the market, and had sufficient knowledge to use the market. The GridEcon project is different in this regard, since the project is not only focused on market-based resource allocation but also on value-added services that support users around the marketplace [4].

The GridEcon project did not implement an implicit business model with certain functionalities as other projects did [4]. Instead, it analyzed basic services that could support consumers in a market environment. Since these business-related, basic services can be provided by different entities, those entities may choose to bundle some services while others may offer them as stand-alone services, depending on the business model of those entities. This approach is similar to many other markets, which have support services for consumers. Examples of such support services include stock brokers in stock exchanges (which perform trades on behalf of the banks' customers), travel agents, and travel search Web sites (e.g., Expedia.com [3], which help customers to book their trips).

2.2 Existing commercial Cloud offers

In recent years, a large number of commercial Cloud providers have entered the utility computing market, using virtualization. Today, there are a number of different types of services, being sold under the label of “Cloud Computing”. Three types can be distinguished. Firstly, there are resource providers, such as Amazon and Tsunamic Technologies [1, 28], who provide computing resources (e.g., EC2 [1]). Secondly, there are providers, who not only sell their own computing resources but also their own software services. Examples of those are Google Apps and Salesforce.com [29, 30]. Finally, there are companies that attempt to run a mixed approach, i.e., they allow users to create their own software services and, at the same time, offer users various support services (i.e., platform services). An example of such an approach is the Sun N1 Grid or force.com [30, 31].

In addition to this, there are providers, who sell Cloud storage resources (e.g., EMC Atmos [41]). However, these specialized services are still in their infancy and have not yet taken hold in the commercial Cloud computing environment.

All these Cloud resource providers currently operate outside a common market. Their offers are not easily comparable and the fact that some, like Amazon's EC2, use proprietary VM interfaces means that it is difficult to move VMs from one provider to the next. At the same time, none of these providers offer any additional services to their customers, expecting them to know how to use Cloud resources effectively. Furthermore, many of these providers (with the exception of Amazon and EMC) offer only a single resource type, thereby limiting customers' options severely.

2.3 Open Cloud market enablers

In addition to the introduction of virtualization in data centers, several companies now offer a platform to integrate in-house resources with externally purchased Cloud resources. One such company is Enomaly [32]. Enomaly offers a product that not only allows users to turn their data center into a Cloud but also allows users to act as providers for computing resource services. Using such software, data center operators could easily be encouraged to participate in an open Cloud market and sell their excess capacity at times of low demand while purchasing Cloud resources when demand is high.

A second company that provides a similar service is Zimory [33]. Its product turns a regular data center into an intra-company Cloud and allowing an efficient use through sharing of resources. However, Zimory takes this idea one step further and allows its customers to sell spare capacity via its own marketplace. During times of high demand, its customers can purchase Cloud purchases via the Zimory Cloud. Since all these providers and consumers have to use Zimory's software package, the integration of Cloud and in-house resources is fairly straight forward. However, any participant faces a number of challenges. Zimory only supports two virtualization engines [34], which means that customers using different virtualization engines either have to replace their virtualization infrastructure or they have to

wait until their virtualization engine is supported. This slows down the growth of Zimory's customer base. Consequently, as long as Zimory has a few customers only, the number of available resources on the market will be limited, making the market illiquid. Besides, since there is no market mechanism specified, sellers have to accept the pricing of Zimory, which can be detrimental if not performed correctly.

These Cloud enablers are all concerned with the inclusion of Cloud resources into an in-house data center management system. This represents a major facilitator for Cloud usage, since these tools allow the consumer to treat all resources as if they were hosted under a single roof. They represent a first step in the move towards the development of services for Cloud users.

The fact that Zimory and Amazon offer a marketplace for its customers shows that the need for more efficient usage of computing resources is a goal for many companies. However, Zimory neglected the development of support services, which would make the computing resource market more attractive to users who have little experience in such matters.

2.4 Market mechanisms

Since price setting (i.e., market mechanism) is a central point of any market, it is also fundamental for providing incentives to users to trade their computing resources [7]. During the recent years, many market mechanisms have been proposed, aiming at economically efficient resource allocations. The allocations take into account the utilities of providers and resource consumers [5, 22–24, 26]. In these works, Grid consumers want to minimize expenses, whereas Grid providers want to maximize their return on investment or reduce their operating costs [10].

Weng et al. focus on the determination of prices for resources of a Grid marketplace [22]. They introduce a method for setting the price of differentiated goods based on a double auction mechanism.

Sandholm et al. address the goal of predicting bids in order to meet the time deadline for a specific job with a given budget [23]. The authors present three high-level prediction techniques and discuss the theory behind these techniques.

Wolski et al. consider a Grid environment that offers two kinds of resources, CPU and hard disk, being complementary to each other [24]. Each producer offers a single resource to the Grid.

Lai et al. introduce the Tycoon market, which incorporates auctions into the resource allocation system [5]. A proportional share-based scheme is introduced to assign the resources according to the bids of the various bidders. The authors also show how the user (agent) can calculate the optimal bid, given a set of available machines and the bids of the other players on each machine.

In all the aforementioned approaches the product concerns computing power, i.e., CPU cycles. In these CPU markets, mostly bag-of-task jobs are assigned to single CPUs. However, since the majority of complex tasks are of workflow nature, these solutions are limited. In addition, these approaches cannot consider urgency and time limits of user requirements. They also do not consider future markets for trading resources that are needed in the future.

3 The GridEcon Marketplace for trading computing resources

This section describes the overall framework of the GridEcon Marketplace approach as well as the details of its market mechanism.

3.1 Requirements

3.1.1 Basic prerequisites for a functioning marketplace for computing resources

Based on the economic prerequisites for marketplaces in general, it can be derived that a successful marketplace for computing resources has to fulfill at least one of the following economic prerequisites:

- The pattern of individual demand of a Cloud user for computing resources shows variability, i.e., it is sporadic;
- There is uncertainty about the demand for computing resources;
- The units of computing power that are needed are smaller than the purchase of a computer could provide.

In addition to these economic prerequisites, two technical prerequisites must be fulfilled, i.e., adequate technology for implementing Cloud computing must be available. Firstly, most importantly, standardized interfaces for services must have been defined so that services developed by different providers can communicate. Secondly, technology for commoditizing computing resources must be easily usable.

While these service interfaces still need to be agreed upon in the current Cloud market, virtualization software is available for use already [25].

3.1.2 Dependencies of a marketplace for computing resources

A major requirement for a functioning marketplace is that there is liquidity in the market, i.e. there is high demand for and supply of computing resources so that market participants can easily trade their goods. In order to enable that,

commodity computing resources have to be designed. The advantage of commodity computing resources is that it bundles demand, which would otherwise be split across a very large number of different types of computing resources. Looking at the success of Amazon EC2 [27], we can state that commodity computing resources are certainly desirable for users.

For a marketplace for computing resources to be useful, there must be a sufficient supply of excess computing resources available. This condition is definitely fulfilled as well, since there is a large amount of underutilized computing resources at enterprises. Currently, only a few large providers are selling their excess computing resources (e.g., Amazon, Microsoft).

3.1.3 Requirements specific to a computing resource marketplace

For a computing resource marketplace to be accepted by users, it is necessary that the marketplace service meets the following requirements specific to computing resources:

- In order to assure quality of the goods offered, computing resource offers should be monitored. Therefore, it can implicitly be assumed that all the goods of the market are of the same quality;
- In order to guarantee security, customers using the same physical machine must not be able to access the other customers resources and data;
- In order to find acceptance of the marketplace among customers, the marketplace service should provide access to computing resources in a transparent and simple way;
- In order to ensure that buyers and sellers do not make the trading transaction directly, the anonymity of sellers and buyers is required.

3.1.4 Operation of the marketplace

In order to bootstrap the marketplace service successfully, we suggest that the marketplace service owns computing resources, which could be sold in case of limited computing resource sales offers. This way, potential buyers would not experience no supply in computing resources, which, otherwise, would make them leave the marketplace for good.

It should also be possible to have a limited number of instances of the market for different good quality (e.g., gold, silver, bronze). This is also in accordance with the practice of Amazon EC2. Therefore, the marketplace service should provide an environment for trading different computing resources.

After reciprocal offers (i.e., offers for purchasing and for selling computing resources) are matched within the marketplace, the task of the marketplace is the timely allocation of a resource from the provider to the buyer, commencement of the execution of deployed applications, and finally, termination of the execution of deployed applications in a timely manner.

3.2 The GridEcon environment

The GridEcon Marketplace environment is shown in Fig. 1, consisting of three tiers: the Marketplace Tier (i.e., the GridEcon Computing Resource Marketplace), the Service Provider Tier, and the Economics-Aware, Value-Added Services Tier. Each of these tiers is designed according to SOA principles.

The Marketplace Tier encompasses the main trading facility. The trading facility is flexible enough to allow trading based on any kind of market mechanism such as auctions, bargaining, or posted prices.

The Service Provider Tier contains all providers of computing resource services, which can be traded in the marketplace. No further distinction is made with respect to the type of hardware service.

The Economics-Aware, Value-Added Service Tier contains five services, which have been identified as being important for the functioning and acceptance of a marketplace for commodity computing resources. The five services allow a consumer to determine which resources

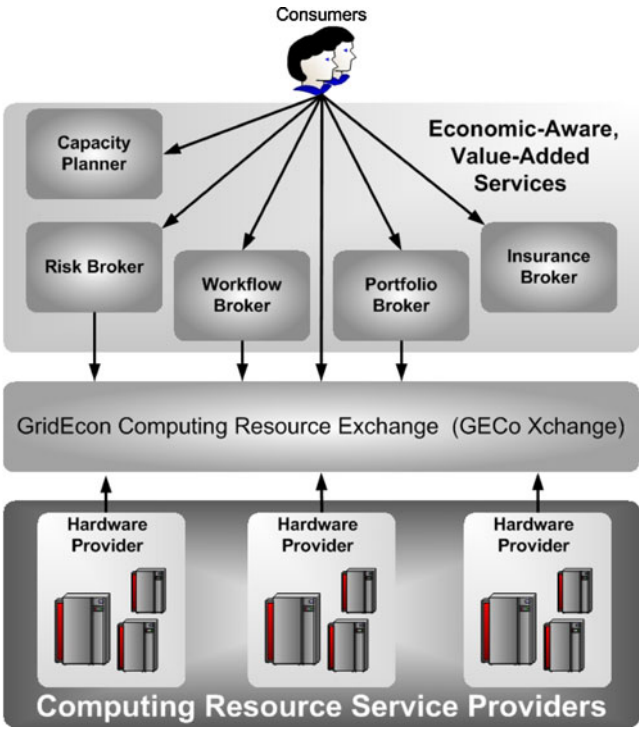

Fig. 1 The GridEcon Marketplace Environment

are required, which resources should be purchased, and how applications should be mapped to the resources. Those services are a Capacity Planning Service, a Fixed Price Quotation Broker, a Workflow Broker [19], a Portfolio Broker, and an Insurance Broker. Since these services are described in the analysis section of this paper (Section 4.1.2), we do not explain them here in detail.

3.3 User interaction with the marketplace environment

All interactions between the user and the market environment occur through a Web Interface. Figure 2 illustrates the interactions of a consumer with the GridEcon marketplace environment. It depicts seven steps, starting from determining the resource requirements to interacting with the deployed application.

In particular, for executing the first three steps, the consumer may consult the Capacity Planning Service [17]. It can help determining the resource requirements, the optimum price, and the best course of action. That means, the consumer actually decides how many computing resources are needed to cover his needs. At step 4, the consumer either bids for computing resources in the market directly or uses a workflow broker that performs the actions on behalf of the consumer. In case of a successful bid for computing resources, the user may opt to also purchase an insurance contract for the resources (step 5). After this step, the user has at his disposal a set of machines in the Cloud environment that he can use to deploy (step 6) and run his applications (step 7). The deployment of the application can also be supported by the workflow broker. Finally, the user can interact with his application (step 8).

3.4 GridEcon market mechanism

The core of the marketplace is the market mechanism. The market mechanism allows a resource provider to register with the marketplace an unused computing resource for sale (i.e., submit an 'ask'). It also allows a resource buyer to register the intent to buy a resource that is for sale within the marketplace (i.e., submit a 'bid'). In the following three subsections, the GridEcon Market Mechanism is described in detail. In particular, it explains the design of the offers (i.e., bids and asks), how bids and asks are matched in a timely manner, and the general format of the specification of a unit of trade.

3.4.1 Unit of trade

The format of the specification of a unit of trade (i.e., the good) is defined through the following three attributes:

- Start Time: The start time is defined as the time, at which the resource is available at the provider, or at which the resource is required by the buyer.
- Unit Duration: The Unit Duration defines a standard time period that the resource will be made available at least by the provider or the duration that the resource is required at least by the buyer. The Unit Duration is set based on the acceptance of users within the market-

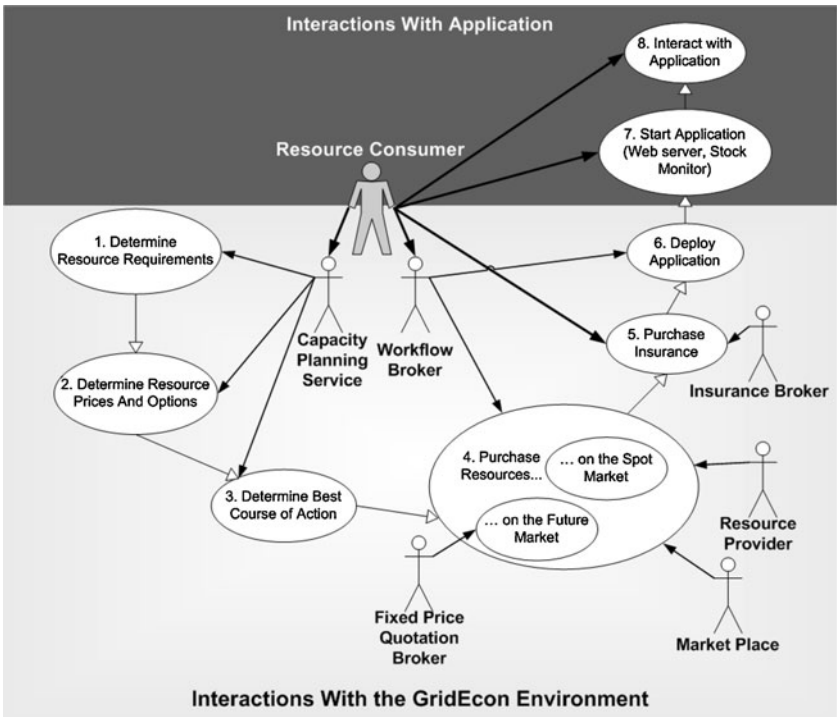

Fig. 2 User interaction with the GridEcon Market Environment

place. For the remaining of the paper, we consider one hour as the only standard unit duration available in the marketplace.

- Resource Type: It defines the type of resource offered by the provider or required by the buyer. In detail, the resource type specifies a virtual machine (VM) that is defined through the processor type, the main memory, the size of the hard drive, and the network bandwidth available (i.e., through the quality of the different resources). If two units of trade differ in any of these attributes, then the two units of trade are different, i.e., the quality differs.

3.4.2 Bids and asks

Based on the unit of trade, the bids and asks can be defined. A bid is submitted by an entity that is in need of one or more units of trade. An ask is submitted by an entity that owns one or more units of trade that it would like to make available to others. Beside the unit of trade, the bid and ask also comprise the following additional attributes:

- Expiration Time: This attribute indicates the time when the bid (or ask) will be removed from the system if not 'matched' with a reciprocal ask (or bid);
- Price: This attribute sets the minimum price that a provider will accept or the maximum price a buyer is willing to pay for a unit of trade. The price is expressed in €/unit-of-trade;
- Number of Resources: This attribute sets the number of units of trade, i.e. the number of resources of the specified type of virtual machine.
- Duration: The duration is set to a multiple of the Unit Duration.

Besides these attributes, a bid also specifies the Application ID attribute:

- Application ID: This attribute specifies the identifier of the application that is supposed to be deployed on a provider's resource at the Start Time.

Figure 3 shows an example of a bid for three virtual machines (b, c, d), starting at 11:00 for a duration of 2 h.

3.4.3 Matching algorithm

The rationale behind the matching algorithm can be summarized as follows:

- If satisfied, then bids are always completely satisfied, i.e., there are never remainder bids (a remainder bid is a piece of the same bid that is still pending). This is not the case for asks. They can be partly matched in order to serve bids.

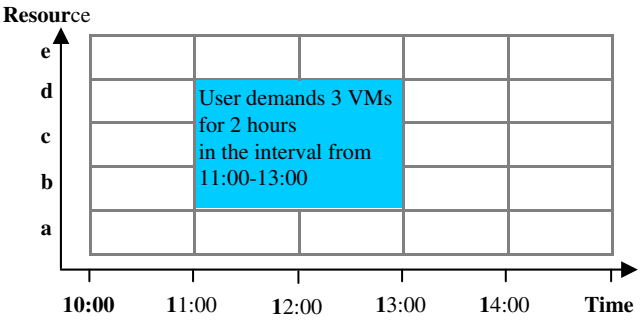

Fig. 3 Example of an order

- To satisfy a bid, it can be served by multiple providers. However, in order to ensure that the demanded VMs are allocated throughout the service duration, so that the user does not have to switch between VMs over time, it is mandatory that (*bid*) *horizontal atomicity* holds. This property is well-suited for applications such as Web servers.
- The matching algorithm considers as candidate matches of a bid, whose price is p_{bid} , only those asks, whose price p_{ask} is such that $p_{\text{bid}} \geq p_{\text{ask}}$. If several asks would could serve a bid, then the ask with the smallest price is selected. Besides, we omit examining higher-priced asks (i.e., $p_{\text{bid}} < p_{\text{ask}}$) and trying combining them with lower-priced asks, even if such combinations could in fact serve the bid and the bidder attains positive net benefit from the overall price of the service. If the bid price is higher than the ask price, the average price of both orders is used as the matching price.
- The matching algorithm periodically checks for expired bids and asks. It removes those from the respective queue.
- The matching module assumes that the owners of expired bids or asks are notified of the removal of their bids or asks.
- The matching algorithm is always executed when a new bid or new ask is submitted.

The matching algorithm is simple, since the time span of all bids and asks is uniquely defined. The start time and the end time of bids/asks are decided upon their submission.

The task of the matching algorithm with respect to bids (which demand a fixed number of VMs (denoted as M) throughout a certain time interval) is to return the M cheapest asks (if any) that are idle in the demanded time interval. Assuming that time is split into time slots, the pseudo code of the matching algorithm can be expressed as shown in Fig. 4.

If a new ask arrives, then the ask is entered into the ask queue and the bid queue is searched for bids, whose price is greater than or equal to that of the new ask. Afterwards, the Bid Matching Algorithm (Fig. 4) is executed for all those bids found.

```
If a new Bid requesting M machines in [t_from, t_end] arrives
Then {
    serviceFound = true;
    }
Foreach slot j of Bid /* Start from slot 1,… , to slot N of Bid */
    If (exists(Ask_i) && isOverlappingInTime(Bid,Ask_i) &&
    (priceOf(Bid)>=priceOf(ask_i)))
    Then {
        serviceBid(slot_j,Cheapest(ask_i)); /* find the cheapest ask
        providing service in this slot and serve it there */
        updateServiceCost();
        }
    Else {
        serviceFound = false;
        break;
        }
If (serviceFound)
Then {
    serveBidWithMatchingAsks(); /* the original asks partly
    matched are removed from the directory; owners of asks are
    informed that remainder has been removed from queue */
    performReservationsAndAccounting();
    }
```

Fig. 4 Bid Matching Algorithm

4 Validation

For validating the economic models that have been introduced in Section 3, we use the GridEcon Platform [21], which has been developed within the GridEcon project [4]. Besides providing an overview about the Platform, we describe the actual implementation of the Marketplace and present the measurement results.

4.1 The GridEcon platform

4.1.1 Architecture and implementation

Since the GridEcon Platform follows a service-oriented approach, all platform services are independent of each other. In order to design a market environment according to a user's specification, the GridEcon Platform allows its user to configure the interaction of the services. In particular, this is supported by the Workflow Engine. By feeding the Workflow Engine with a workflow, which describes the interaction between the different economic-aware services, the Workflow Engine orchestrates the communication between the platform services. Once the appropriate interactions have been orchestrated, it ensures that all communications between the required services are executed as defined in the workflow.

For the implementation of the GridEcon Platform, JavaEE was used. Apache Tomcat 6.x was used for the deployment and MySQL Server for the database layer. In addition, the following libraries and frameworks were employed: Spring, Hibernate, Axis2, and Quartz. For the client-side services, the ExtJS and jQuery frameworks were utilized.

4.1.2 Platform services

The GridEcon Platform Services can be divided into three groups: Economic-Aware Services, Middleware Services, and Auxiliary Services.

The Economic-Aware Services comprise an Insurance Broker Service, a Fixed-Price-Quotation Broker Service, a Capacity Planner Service [17], a Workflow Broker, and a Marketplace Service. The Insurance Broker Service takes the details of a proposed contract and calculates the risk of a contract breach. Based on the results of the calculation and the coverage type, it calculates a premium. The user has to pay this premium, if he opted for insurance coverage.

The Fixed-Price-Quotation Broker Service offers a fixed price contract to a user, who wants to use a specific computing capacity on a future date. However, since it is assumed that the broker does not own any resources, the broker has to buy them at the marketplace.

The Capacity Planner Service offers decision support, helping the user to determine an economically sound resource trading plan. This can include the purchase of Cloud resources, the purchase of in-house resources, and even the sale of resources on the marketplace [17].

The Workflow Broker Service performs actions on the marketplace on behalf of the user [19].

The Marketplace Service provides an environment for trading different computing resources and performs matching operations on available resources. This core component is responsible for running a market mechanism.

The Middleware Services that are necessary to perform real calculations on distributed computing resources are the Execution Engine, the Supplier Component, and the Monitoring Service. The Execution Engine connects the Marketplace to the Computing Grid of the supplier. It accepts requests for starting, stopping, and scaling clusters in the Cloud.

The Supplier Component is executed on the provider's computing resources, metering and monitoring the usage of the Cloud resources (e.g., CPU, Memory). It provides feedback to the buyer of the resource.

The Monitoring Service monitors the resource usage of the allocated resources and makes the data available to the economic-aware Services.

Finally, the Auxiliary Services are comprised of the Workflow Engine, the History Service, the Web User Interface, and the User Emulation Service. The Workflow Engine is responsible for orchestrating the interactions between the services according to the workflow of the platform user (Section 4.1.1).

The History Component stores all information that is generated during the runtime of the system.

The Web User Interface provides a single User Interface to the platform. A platform user can use the interface to

connect to its platform configuration and to test his design of the market environment.

4.1.3 User emulation service

The User Emulation Service is not used during the real operation of the system. It has been developed for testing the system against some defined usage scenarios. A simple command line interface is provided in order to configure the test/simulation scenario. After reading input data that describes a specific usage scenario, it allows changing any subset of the input data (e.g., number of requests, and the duration of the test).

This service generates requests towards a Web Service interface implemented by the Workflow Engine specifically for this purpose. This Web Service offers the same functionality as the nominal Web User Interface, but it facilitates the development of a simulation. For example, this service can generate bids and asks towards the Marketplace Service.

In particular, the User Emulation Service allows specifying the model according to which demand and supply are generated. For each simulation scenario, the basic market parameters can also be specified. This comprises the number of time slots, N, for which the market will be simulated, the number of providers, P, the number of customers, B, which will participate, as well as the number of bids and asks that each of them will submit.

4.2 Simulation parameters of the user emulation service

With respect to the model for generating supply, we have opted for a randomization of the ask parameters in order to create a realistic market supply. This randomization allows us to generate a wide variety of asks. In order for these asks to be able to be meaningful under realistic market conditions, we have decided that the values of the various parameters are drawn from uniform distributions having support over certain intervals. We have chosen uniform distributions since they provide varying load conditions within a simulation run. Therefore, we can ensure that the difference of demand minus supply fluctuates over time with highest competition experienced towards the middle of simulation. This enables us to simulate varying load competitions within a simulation run.

In particular, we generate users' bids and asks such that some variance in the willingness-to-pay is both expected and reasonable. This way, the resulting values for the price, duration, and quantity are what we expect to see in an actual system under various load conditions (e.g., high supply with few large providers offering large numbers of resources, or a more competitive system, in which a few large providers coexist with many smaller cheaper providers that offer computing resources for shorter time intervals). In detail, the various ask parameters are selected as follows:

- Ask price: This is the price wanted per VM and slot by the provider. This value is generated in the *[1, maxPrice]* interval. In our model we specify two classes of providers, namely "High" and "Low" price providers.
  - Providers of "High Price" draw each of their ask price from a uniform distribution in the *[maxPrice/2+1, maxPrice]* interval.
  - Providers of "Low Price" draw each of their ask price from a uniform distribution in the *[1, maxPrice/2]* interval.
- Quantity of VMs: The quantity of virtual machines offered in the market per ask, denoted as q, is drawn from a uniform distribution in the *[s_min, s_max]* interval. The actual values of this interval may vary in order to generate various types of providers (e.g., large or smaller).
- Ask duration: This parameter defines the number of slots that the quantity of VMs will be offered in the market. In our simulation model, we specify two classes of providers with respect to this parameter, namely providers of "High Supply Duration" and "Low Supply Duration".
  - Providers of "High Supply Duration" choose their ask duration value from a uniform distribution in the *[N/2+1, N]* interval.
  - Providers of "Low Supply Duration" choose their ask duration value from a uniform distribution in the *[1, N/2]* interval.
- Time interval: For the exact time interval *[from, to]* when the resources will be offered, it suffices to define the *from* parameter, since we have already specified the ask duration. The parameter *to* calculates as $to = from + duration$. The value of the start of the ask time interval *from* is drawn from a uniform distribution in the *[1, N – duration]* interval. This interval ensures that all ask time intervals are valid with respect to the simulation time of the system, N.
- Submission time: The submission time determines the time when each ask will be submitted to the system. Since it is only reasonable to set this value prior to the *from* value, this is chosen randomly to a time smaller than *from*.

The model used to generate demand is equivalent to that of supply. Therefore, we do not describe in detail the bid price, quantity of VMs, bid duration, time interval, and submission time.

It is worth noting that the aforementioned User Emulation Service, though simple and intuitive, can be used to simulate a wide variety of market conditions. It also ensure randomization of demand and supply.

4.3 Implementation of the GridEcon Market Environment

In order to implement the GridEcon Market environment that has been introduced in Section 3, we use the GridEcon Platform (Section 4.1). The implementation of the GridEcon Market Environment and the interactions between the different services are depicted in the following figure.

In particular, Fig. 5 highlights that the buyers and sellers only interact via the marketplace, which connects to the middleware and auxiliary services. The Capacity Planner, Insurance Broker, and the Fixed Price Quotation Broker interact with the buyer as well as with the middleware and auxiliary services.

In more detail, Fig. 6 visualizes the main sequence of transactions between the services, as specified by the GridEcon Market Environment. Note, in order to better visualize interactions of the Marketplace, the Marketplace service has been split into two components (Marketplace and Marketplace Scheduler).

Figure 6 also depicts that the Workflow Engine (synchronously or asynchronously) interconnects all services. A bid/ask submitted by the user is forwarded by the Workflow Engine to the Marketplace. The Marketplace informs the Workflow Engine about matched bids and asks and the Workflow Engine notifies in turn the user. When an application is to be started, the Marketplace Scheduler notifies the Workflow Engine, and, then, the appropriate request is sent to the Execution Engine. The Execution Engine is responsible for instantiating and terminating a cluster in the Cloud.

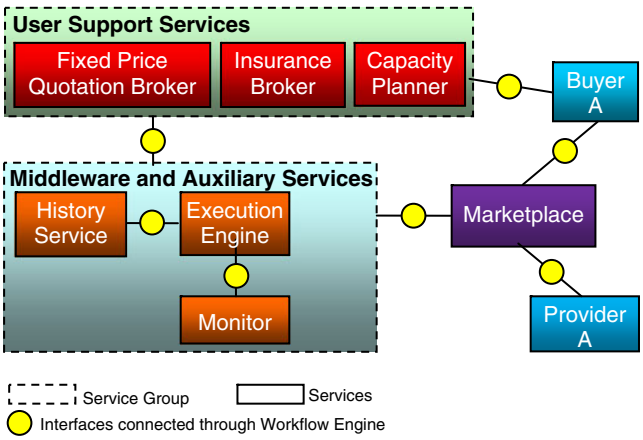

Fig. 5 Service configuration of the platform for validating the GridEcon Market Environment

Based on the sequence diagram of Fig. 6, the Web User Interface has been developed. An example of the implemented Web User Interface is given in Fig. 7. It depicts a snapshot of the market panel, showing the current asks and bids in the market. As can be seen in Fig. 7 as well, there are three more panels, allowing the user to submit orders, to view existing orders, and to logout. The *emulation* panel allows the user to set the simulation parameters.

4.4 Discussion

Through the implementation of the GridEcon Marketplace prototype, we could verify the validity of the design of the GridEcon Marketplace Environment as described in Section 3. In particular, we could demonstrate that the data exchange required between the user and the envisioned value-added services could be realized through a service-oriented architecture. Besides, it could be shown that the GridEcon double auction market mechanism for commodity resources (i.e., VMs) can be offered in a simple way. In particular, the market mechanism can run in parallel with other market mechanisms or be substituted without effort.

In addition to this, the implemented GridEcon Market Environment can be used for analyzing the performance of the GridEcon Market Mechanism. For this analysis, instead of having actual users submitting bids, we use the User Emulation Service. The simulation results (i.e., the data about bids and asks, the matches, and point in time when the match happened) are stored in the database of the History Service. The detailed results are presented in the following section.

4.5 Measurement results

4.5.1 Measures

In this subsection, we present the technical evaluation metrics, which we use to evaluate specific aspects of the GridEcon market mechanism by means of simulations using the GridEcon Platform. In particular, we consider the following three metrics:

Impact of frequency of invocation Though user demand/supply is expressed in discretized time slots, the frequency of invocation of the matching algorithm remains an open issue. Therefore, we have to investigate how often the matching module should be invoked so that the system performance is optimized. In particular, we examine various invocation methods of the matching module:

- Asynchronous Invocation Method: Whenever a new ask/bid is submitted to the system, the matching module is invoked. This way the matching module's

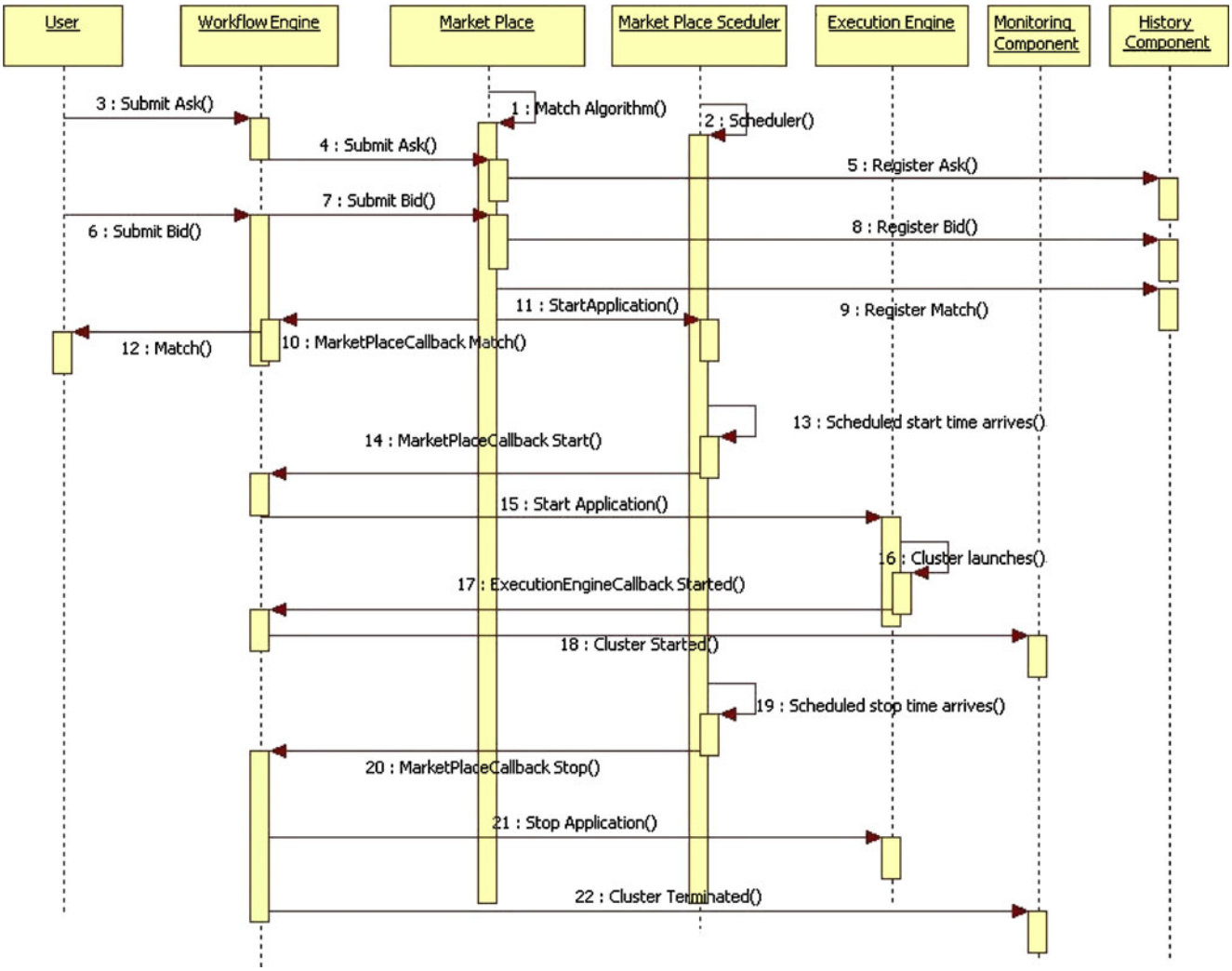

Fig. 6 Sequence diagram of implemented services

invocation frequency is the maximum possible with respect to the data (asks and bids) stored in the market directory.

- Per-Time-Slot Invocation Method: Rather than invoking the matching algorithm on every ask/bid submission, it is invoked once prior to the beginning of the next time slot. This way, the market directory is updated with all asks and bids submitted in the system between the next time slot and the previous time slot. Compared with the asynchronous invocation, the frequency of invocation is less, while the data set that the matching algorithm processes is larger.
- Per-Many-Time-Slots Invocation Method: The matching algorithm is being processed every T time slots, where $T > 1$. For our experiments, we have set three values of T, namely 5, 10, and 15. This approach invokes the matching algorithm less frequently, compared with the previous approaches. However, it also may operate on a large amount of bids and asks.

Impact of price indexing Rather than keeping all asks and bids in a common list, we can partition them in different indexes with respect to price. The idea is that there is no need to waste time by trying to match a bid with higher price asks. A set of indexes that group bids and asks with respect to their price may improve the system performance. The number of price indexes can be set to any value that is greater than one. For our simulations, the number of price indexes is set to five (for both bids and asks, respectively).

Impact of time indexing The idea here is similar to that of price indexing. We partition the bids and asks with respect to the time interval where they try to sell or buy resources.

4.5.2 Simulation scenarios and results

We conducted simulation runs for two simulation scenarios. The first simulation scenario comprised 1,000 time slots

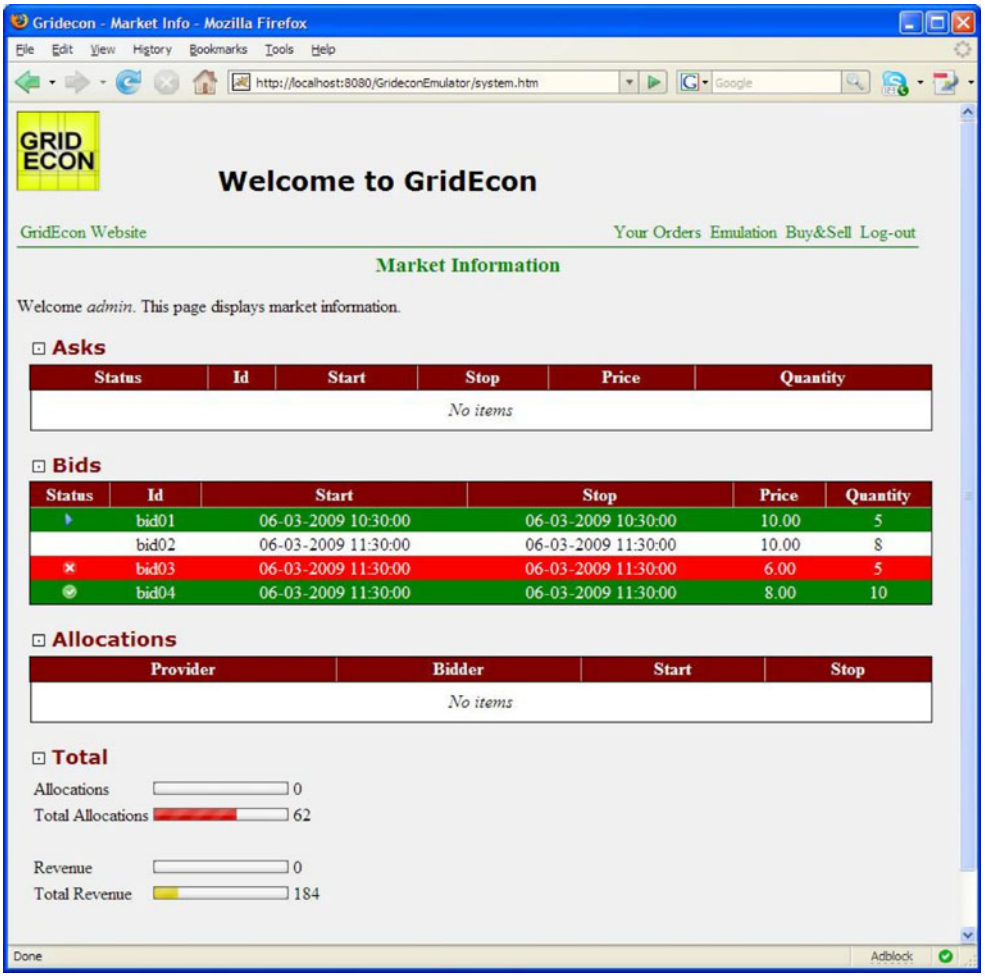

Fig. 7 Snapshot of the market

with 30 providers submitting 300 asks and ten bidders submitting 100 bids. The experiments with these simulation parameters were run on a laptop with AMD Turion TL50 processor with 1 GB DDR2 running Windows XP. In order to check on the correct workings of the implementation, we verified the matches of bids and asks. Indeed, the number of possible matches (i.e., 90), which could be performed, were found by the algorithm. Besides, in order to account for the variation of simulation results, we repeated the simulation experiment 50 times. Based on these simulation results, we computed the average values of the different system performance parameters. The average execution time of the different variations of the matching algorithm is shown in Fig. 8.

The performance differences of the various flavors of the GridEcon market mechanism are distinct. In particular, asynchronous invocation is the most efficient solution. This is due to the fact that most of the complexity of the matching algorithm comes from sorting and searching the bids and asks indexes. Since the asynchronous invocation tends to have indexes of smaller sizes, given a certain set of bids and asks, this also optimizes the overall system performance. Per slot invocation performs worst, however, it is substantially improved by means of price and time indexing. It is worth noting that in this case time indexing outperforms price indexing due to the much larger spread of bids and asks over time, as compared with their spread over price values. The performance of the system deteriorates substantially, if the matching module is invoked rarely (only every 5, 10, or 15 slots).

It is also worth emphasizing that due to the fact that the market directory reveals full information, the matching algorithm is always efficient and optimal, i.e., it computes a match for a bid (if possible) and this match is the cheapest possible for the two customers. Consequently, the number of matches found by the market matching algorithm is also the maximum possible.

A second simulation scenario was designed to estimate the execution times under increased load (compared to the first simulation scenario). Therefore, the simulation experiments of the market mechanism were performed for 1,000 time slots with 75 providers submitting 750 asks and 75 bidders submitting 750 bids. As for the previous simulation scenario, 50 simulations experiments were executed on the same computer as before. Based on these simulation results, the average execution time over the 50 experiments was

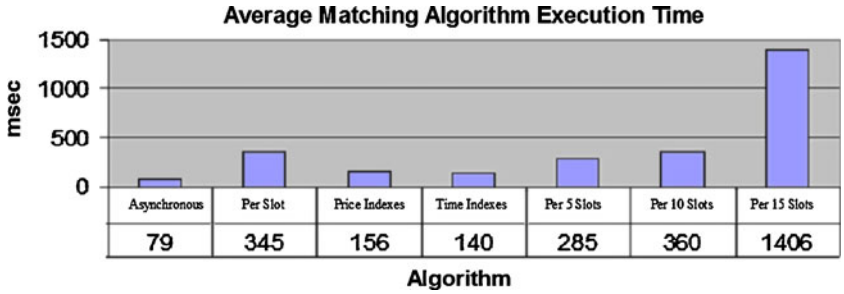

Fig. 8 Results of simulation scenario 1

calculated. The average number of matches feasible and actually found by the algorithm was 739. Figure 9 depicts the average execution times of the different variations of the matching algorithm.

As Fig. 9 shows, the asynchronous invocation still outperforms all other approaches. The per-slot invocation results in a delay that is almost double the delay that an asynchronous invocation incurs. The performance of price and time indexing is very good due to the large number of bids and asks in the directory. Both indexing methods result in an impressive system performance, which is similar to that of asynchronous invocation. Periodic invocation of the matching module results in much worse performance, due to the larger size of the market directory.

In general, since the number of bids and asks is high in the market directory in the second scenario, the matching algorithm typically works over larger directory sizes than in the case of the first simulation scenario. For clarification, assuming the simple case, in which asks are stored in the directory without additional indexing, the matching algorithm needs to perform two tasks for finding potential matching asks for a bid: Firstly, it has to check all the ask prices and time durations to see if an ask is indeed a candidate match for the bid; Secondly, it has to sort the candidate asks so that the cheapest unit-of-trades (i.e., VMs) are returned as a matching solution. Therefore, the complexity of invoking the matching algorithm for one active bid can be defined as $O(\#asks) + O(\#candidate\_asks * log(\#candidate\_asks))$, i.e. the complexity is proportional to the number of asks in the directory. Moreover, whenever a new ask is submitted to the directory, the matching algorithm must check for all the active bids whether they can be served. Subsequently, it must also compute the cheapest solution for them. Therefore, the time that the matching algorithm needs to be executed is an increasing function of the number of asks and bids that remain unmatched in the market directory.

4.5.3 Discussion

The main findings of the experimental assessment of the matching algorithm are independent of the load of the system. However, the more the system is loaded the clearer the differences in the system performance become. The main conclusions are as follows:

Impact of frequency of invocation The more frequently the matching module is invoked, the better the system performance is. In fact, simulations indicate that the best solution is that the matching module is invoked asynchronously. On the contrary, invoking the matching module periodically per $T > 1$ time slots results in poor performance. These results are caused by the fact that most of the algorithm complexity is due to the searching and sorting of the list of active bids and asks. The more frequently the module is invoked, the less the length of the market data and, thus, the better the performance. Note also that this approach minimized the delay of the system response to a user on whether a match for his bid/ask has been found or not.

Impact of price indexing Price indexing accelerates the matching. The higher the variance of the prices, the better the system performs with price indexing. The best number and ranges of price indexes can be obtained ex post (i.e., upon the deployment of the system) by analyzing the bid

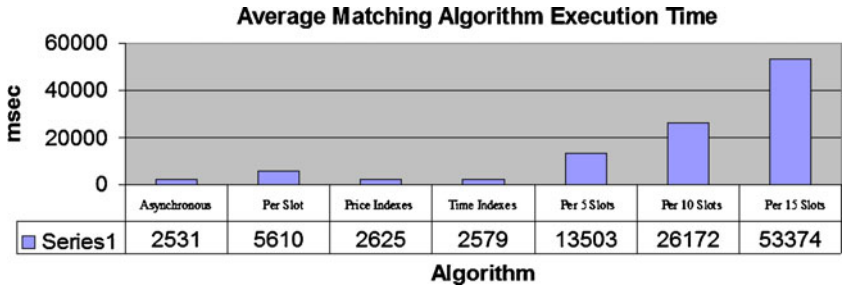

Fig. 9 Results of simulation scenario 2

prices and ask prices stored in the system. The analysis can provide a good answer to the optimality question. Nevertheless, it is also possible to use a simple clustering algorithm that operates on-the-fly on bid prices and ask prices and dynamically reconfigures these indexes. However, the specification and implementation of such algorithms are beyond the scope of this analysis.

Impact of time indexing Results are similar to that of price indexing. For the simulations conducted, we used a time index, which groups those bids (and asks respectively) together that overlap in time. The results indicate that the shorter the length of the duration of the asks (and bids) is, the better the classification of bids and asks in the various indexes is and, therefore, the better the overall system performance is. However, finding the optimal grouping algorithm is also not in the scope of this analysis.

Concluding, throughout all experiments conducted, the difference on the performance of the various flavors of the GridEcon market mechanism is significant. Therefore, we recommend that the GridEcon marketplace should opt for an architecture that allows invoking the matching module asynchronously. User actions that modify the market status (i.e. the bids and/or the asks in the system) should trigger the asynchronous invocation of the matching module. We can also argue that price and time indexes should be used to further speed up the system performance. It is also worth noting that the matching algorithm could be performed in parallel, if each index runs on a separate computer. Though this was not actually simulated, it is expected to further improve the system performance.

5 Conclusion

Within this paper, we designed a technical and economic framework for trading computing resources. In particular, we designed the GridEcon Marketplace (along with its value-added support services) and its market mechanism. The market mechanism design comprises a detailed specification of the unit-of-trade, the bids and asks, as well as the matching algorithm.

The unique feature of our marketplace is the opportunity of users to buy and sell computing resource services. This feature allows users of the marketplace to adapt their usage strategies (e.g. buy more and own less computing power; compute during the night only) based on their demand and the supply of the market. Therefore, the GridEcon Marketplace provides an alternative to the existing oligopoly in the market for Cloud computing (e.g., Amazon, HP, IBM, Google, Sun). Finally, it allows new businesses (i.e., value-added service providers) to emerge on top of this marketplace, offering value-added support services.

We validated the GridEcon Marketplace and the GridEcon Market Mechanism, using the service-oriented GridEcon Platform. This platform can be used for testing and validating new business models and business scenarios in the Cloud. In detail, for validating the market mechanism, we used the User Emulation Service of the platform. It helped finding the most efficient variation of the basic market mechanism.

The validation results show that the market mechanism is efficient with respect to handling bids and asks. The most efficient variation of the basic market mechanism is the one that invokes the matching algorithm immediately after the arrival of a bid or ask. Our results also indicate that the performance of the matching algorithm can be improved even further, if price or time indexing is used in combination with the asynchronous invocation option. However, this is to be validated in our future studies.

In addition to this, our future work will use the GridEcon Marketplace for simulating different capacity planning strategies in a market for Cloud computing resources. In particular, we will focus on the effect of the market mechanism on different capacity planning decisions.

Acknowledgement This research has been supported by the European Commission within the framework of the FP6 ICT GridEcon project (contract no. 033634). The authors would also like to thank the members of the GridEcon consortium for their contributions. Special acknowledgement is due to George Stamoulis, Manos Dramitinos, Kostas Giannakakis, Thierry Rayna, Alan Fleming, and Alon Lahav.

References

1. Amazon, Elastic Compute Cloud (Amazon EC2). Available at: http://www.amazon.com/gp/browse.html?node=201590011. Accessed December 2008
2. Buyya R, Abramson D, Giddy J (2001) An economy grid architecture for service-oriented grid computing. 10th IEEE International Heterogeneous Computing Workshop (HCW 2001). San Francisco, IEEE Computer Society Press, Los Alamitos, CA
3. Expedia. Available at: http://www.expedia.com/Accessed January 2010
4. Altmann J, Courcoubetis C, Stamoulis GD, Dramitinos M, Rayna T, Risch M, Bannink C (2008) "GridEcon—a market place for computing resources", GECON 2008. Workshop on Grid Economics and Business Models, Springer LNCS, Las Palmas, Spain, August
5. Lai K, Rasmusson L, Adar E, Zhang L, Huberman BA (2005) Tycoon: an implementation of a distributed market-based resource allocation system. Multiagent Grid Systems 1(3):169–182
6. Musil, S (2008) Amazon's S3 Experiences Outage. Cnet News.com. Available at: http://news.cnet.com/8301-1023_3-9995301-93.html. Accessed January 2010
7. Neumann J v, Morgenstern O (2007) Theory of games and economic behavior. Princeton University Press, Princeton, NJ 60 Anv. edition
8. The Popcorn Project. Available at: http://www.cs.huji.ac.il/~popcorn/. Accessed 2008

9. Regev O, Nisan N (1998) The POPCORN Market——an Online Market for Computational Resources. In: Proceedings of the First International Conference on Information and Computation Economies. ICE 1998. ACM, New York, NY
10. Risch M, Altmann J (2008) Cost Analysis of Current Grids and its Implications for Future Grid Markets. In: Proceedings of the Grid Economics and Business Model Workshop. GECON 2008. Springer LNCS, Heidelberg. pp 13–27
11. Waldspurger CA, Hogg T, Huberman BA, Kephart JO, Stornetta WS (1992) Spawn: a distributed computational economy. IEEE trans softw eng 18(2):103–117
12. Pattanaik KK, Singh R, Sahoo G (2007) An e-resource trading paradigm for computational grids, IJCSNS international. J Comput Sci Netw Secur 7(7):302–309
13. The gridbus project. Available at: http://www.gridbus.org/ Accessed 2009
14. Rahman M, Buyya R (2008) An Autonomic Workflow Management System for Global Grids. Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid, CCGrid. IEEE Computer Society, Washington, DC, In, pp 578–583
15. Buyya R, Abramson D, Giddy J (2000) Nimrod-G: an Architecture for a Resource Management and Scheduling System in a Global Computational Grid, The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), Beijing, China. IEEE Computer Society Press, New York, USA
16. Venugopal S, Buyya R, Winton L (2004) A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids. In: Proceedings of the 2nd Workshop on Middleware for Grid Computing, MGC 2004, vol.76, ACM, New York, NY, USA. pp 75–80
17. Risch M, Altmann J, Makrypoulias Y, Soursos S (2008) Economics-Aware Capacity Planning for Commercial Grids. In: Collaborations and the Knowledge Economy, pp.1197-1205. IOS Press, Amsterdam
18. Quan DM (2006) "Mapping heavy communication Workflows onto Grid Resources within SLA context", Proceedings of the Second International Conference on High Performance Computing and Communications (HPCC). Munich, Germany
19. Quan DM, Altmann J (2007) "Mapping of SLA-Based Workflows with Light Communication onto Grid Resources", GSEM 2007, 4th International Conference on Grid Service Engineering and Management. Leipzig, Germany
20. Quan DM, Kao O (2005) Mapping workflows onto grid resources within an SLA context. Proc Eur Grid Conf EGC LNCS 3470:1107–1116
21. Risch M, Altmann J, Guo L, Fleming A, Courcoubetis C (2009) "The GridEcon Platform: A Business Scenario Testbed for Commercial Cloud Services," GECON 2009, Workshop on Grid Economics and Business Models. Springer LNCS, Delft, Netherlands
22. Chuliang Weng, Minglu Li, Xinda Lu, Qianni Deng (2005) "Economic Based Resource Management Framework," CCGrid 2005, Cardiff, Wales, UK
23. Sandholm T, Lai K, Ortiz JA, Odeberg J (2006) "Market-Based Resource Allocation Using High Performance Computing Grid for Scientific Applications," In: Proceedings of IEEE HPDC
24. Wolski R, Brevik J, Plank JS, Bryan T (2003) "Grid resource allocation and control using computational economies," In: Grid computing: making the Global infrastructure a reality. Wiley, New York
25. Xen. Available at: http://www.xen.org/. Accessed May 2009
26. Meinl T, Neumann D (2009) "A Real Options Model for Risk Hedging in Grid Computing Scenarios," HICSS '09. 42nd Hawaii International Conference on System Sciences. pp 1–10
27. Eric Schonfeld, TechCrunch. Available at: http://www.techcrunch.com/2008/01/30/amazon-earnings-call-details-web-services-use-up-more-bandwidth-than-amazoncom-the-kindle-is-a-hit/. Accessed January 2008
28. Tsunamic Technologies Inc. Available at: http://www.clusterondemand.com/. Accessed December 2009
29. Google Apps. Available at: http://www.google.com/apps/. Accessed March 2009
30. Salesforce.com. Available at: http://www.salesforce.com. Accessed March 2009
31. Sun Grid. Available at: http://www.sun.com/service/sungrid/index.jsp. Accessed December 2008
32. Enomaly. Available at: http://www.enomaly.com/. Accessed December 2009
33. Zimory. Available at: http://www.zimory.com/. Accessed December 2009
34. Zimory restrictions. Available at: http://www.zimory.com/index.php?id=33#c94. Accessed December 2009
35. Globus. Available at: http://www.globus.org/. Accessed January 2010
36. OGSA. Available at: http://www.globus.org/ogsa/. Accessed January 2010
37. Elmroth E, Gardfjäll P, Mulmo O, Sandgren A, Sandholm T (2004) An OGSA-Based Bank Service for Grid Accounting Systems; 2nd International Conference on Service Oriented Computing. pp 279–288
38. Sandholm T, Gardfjäll P, Elmroth E, Johnsson L, Mulmo O (2004) "An OGSA-based accounting system for allocation enforcement across HPC centers." In: Proceedings of the 2nd International Conference on Service Oriented Computing, New York, NY, USA. ICSOC, ACM, New York)
39. Karasavvas K, Antonioletti M, Atkinson M, Hong NC, Sugden T, Hume A, Jackson M, Krause A, Palansuriy C (2005) Introduction to OGSA-DAI services. In Lect Notes Comput Sci 3458:1–12
40. Mason R, Courcoubetis C, Miliou N (2009) "A Framework for Analyzing the Economics of a Market for Grid Services," 6th International Workshop on Grid Economics and Business Models, GECON 2009, Delft, The Netherlands
41. EMC CIS. Available at: http://www.emccis.com. Accessed January 2010