Mandi: A Market Exchange for Trading Utility Computing Services

Saurabh Kumar Garg, Christian Vecchiola, Rajkumar Buyya

Cloud Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{sgarg}@csse.unimelb.edu.au

***Abstract*—The recent development in Cloud computing has enabled the realization of delivering computing as utility. A number of companies have started building Cloud and Grid computing environments, and offering access to the resources as a "Pay as you go" basis. This has created a need for a market exchange (ME) that brings providers and consumers together and facilitates their trading. Such market environment eases the process of finding, comparing and buying, since they aggregate goods and services from a variety of sources and display them in a way which is helpful to customers. In this paper we propose a ME framework called "Mandi" that allows consumers and providers to trade computing resources according to their requirements. We first present the ME requirements that motivated our design and discuss how these facilitate trading of compute resources using multiple market models. Finally, we evaluate the performance of the first prototype of "Mandi" in terms of its scalability.**

I. Introduction

Utility computing paradigms such as Clouds and Grids promise to deliver highly scalable and cost effective infrastructure for running High Performance Computing (HPC) applications [1]. As a result, the scientific and industrial communities [2] have started using cheaper commercially available infrastructures to run their applications which can scale up based on demand, rather than maintaining their own expensive HPC infrastructure. For such communities, it would be useful to have access to a ME where the resource availability is aggregated from various commercial providers. The presence of a Market Exchange (ME) infrastructure can easily fulfill their demand for compute resources, that can also resolve other problems such as high cost of ownership and the infrequent use of HPC infrastructures.

ME represents a significant advancement in the state-of-the-art of utility computing, where just one large provider of resources is dominant (e.g. Amazon [3]) [4]. It can bridge disparate Clouds and Grids allowing consumers to choose providers that suit their requirements. Due to the competition between different providers in the exchange, the price will also be lowered down for compute resources, making it even more affordable to enterprises with a low budget.

To realize this vision, ME needs to support diverse services [4] including an infrastructure that allows (a) registration, buying and selling; (b) advertisement of free resources; (c) coexistence of multiple market models or negotiation protocols such as auction; and (d) Grid resource brokers to discover resources/services and their attributes (e.g., access price and usage constraints) that meet user QoS requirements. On the consumer side, projects such as Gridbus broker [5] and GridWay [6] provide the capability of negotiation with providers, and job submission and monitoring on the heterogeneous resources. On the provider side, technologies such as virtualization and market based resource management systems such as Tycoon [7] and Mirage [8], has enabled trading of compute resources. Thus, the necessary middleware for enabling utility markets from the resource provider and consumer sides are already present to make ME a viable step.

Current efforts in this direction by projects such as Bellagio [9], CatNet [10] and Ocean [11] are designed for specific market models and specific application models. But Grid is well known to have heterogeneous resources and applications. Moreover, each participant has its own objectives and requirements. Since each market model or negotiation protocol has its own advantages and disadvantages, one market model fits for all problems can be impractical [12]. In other words, in order to really achieve different objectives, the ME should support multiple and concurrent market models to provide flexibility for consumers and resource providers to choose market models or negotiation protocols according to their requirements.

In this paper, we propose the design and evaluation for such a ME called Mandi[1] that is the first step towards providing a generic marketplace to isolated resource providers and consumers, fulfilling their trading requirements. We aim to develop a light weight and platform independent service oriented market architecture which can be accessed easily by current Grid systems. Mandi supports automated buying and selling of compute services such as CPU time and storage. Mandi has been implemented using Web Service technologies as part of the Cloudbus Project, which is developing solutions for service-oriented utility computing that scale from centralized systems to Clouds.

In the next section, we will discuss in detail the need of marketplaces for advancement of market-based grid and

[1]Mandi is a colloquial term for marketplace in Indian languages

cloud computing. Then, in Section 3, we discuss related work and compare them with Mandi. In Section 4, we discuss requirements of a market exchange. Then in the subsequent sections, we describe the design and implementation of Mandi with evaluation results and conclusions.

II. Related Work

Many projects focussed on building a marketplace for Grid and Cloud infrastructures. Among them, the most prominent, which are related to our work, are GridEcon [13], SORMA [14], Ocean Exchange [11], Tycoon [7], Bellagio [9], and CatNet [10].

Many existing systems (such as Bellagio [9] and Tycoon [7]) have restrictive pricing and negotiation policies. Auctions are held at fixed intervals, and only one type of auction is allowed (e.g. First Price, Second Price [15]). More generic market architectures such as CatNet, Ocean Exchange, GridEcon and SORMA also currently support only one or two market models such as bilateral negotiation and combinatorial auctions. In SORMA, automated bidding is provided to participate in an auction or to bargain with a resource provider that may lead to increased delays for consumers who urgently need the resources. GridEcon project started with a vision to research into a viable business model for trading services in an open market. The current implementation of GridEchon only support commodity market model. Ocean Exchange currently supports commodity market model, while CatNet supports the only bargaining and contract/net models. SORMA project's Open Grid Market has similar architecture as proposed by CatNet [16]. Thus, in the most of the previous work, the choice of market model is decided by market itself. On the other hand, in Mandi, we leave the choice of negotiation and pricing protocols to the consumers and providers in the system. This is crucial as the choice of the market model (such as the auction and commodity model) and pricing (fixed, variable) can vary from participant to participant depending on their utility gained. Thus Mandi acts as neutral entity or middleman giving the flexibility to participant to use any market model or negotiation protocol for trading their service and allowing concurrent and multiple negotiations between market participants.

Moreover, these systems also handles the management of job execution after matching it to appropriate resource. While, in Mandi, unlike the other systems, the responsibility of actual job submissions to resources and monitoring lies with user brokers and provider's resource management system.

Thus, our main contribution in this paper is to propose a novel market-exchange architecture and design which reflects the real-world markets where different participants interact with each other using the mechanisms of their choice.

III. Motivation and Market Scenario

In past, the need for marketplace and market based mechanisms is argued and counter argued by many researchers such as Nakai and Van Der Wijngaart [17] argued about the instability of markets using the General Equilibrium (GE) theory. In spite of these arguments, the need for markets in Grid like infrastructures where any user can get resources on demand is a reality. Many commercial providers and consumers from various industries including logistics and automobiles companies exist today [18]. Moreover, it is pointed out by Shneidman et al. [19] that many computer systems have reached to a point where the goal is not always to maximise utilisation; instead, when demand exceeds supply and not all needs can be met, a policy for making resource allocation decisions is required. Hence, market based systems and exchanges are the next step. The benefits are following: a) Coordination between concurrent user demands, and answers a critical question of resource allocations i.e., who gets which resources and when?, b) Ease the resource discovery by aggregation of different resource providers and services, c) Increase the efficiency of system usage by reducing resource wastage and avoid problems like tragedy of commons, d) Competition among various resource providers leads to reduction in money and time, e) Automatically balance supply and demand, and f) the market exchange acts as a neutral third party which can support resolution of conflicts between consumers and providers.

The key participant in any market are resource consumers, providers and ME which act as an auctioneer or clearing house. The consumers need to execute their applications that require compute resources, while the providers have free compute resources which can be leased for executing the application. The ME on the behalf of consumers or providers do the matching according their objectives using a negotiation protocol such as Vickrey auction or double auction [4]. There are three trading scenarios exist in market exchange:

- **Consumer contacts providers** to initiate the trading. The ME in this context can be used to allow consumer to discover various compute services on sale by different resource providers. It allows consumers to join an auction running for compute resources conducted by a provider, or directly contact a provider for reserving compute resources as in the commodity market model. It can also hold auction for consumers if their need is not satisfied by current advertised services.
- **Provider contacts consumers** to initiate the trading. In the ME, the providers can also find clients for selling their compute resources. Thus, ME allows providers to join an auction initiated by consumers. It also facilitates the providers to start their own trading protocol such as auction, or advertise their compute resources.
- **Consumers and Providers use ME services** to coordinate the trading and finding the most appropriate match. In this scenario, the ME act **as a clearing house** running market model such as double auction.

The market scenario where ME acts as a clearing house is explained in Figure 1. Several users can submit their application requirements to the ME. These requirements can be in the form of number of compute resources, time at which users need these resources, minimum memory require-

ments, and the power of each computing resource. Users can access all services of ME through a user interface which also manages the authentication and authorization. Similarly resource providers can advertise their compute services (time slot: number of compute resources and time for which they are available) in the ME which will be maintained in resource catalogue and in a database for backup.

The Meta-Broker is the main component of ME that coordinates the trading between various consumers and providers. The meta-broker periodically holds a double auction. At each scheduling event, the Meta-Broker collects the time-slots and the broker (consumer) requests from the Resource Discovery and Queuing Service. According to the objective of the consumers and the providers whether it is cost minimization or time minimization, matching is performed. After matching each broker resource-request to a time-slot, the reservation of matched time-slot will be initiated by the Reservation Service. The information about reservations can be collected by user brokers from the ME.

IV. Market Requirements

The market framework requirement can be divided into two categories: infrastructure requirements and the marketplace requirements.

A. Infrastructural Requirements

1) **Scalability:** Since, increase in the number of resource requests can affect the performance of the ME, thus the scalability of the exchange can become an issue. The exchange architecture should be designed such that access to market services be least effected by the number of service requests. In addition, it should guarantee the best efficiency in matching the consumers demand and provider's supply.
2) **Interface Requirements and Grid Heterogeneity:** The user interface plays an important role in making the usage of any system easy for a wide variety of users. Depending on how user wants to access the market, different types of interfaces should be provided. In Grids, many market based brokers [6] [5] ease the process of accessing the Grid middleware. Similarly, on the resource provider side, heterogeneous resource brokers [7] with market based capabilities are available. Thus, these brokers should seamlessly access ME's services whenever required by invoking simple platform independent exchange APIs.
3) **Fault Tolerance:** As failure is the reality of any system, the ME should be able to resume its services from the closest point before the failure.
4) **Security:** To avoid spamming, there should be a security system for user registration. All the services of the exchange must be accessed by authorized users.

B. Marketplace Requirements

1) **Variety of Application Models and Compute Services:** The user resource requirement can vary according to their application model. For example, to run an MPI application, users may want to lease all the compute resources from same resource provider, which gives high bandwidth between communicating processes. Thus, users can have different types of compute resource demands depending on their applications. Similarly, resource providers can advertise different type of resources such as storage and virtual machines. Thus, the ME should be generic enough to allow submission of different types of compute resource requests and services.
2) **Multiple User Objectives:** Users may wish to satisfy different objectives at the same time. Some possible objectives include receiving the results in the minimum possible time or within a set deadline, reducing the amount of data transfer and duplication, or ensuring minimum expense for an execution or minimum usage of allocated quota of resources. Different tasks within an application may be associated with different objectives and different QoS (Quality of Service) requirements. The exchange should, therefore, ensure that different matching strategies meeting different objectives can be employed whenever required.
3) **Resource Discovery:** As discussed earlier, users may have different resource requirements depending on their application model and Quality of Service needs. Thus, the exchange should be able to aggregate different compute resources and should allow users to access and discover them on demand.
4) **Support for Different Market Models:** In Grids, many market based mechanisms have been proposed on different trading or market models such as auctions and commodity market [20]. Each mechanism, such as English auction and Vickery auction, has different matching and pricing strategies and has their own advantages and disadvantages. Thus, the exchange should be generic enough to support as many market models as possible.
5) **Coexistence/Isolation of Market Models:** Similar to real world markets, the ME should support concurrent trading of compute services by different negotiation protocols such as auction. For example, double auction and Vickery auction can coexist simultaneously and users can participate in each of them.
6) **Support for Holding, Joining and Discovery of Auctions:** Users can have requirements that may not be fulfilled by currently available compute resources, and thus, may want to hold their own auctions and invite bids. Moreover, any user can discover these auctions and join them if necessary.

The following sections present the architecture, design and implementation of Mandi market exchange that takes into account the challenges mentioned so far, and abstracts the heterogeneity of the environment at all levels from the end-user.

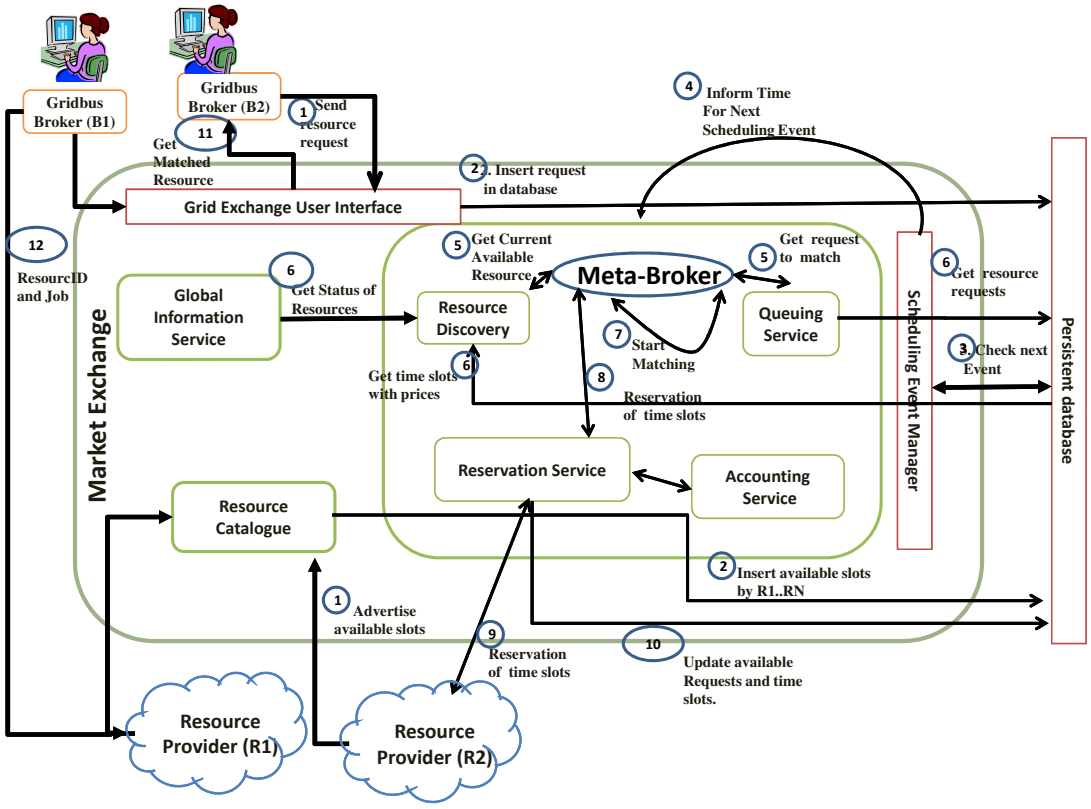


Fig. 1. Grid Exchange Protocol

V. Mandi Architecture and Design

A. Design Considerations and Solutions

The primary aim of Mandi is to provide a marketplace where the consumer's resource requests and the provider's compute resources can be aggregated, and matched using different market models. The main challenges and the way they are approached in Mandi design are as follows:

1) **Flexibility in choosing market model and user objectives:** As already discussed various market models or negotiation protocol can be used by users to trade compute resources. Each market model has different requirements [20]. For example, in the commodity market model, consumers search the current state of the market and immediately buy some compute service. This requires synchronous access to that instance of compute resource. In the case of auction, there is a clearing time when the winner selection is done. In addition, any user can request to hold their own auctions, thus a separation of each auction is required. Thus, the components within the Mandi are designed to be modular and cleanly separated on the basis of functionality. Each of them talks through the persistence database which reduces the synchronization problems. It is also possible to introduce new auction protocols by extending the one-sided auction and the double sided auction components. Each auction type is separated by "objective" and the winner selection mechanism. The reservation of matched services is handled independent of the trading mechanisms that allows flexibility and coexistence of different market models.
2) **Support heterogeneous compute resources and applications:** Mandi is designed to be as generic as possible independently of user application model and resource provider's middleware. Mandi has a service oriented architecture which allows platform independent interaction with Grid middleware.
3) **Fault tolerance:** Mandi can handle failure at two stages: during trading, and during reservation. The failure during reservation can occur due to network problems, and over subscription of resources. In the case of network problem, the failed resource requests will be considered in the next scheduling cycle. The reservation failure due to resource oversubscription is handled by consumers and providers. For the failures during trading, a parallel or bag of task application is not scheduled until all jobs/tasks are matched with a resource. In addition, The persistence database protects the Grid Exchange against failure during trading. The state of Mandi is periodically saved in the database. Thus, Mandi can resume its work from where it left.
4) **Scalability:** Most of the Mandi's component work in-

dependently and interact through the database. This facilitates the scalable implementation of Mandi as each component can be distributed across different servers accessing a shared database. In addition to that, most of the threads of Mandi are light weight and short lived.

B. Architecture Details

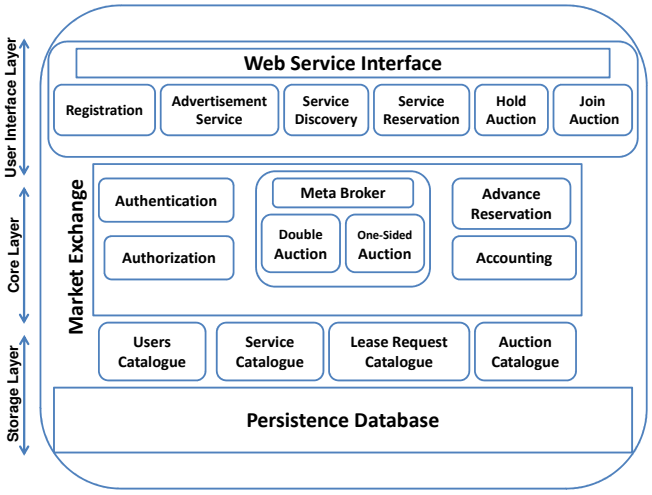


Fig. 2. Mandi Architecture

The architecture proposed in this work is inspired by the concepts of the 'Open Market' where any user can join , sell and buy their compute services. Mandi is made of three layers i.e. the user interface, the core layer consisting of the meta-broker service and the reservation service, and the storage layer. The main functions of these layers is described as follows:

User Interface layer: Consumers and compute resource providers can access the services of the market exchange through a Web Service interface. This layer provides the following features:

1) **Registration:** The users are needed to be registered before they can access the exchange's services. Their details are stored in the storage layer, and are used for authentication and authorization.
2) **Hold Auction and Join Auction:** This allows users to join any auction and bid for the items. Hold Auction allows participants to specify the type and the objective of auctions they want to hold. The objective of the auction help Mandi to decide the winner. Users have to provide details of the auction item (e.g. CPU time slot).
3) **Service Discovery and Service Reservation :** allow consumers to find services of their requirements and reserve them.
4) **Advertisement Service:** allows resource providers to advertise their CPU time slots.

Core layer: This layer performs main function of grid exchange. It provides functionalities like user authentications and authorization when users submit their login details. Its core components are the meta-broker, the accounting and the advance reservation system.

1) **Meta-broker:** It finds out the current one-sided auction and double auction from the auction catalogue and initiates the match process. After matching it invokes the advance reservation service to inform providers about the matching and the reservation of the resources.
2) **Advance reservation:** It informs the resource provider about the match, reserves the advertised (matched) service, and gets the reservation id that is used by consumer to submit his application.
3) **Accounting Service:** keeps the trading information of each user. It also keeps the information about transactions failed and successful.

Storage Layer: This layer is the interface between the database and other agents such as the web interface, the advance reservation and the meta-broker. Its main objective is to maintain the state of Mandi. It enables the recovery of Mandi in the case of unexpected failure, and is also useful in synchronizing exchange's various components.

To understand the interrelationships between the Mandi's components, it is necessary to see how they interact in different scenarios. The objects in the Mandi can be broadly classified into two categories- entity and workers. This terminology is derived from standard business modelling processes[21]. Entities exist as information containers representing the properties, functions and instantaneous states of the various architectural elements. The entities are stored in the database and are updated periodically. Workers represent the functionality of the broker, that is, they implement the actual logic and manipulate the entities in order to achieve the application objectives. Therefore, workers can be considered as active objects and the entities as passive objects. The next section (Section V-C) takes a closer look at the entities and workers within the broker accompanied by UML diagrams that illustrate the relationships between the components.

C. Entities

1) Users: Figure 3 shows the class diagram representation of user details. The Users class is used to store information about the participant members (consumers and providers) of Mandi. This information is used for authentication and authorization. From the point of view of the exchange, any user can act as consumer or provider thus there is not special field to differentiate this in the Users class.

2) Compute Resource: Figure 3 shows the TimeSlot Class that is used to represent of compute resources which are available as "commodities". The "TimeSlot" indicates the time for which resource will be available and how many CPUs will be available. Each "TimeSlot" is associated with a compute resource that is a representation of a set of CPUs or virtual machines advertised by the resource provider. The "TimeSlot" class can be also used for a storage resources where only the attributes related to time will be used. If a resource provider has conducted an auction for inviting bids for the time-slot,

then the AuctionType and the AuctionID attributes will be used to store information about the auction.

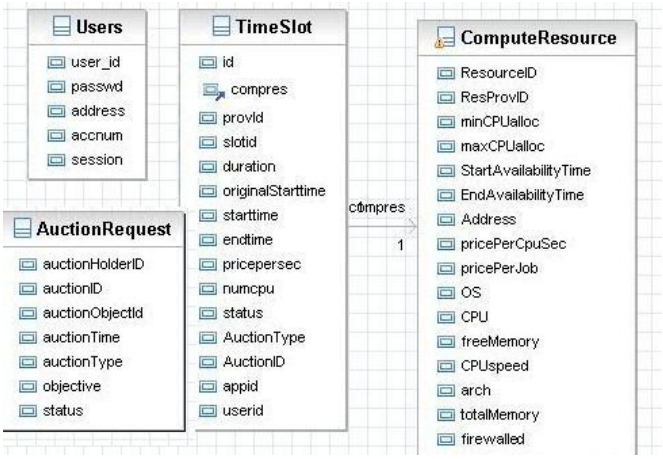


Fig. 3. User, and Timeslot Classes

3) Auction Request: All the information for holding an auction for any commodity advertised by a user is represented using the AuctionRequest Class. Figure 3 shows the class diagram of the AuctionRequest. Every auction is identified by a unique identifier i.e. auctionID. In economics, generally bids in auctions are considered in the form of monetary value. But in the case of computing service, a bid can be a more generalized form depending on the requirements of an auction holder. For example, a user holds an auction to find a resource provider who can lease the compute resource with minimum delay and within the specified budget. Thus, the user can invite bids in terms of the start time of the resource lease. Thus, Mandi provides facilities to define the ”objective” which is used as the criteria for choosing the winner bid. To enable integration of the different auction models with different matching and pricing strategies, the AuctionRequest class contains the ”auctionType” as an attribute which informs Mandi which auction user wants to hold.

4) Application: Application class represents the resource requests of the user's application such as total number of CPUs required, QoS requirements, deadline, and budget. The ”deadline” attribute represents the urgency of the user to get his/her application finished. The ”QoS” is an abstract class which can be extended to codify special application requirements such as bandwidth. Each application can consist of several jobs that may differ in their resource requirements such as execution time. To allow users to submit different application model requirements such as parameter sweep and parallel application, in Mandi, each application is associated with the ”appType” attribute that will be considered while matching an application with a resource. The application object also stores the information about the auction in which the user (consumer) has opted to participate for leasing resource for its application.

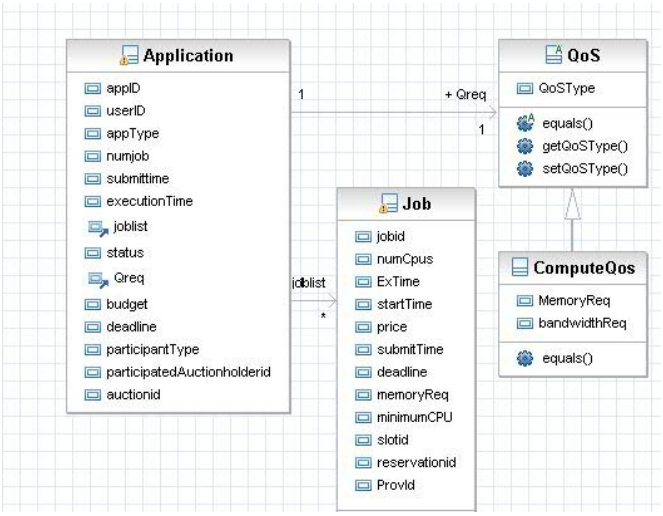


Fig. 4. Application, Job, and Qos Classes

D. Workers

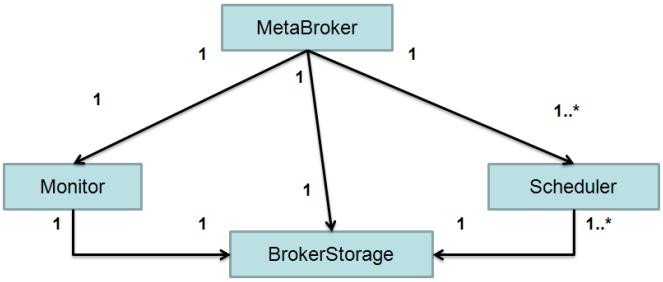


Fig. 5. Main Worker Classes

1) MetaBroker: MetaBroker is the first component in Mandi to be started that instantiates other worker components of Mandi, and manages the life cycles of other workers such as Scheduler, and Monitor. The BrokerStorage is the front end to the persistence system and implements interfaces used to interact with the database. Another function of the MetaBroker is to periodically get the list of current auction requests from the database and start a scheduling thread for clearing each auction.

2) GridExchange Service: The GridExchange Service is a Web Service interface which enables users to access various services of Mandi. The services that are available to users are registration, submission of application and time slots, holding and joining auctions, and discovering services and getting service reservations. The GridExchange Service interacts with the BrokerStorage class to access the persistence database. The example sequence of operations for user registration is shown in Figure 6. The UserBroker sends a registration request to the exchange using the GridExchange web service. It submits the preferred login name and password. The GridExchange service gets the registered user list from the database and checks whether the user is registered or not. If the user is not registered, it sends a reply back to user broker with

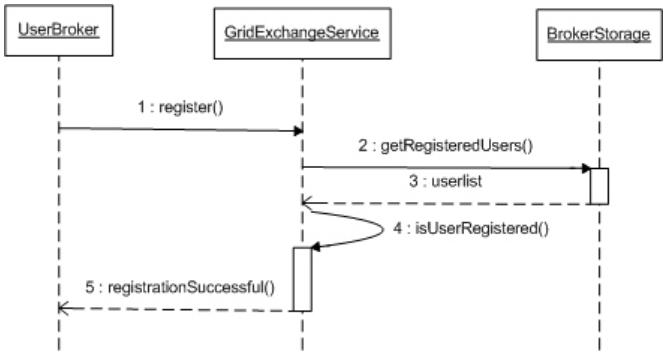


Fig. 6. Registration Process

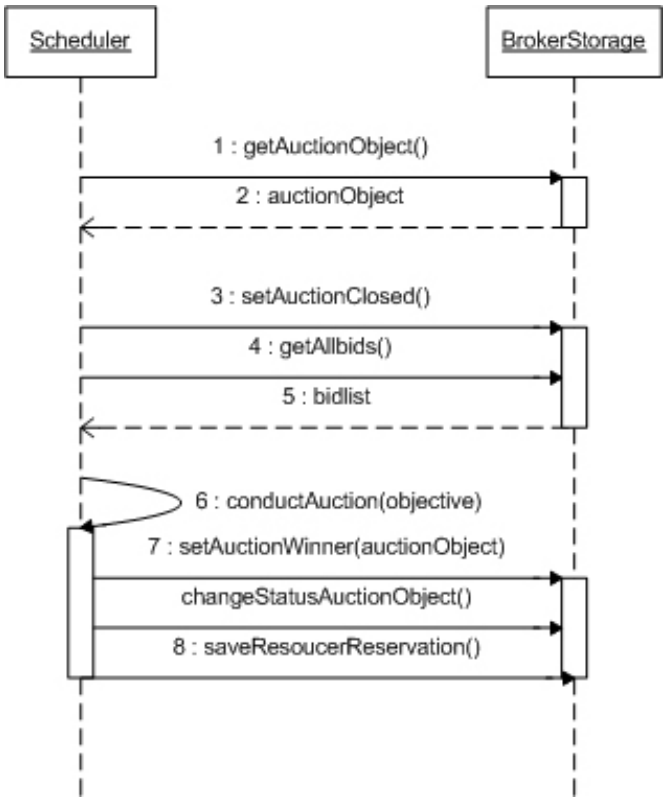


Fig. 7. Scheduling Sequence

"registration success" message.

3) Scheduler: For each market model, the Scheduler matches the user application to the advertised compute resources and also decides the price for executing the application. The Figure 7 shows the basic steps that are performed by the Scheduler. The Scheduler gets the auction object (can be in the form of a timeslot or an application) from the persistent database and the list of all the bids submitted for the auction. The Scheduler sets the auction status to "closed" to prevent any further bid submission to the auction. Depending on the auction type and objective, the winner bid is chosen and the trading price is calculated. The status of winner bid is changed to "matched" from "unmatched". The match is saved to database in the form a reservation request which will be used by the Monitor to inform/reserve resources on the compute resource. The function of the Monitor is described in detail below.

4) Monitor: The Monitor keeps track of all the reservation request in the database, as shown in the Figure 8. The Monitor periodically requests all the reservation requests from the persistent database. It uses Web Services to send SOAP messages to the resource provider, which informs the provider of the matching of the user application to the advertised timeslot (compute service). In the return, the Monitor gets the reservationID from the provider. The reservationID is used by the consumer to access the compute services offered by the resource provider. It represents the time-slot reserved and is also the security key for accessing the resource. After getting the reservationID, the Monitor will set all the reservation details in the user application object stored in the persistent database. The consumers (using brokers) can access the reservation information by using the GridExchange service.

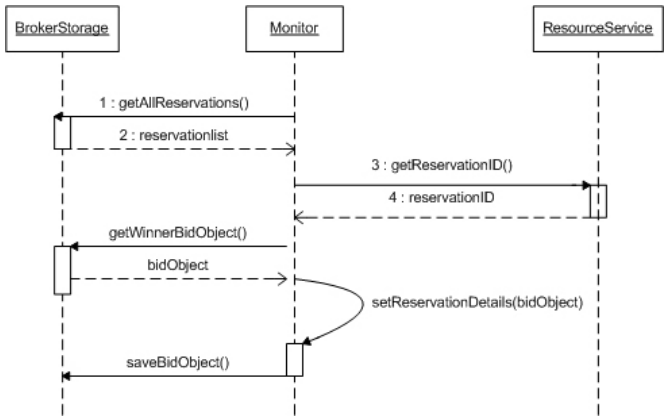


Fig. 8. Reservation Process

VI. Prototype and Performance Evaluation

In order to evaluate the performance of Mandi and provide a proof of concept of its architecture, we implemented a prototype and tested it by using Aneka as a service provider. In this section we will give an overview of the components composing the system used for testing and discuss the performance evaluation.

A. System Details

1) Mandi: Mandi has been implemented in Java which allows in order to be portable over different platforms such as the

Windows and Unix operative systems. From an implementation point of view Mandi is composed of a collection of threads that interact by means of a persistence layer represented by the HSQL database. The system is accessible from external components through a web service that has been deployed by using Apache Axis2 on a TOMCAT web server (v. 5.5). This makes the interaction with Mandi platform independent. The current prototype support three type of trading mechanisms: i) *First Bid Sealed Auction*; ii) *Double Auction*, and iii) *Commodity market*.

2) Aneka: On the provider side, Aneka [22] has been used and extended to support the reservation and advertisement of slots on Mandi. Aneka is a service-oriented middleware for building Enterprise Clouds. The core component of an Aneka Cloud is the Aneka container that represents the runtime environment of distributed applications on Aneka. The container hosts a collection of services through which all the tasks are performed: scheduling and execution of jobs, security, accounting, and reservation. In order to support the requirements of Mandi a specific and lightweight implementation of the reservation infrastructure has been integrated into the system. This infrastructure is composed by a central reservation service that provides global view of the allocation map of the Cloud and manages the reservation of execution slots, and a collection of allocation services on each node hosting execution services that are in charge of keeping track of the local allocation map and ensures exclusive execution for reserved slots. The reservation service is accessible to external applications by means of a specific Web Service that exposes the common operations for obtaining the advertised execution slots and reserving them.

3) Client Components: The client components are constituted by a simple web service client that generates all the resource requests to Mandi.

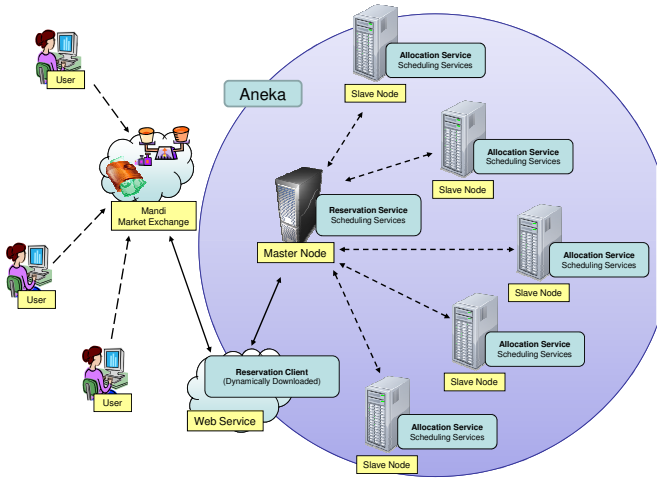


Fig. 9. The Topology of Testbed

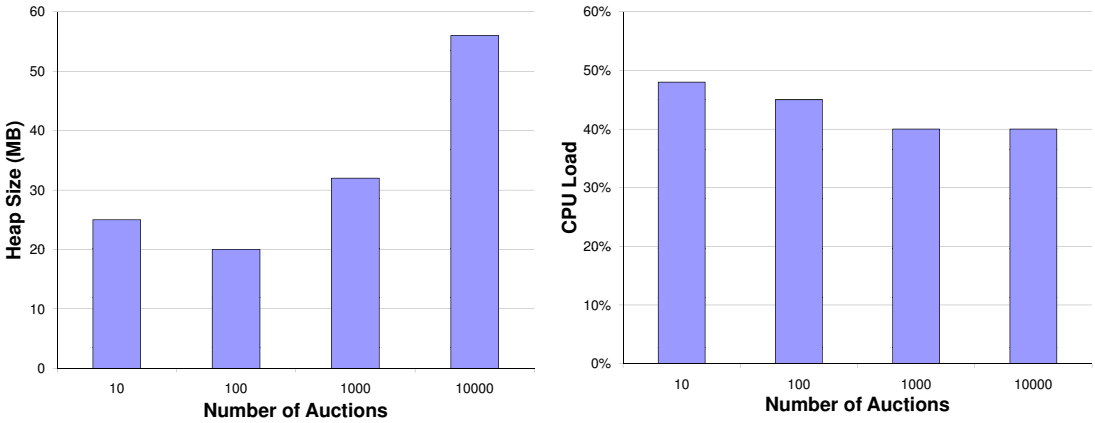
B. Performance Evaluation

We evaluated the performance of Mandi in terms of overhead caused to the system due to the interaction between the internal components and Mandi's interaction with users requests and provider's middleware. As discussed previously, Mandi is designed to handle multiple market models concurrently and exposes a service oriented interface to handle users requests and reservations of resources. Thus, to evaluate the scalability of Mandi, the first set of experiments examines the CPU and memory requirements of our implementation of Mandi. However, the performance of Mandi is also determined by how quickly and how many simultaneous user requests can be handled. Hence, the second set of experiments evaluates time incurred in resource request submission (which is initiated from client machine) and resource reservation (which involve negotiation of Mandi with providers).

The experimental setup for this evaluation is characterized as follows:

- An instance of Mandi has been deployed on 2.4 GHZ Intel Core Duo CPU and 2 GB of main memory running the Windows operative system and Java 1.5. The HSQL Database was configured to run on the same machine. The performance of Mandi evaluated using JProfiler [23] profiling tool.
- The Aneka setup was characterized by one master node and 5 slave nodes. The reservation infrastructure was configured as follows: the master node hosted the reservation service while each of the slave nodes contained an instance of the allocation service. Each container has been deployed on a DELL OPTIPLEX GX270 Intel Core 2 CPU 6600 @2.40GHz, with 2 GB of RAM and Microsoft Windows XP Professional Version 2002 (Service Pack 3). As a result the reservation infrastructure can features ten concurrent execution lines (one per core). The topology of resources is given in Figure 9.

1) Memory Usage and CPU Load: The main threads running in Mandi are: i) MetaBroker, which initiates other threads and controls the overall execution of Mandi, ii) Monitoring Thread, and iii) Scheduler Threads, which dynamically vary based on the number of auctions. Thus, the performance of Mandi is highly dependent on the number of auctions conducted concurrently. Thus, to evaluate the performance of Mandi, we varied the number of auctions from 10 to 10,000 that are conducted over period of 5 seconds. For this experiment, we generated 50,000 resource requests for matching. Each resource request is mapped to an auction using uniform distribution. Figure 10 shows the graphs of the memory and CPU usage by the broker over a period of 5 Second run. In Figure 10(b), the variation in CPU usage is about 10% with increase in number of auctions. This is because scheduler threads conducting auctions are short lived and has comparable CPU need. The little higher value of CPU usage in the case when 10 auctions are conducted is due to the large number of resource request per auction (50,000/10) needed to be matched.



(a) Memory Required by Mandi VS Auction Conducted

(b) CPU Usage by Mandi vs Auction Conducted

Fig. 10. Performance of Mandi for 50,000 clearance requests

In figure 10(a), we can see how memory usage of Mandi is increasing with the number of auctions. For instance, the memory usage increases from 32 MB to 56 MB when the number of auctions increases from 1000 to 10000. Therefore, there is only a 2 times increase in memory usage for 10 times increase in the number of auctions. This is due to the fact that the auction thread loads resource requests from database only when a decision for the auction winner needs to be taken. In addition, the memory is freed for all resource requests participating in the auction as soon as auction finished executing. This reduces the memory occupied by resource request objects waiting to be matched.

2) Overhead in Interaction with Resource Provider and Consumer: Two experiments were performed; one for measuring the resource request submission time and the other for reservation time and free slot advertisement by the provider middleware. All interactions between different entities i.e Mandi, consumer, and provider middleware is using web service. To measure these parameters, we used JMeter tool that generate SOAP messages to test the performance of web services. We generated SOAP messages until no more connection can initiated with the web service located at Mandi and resource provider's site. In case of interaction with Mandi's web service, about 750 concurrent resource submission requests were generated, while in case of interaction with Aneka reservation web service about 100 concurrent requests were generated. Table I shows the time taken to serve a request by web service in milliseconds. Overhead in terms of time for resource request submission is only 11.75 ms. The time taken by Aneka web service to serve free resource and reservation requests is much longer because each reservation request can trigger the interaction between the reservation service on the master node and the allocation service on the slave node where the reservation is allocated. This interaction implies the communication between two different containers and varies sensibly according to the network topology.

C. Discussion

The performance results indicate good scalability of current prototype of Mandi which is able to clear about 50,000 resource requests and 10,000 auctions in about 5 seconds. The major bottleneck in the scalability of Mandi's architecture is the shared database. The database constraints the number of multiple and concurrent accesses which is also the reason that experiments over 50,000 resource requests are not conducted. In addition to that the database can be cause of single point failure of whole system. The distributed databases which use replication and load balancing techniques can be helpful in increasing the scalability of the system.

TABLE I
OVERHEAD DUE TO INTERACTIONS OF MANDI

Web Service Request	Service Time/request (ms)
Resource Request Submission	11.75
Getting Free Resources	30
Resource Reservation	240

VII. CONCLUSION

The presence of IT demand and supply in utility oriented Clouds and Grids led to the need of a market exchange that can ease the trading process by providing the required infrastructure for interaction. In this paper we introduced the novel market exchange framework named "Mandi" for facilitating such trading. We identified the various technical and market requirements and challenges in designing such an exchange. We described the architecture and the implementation of Mandi and evaluated it with two experiments: measuring the effect of design choices on the performance of Mandi and measuring overhead time incurred in the interaction between the consumer and the provider through Mandi. The experiments shows that Mandi can scale well and can handle many concurrent trading models and resource requests. We can thus conclude that the overhead generated for matching a large number of resource requests in concurrent auctions

is minimal. The only limit to the scalability of the system is the persistence layer, which also constitutes the single point of failure. In order to address this issue, a more efficient database server and a solid replication infrastructure has to be put in place.

In the current implementation, the accounting and the banking service are not implemented, thus we aim to implement them in next version of Mandi. In future, we will like to consider large scale setups using Mandi. We plan to extend the Gridbus Broker [5] and integrate various resource providers such as Amazon. In addition, since in reality there will be multiple exchanges, thus we will research how they will intercommunicate.

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*, 2008, pp. 1–10.

[2] "Ian Foster, What's faster-a supercomputer or EC2? ," http://ianfoster.typepad.com/blog/2009/08/whatsfasterasupercomputerorec2.html, Aug. 2009.

[3] Amazon, "Amazon Elastic Compute Cloud (EC2)," http://www.amazon.com/ec2/, Aug. 2009.

[4] J. Broberg, S. Venugopal, and R. Buyya, "Market-oriented grids and utility computing: The state-of-the-art and future directions," *Journal of Grid Computing*, vol. 6, no. 3, pp. 255–276, 2008.

[5] S. Venugopal, K. Nadiminti, H. Gibbins, and R. Buyya, "Designing a resource broker for heterogeneous grids," *SoftwarePractice & Experience*, vol. 38, no. 8, pp. 793–825, 2008.

[6] E. Huedo, R. Montero, and I. Llorente, "A framework for adaptive execution in grids," *Software Practice and Experience*, vol. 34, no. 7, pp. 631–651, 2004.

[7] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system," *Multiagent and Grid Systems*, vol. 1, no. 3, pp. 169–182, 2005.

[8] B. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. Snoeren, and A. Vahdat, "Mirage: a microeconomic resource allocation system for sensornet testbeds," in *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*. IEEE Computer Society, 2005, pp. 19–28.

[9] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat, "Resource allocation in federated distributed computing infrastructures," in *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure*, vol. 9. Citeseer, 2004.

[10] T. Eymann, M. Reinicke, O. Ardaiz, P. Artigas, L. de Cerio, F. Freitag, R. Messeguer, L. Navarro, D. Royo, and K. Sanjeevan, "Decentralized vs. Centralized Economic Coordination of Resource Allocation in Grids," in *Grid computing: first European Across Grids Conference, Santiago de Compostela, Spain*. Springer-Verlag New York Inc, 2003.

[11] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M. Frank, and C. Chokkareddy, "OCEAN: The Open Computation Exchange and Arbitration Network," in *International Symposium on Parallel and Distributed Computing, Ljubljana, Slovenia*, October 2003.

[12] D. Neumann, J. Stößer, and C. Weinhardt, "Bridging the adoption gap - developing a roadmap for trading in grids," *Electronic Markets*, vol. 18, no. 1, pp. 65–74, 2008.

[13] J. Altmann, C. Courcoubetis, G. Stamoulis, M. Dramitinos, T. Rayna, M. Risch, and C. Bannink, "GridEcon: A market place for computing resources," *Lecture Notes in Computer Science*, vol. 5206, pp. 185–196, 2008.

[14] D. Neumann, J. Stoesser, A. Anandasivam, and N. Borissov, "Sorma-building an open grid market for grid resource allocation," *Lecture Notes in Computer Science*, vol. 4685, p. 194, 2007.

[15] C. Yeo and R. Buyya, "A taxonomy of market-based resource management systems for utility-driven cluster computing," *Software Practice and Experience*, vol. 36, no. 13, p. 1381, 2006.

[16] SORMA, "Economic Middleware and Grid Operating System Extensions," http://www.im.uni-karlsruhe.de/sorma/fileadmin/SORMA_Deliverables/D5.1_final.pdf.

[17] J. Nakai and R. Van Der Wijngaart, "Applicability of markets to global scheduling in grids," *NAS Report*, pp. 03–004.

[18] TOP500 Supercomputers, "Supercomputer's Application Area Share," http://www.top500.org/stats/list/33/apparea, 2009.

[19] J. Shneidman, C. Ng, D. Parkes, A. AuYoung, A. Snoeren, A. Vahdat, and B. Chun, "Why markets could (but dont currently) solve resource allocation problems in systems," in *10th USENIX Workshop on Hot Topics in Operating System*, 2005.

[20] J. Altmann, M. Ion, and A. Mohammed, "Taxonomy of grid business models," *Lecture Notes in Computer Science*, vol. 4685, p. 29, 2007.

[21] H. Eriksson and M. Penker, "Business Modeling with UML: Business Patterns at Work, John Wiley&Sons," 2001.

[22] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," *CoRR*, vol. abs/0910.1979, 2009.

[23] E. Cho, "JProfiler: Code Coverage Analysis Tool for OMP Project," 2006.