Software: Practice and Experience, Vol. 0(0), 1–0 (0000)
(will be inserted by the editor)

Mandi: A Market Exchange for Trading Utility Computing Services

Saurabh Kumar Garg, Christian Vecchiola, · Rajkumar Buyya

Received: date / Accepted: date

Abstract The recent development in Cloud computing has enabled the realization of delivering computing as an utility. Many industries such as Amazon and Google have started offering Cloud services on a "pay as you go" basis. These advances have led to the evolution of the market infrastructure in the form of a Market Exchange (ME) that facilitates the trading between consumers and Cloud providers. Such market environment eases the trading process by aggregating IT services from a variety of sources, and allows consumers to easily select them. In this paper we propose a light weight and platform independent ME framework called "Mandi", which allows consumers and providers to trade computing resources according to their requirements. The novelty of Mandi is that it not only gives its users the flexibility in terms of negotiation protocol, but also allows the simultaneous co-existence of multiple trading negotiations. In this paper, we first present the requirements that motivated our design and discuss how these facilitate the trading of compute resources using multiple market models (also called negotiation protocols). Finally, we evaluate the performance of the first prototype of "Mandi" in terms of its scalability.

Keywords Cloud Computing, Utility Computing, Market Exchange, Market Model

1 Introduction

Utility computing paradigms such as Clouds and Grids promise to deliver a highly scalable and efficient infrastructure for running IT applications. As a result, the scientific and industrial communities have started using commercially available infrastructures to run their applications that can scale up based on demand, rather than maintaining their own expensive HPC infrastructure.

To ease and control the buying and selling process, there are other players in the utility grid such as the Cloud market place or the Cloud market exchange [1] [2] which

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

allow consumers and providers to publish their requirements and goods (compute power or storage) respectively. The market exchange service provides a shared trading infrastructure designed to support different market-oriented systems. It provides transparent message routing among participants, authenticated messages and logging of messages for auditing. The market exchange can coordinate the users and lower the delay in acquiring resources. Moreover, the market exchange can help in price control and reduces the chances of the market being monopolized.

In addition, with the maturity of utility computing in the form of Clouds, both consumers and providers have to consider complex parameters for trading compute resources. For instance, users may have sophisticated Quality of Service requirements such as deadline, resource availability, and security. Moreover, the providers can trade using multiple negotiation protocol (auction, commodity market, and one-to-one) and pricing (fixed, variable), since each of them can enormously affect their utility depending on the current demand and supply. Thus, today's market exchange needs to support multiple and concurrent market models to provide flexibility for Cloud consumers and providers to choose negotiation protocols (market models such as commodity market, auction) according to their requirements.

Thus, in this paper we propose the design of a market exchange framework called 'Mandi'[1] which specifically addresses the needs of computing consumers and providers. It support diverse market exchange services [3] including (a) registration, buying and selling; (b) advertisement of free resources; (c) coexistence of multiple market models or negotiation protocols such as auctions; and (d) resource brokers and Resource Management Systems (RMSs) to discover resources/services and their attributes (e.g., access price and usage constraints) that meet user QoS requirements.

Currently, there is no real service available for trading resources on a market basis. The common usage pattern of Cloud Computing resources is to directly rent them from the provider by using a pay as you go model and a fixed price. Just recently, Amazon EC2 introduced the spot pricing market, which allow users to utilize a virtual machine as long as its hourly price does not exceed a predefined bid. This new opportunity, demonstrates the interest from the current providers in using more flexible market models for selling their service, but still does not fully implement the features envisioned by Mandi, where providers can advertise their services and offer different models for trading them. Other solutions, such as RightScale, provide support for transparently using multiple Cloud providers and simplifying the process for deploying application on different Cloud platforms. This support is mostly concerned with providing an infrastructure enabling the portability of applications without any automation for selecting a specific provider on a market basis. Times are mature then, both from an infrastructure and a market point of view, to establish a global market where resources providers can advertise their offerings and provide access to them with flexible negotiation models. In such a market, users should be able to publish their preferences and also actively initiate processes aimed at comparing and selecting the best offering suiting their needs, by means of, for example, auctions. Current market exchange solutions [4][1][2] provide very limited support in this direction especially for what concerns the selection of models that can be used to trade resources. In many cases [2][4], a single type of auction is available. Mandi, is a step towards this global vision and specifically designed to act as a flexible market supporting multiple negotiation protocols (such as auctions and commodity market) and pricing models. It is

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

a virtual market place where users and resource providers can trade to optimize their revenue.

In addition, Mandi is developed as a light weight and platform independent service oriented market architecture whose features can be easily accessed by current Grid/Cloud systems without many infrastructural changes. Our proposed architecture distinguishes itself from other meta-scheduling systems by separating job submission and monitoring (done by the user brokers such as Nimrod/G [5], GridBus Broker [6] and GridWay [7]) from resource allocation (done by the market exchange). It also overcomes some of the short-comings of existing systems by allowing co-existence of multiple negotiations of different types.

In the next section, we discuss requirements for a market exchange. Then in the subsequent sections, we describe the design and implementation of Mandi with evaluation and results. Then, we discuss related work on market exchange and their comparison with architecture of Mandi. Finally, we present the conclusions and future directions.

2 Market Exchange Requirements

The market exchange framework requirements can be divided into two categories: infrastructural requirements and market requirements.

2.1 Infrastructural Requirements

1. **Scalability:** Since the increase in the number of resource requests can effect the performance of the ME, thus the scalability of the exchange is an issue. The exchange architecture should be designed such that access to market services can be least effected by the number of service requests. In addition, it should guarantee the best efficiency in matching the consumer's 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 a user wants to access the market, different types of interfaces should be provided. In Grids, many market based brokers [7] [6] ease the process of accessing the Cloud resources. Similarly, on the resource provider side, heterogeneous resource brokers [8] 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 can occur anytime, 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.

2.2 Market Requirements

1. **Multiple Application Models and Compute Services:** The user resource requirements can vary according to their application model. For example, to run an MPI application, users may want to lease all the compute resources from the same resource provider, which gives much better bandwidth for communicating

processes. Thus, users can have different types of compute resource demands depending on their applications. Similarly, resource providers can advertise different types of resources such as storage and Virtual Machines (VMs). Thus, the ME should be generic enough to allow the 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 Quality of Service(QoS) 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 Multiple Market Models:** In Grids, many market based mechanisms have been proposed on different trading or market models such as auctions and commodity market [9]. Each mechanism, such as the English auction and the 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 Discoverying 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.

3 Mandi Architecture and Design

3.1 Design Considerations based on Requirements

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. We have summarized how each market exchange' requirement is considered in Mandi's design in the Table 1. The details of some of the main issues addressed in the design of Mandi are following:

1. **Flexibility in Choosing Market Model based on User Objectives:** As already discussed, various market models or negotiation protocols can be used by users to trade compute resources. Each market model has different requirements [9].

For example, in the commodity market model, consumers search the current state of the market and immediately buy some compute services. This requires synchronous access to those instances of compute resource. In the case of an auction, there is a clearing time when the winner selection is done. In addition, any user can request to hold auctions which require the separation of each auction. Thus, the components within the Mandi are designed to be modular and cleanly separated on the basis of functionality. Each of them communicates through the persistence database that constitutes a synchronization point in the entire system. Different auction protocols are abstracted as "one-sided auction" and "two- sided auction" which can be extended to add new auction mechanism. Each auction type is characterized by the winner selection mechanism. The reservation of matched services is separated from the trading mechanisms to allow the flexibility and coexistence of different trading models such as commodity market and auction.

2. **Handling Heterogeneity in Interaction with Compute Resources and User Brokers:** To allow various Resource Management Systems and brokers to access market exchange services, Mandi's architecture needs to provide simple platform independent APIs. Current market exchanges such as Sorma [2] handle the heterogeneity by implementing a plug-in for each resource management system to allow resource reservation, job submission, execution and monitoring. These special plug-ins can restrict the adoption of exchange since APIs of different resource providers for job submission, execution and monitoring may get updated with time. Thus, Mandi is designed to handle mainly the allocation of resources to user applications, while the job submission, monitoring and execution are left to user brokers. Mandi's services are available through platform independent APIs implemented using Web Services.
3. **Fault Tolerance:** Mandi can handle failures at two stages: during trading, and during reservation. To avoid failures during trading, the resources are not reserved, unless all the tasks of an application are mapped. To avoid allocation of one resource to multiple applications, one compute resource is allowed to be traded only in one negotiation.
   In addition, the persistence database protects Mandi against failures during trading. The state of Mandi is periodically saved in the database. Thus, Mandi can resume its work from the point of failure. he 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.
4. **Scalability:** To address the scalability issue, most of Mandi's components work independently 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. Since, Mandi handles only the resource allocation and delegates the management of job submission and execution to the participating brokers and providers' RMS, thus most of the threads in Mandi are light weight and short lived.

Table 1 Mapping of Market Exchange Requirements to Mandi's Architectural Design

Requirements	Design Consideration	Components
Scalability	Each auction thread is short lived and all information is maintained within database which can be distributed	Database and Auction Service
Handling heterogeneity	Web Service interface is enable Mandi to interact with heterogeneous user schedulers/brokers, Mandi itself is implemented with Java to handle platform heterogeneity	Core implementation
Fault Tolerance	Persistence database for fast recovery	Database Service
Security	Regular auditing of system, with authorization and authentication to check spam users	Accounting Service, Authorization/Authentication Service
Coexistence/Isolation of Market Models	Abstraction of Two-sided and One-side auction, API's are available to users to define new auctions	Metabroker service
Multiple User Objectives	Auction service allows holding of two types of objective i.e. response time and cost	
Resource Discovery	Information and resource repository	Resource and Database service
Support for Holding, Joining and Discovering Auctions	APIs are provided for users to allow such functionalities	Auction, Database and Metabroker service

4 Trading Scenario

Figure 1 shows a typical scenario of trading conducted within Mandi. In the example shown, Mandi conducted a double auction to match bids of multiple resource requests to the providers' ask. First, the providers advertise their resources with their price (*aka* asks). Consumers submit their bids to show their interest in leasing the advertised resources. All the bids and asks are stored in the database which will be accessed at the end of the auction for calculating the winning bids.

The Meta-Broker, which is the main agent of Mandi, coordinates the matching of asks and bids, and trading between auction participants. At the end of the auction, the Meta-Broker decides the winners and sends the reservation requests to the Reservation Service of Mandi. Then, the Reservation Service informs the resource providers and consumers about the auction result. The information about reservations is stored within Mandi using the Accounting service.

4.1 Architectural Components (Services)

The architecture of Mandi is inspired by the concepts of the 'Open Market' where any user can join, sell and buy their compute services. Figure 2 shows the service oriented architecture design of the Mandi and its main services. Mandi is organized into two main services i.e. the user services, and the core services. The core services consist of the Meta-Broker Service, the Reservation Service, Accounting service and the Database Service. Each of the services can run on different machines independently

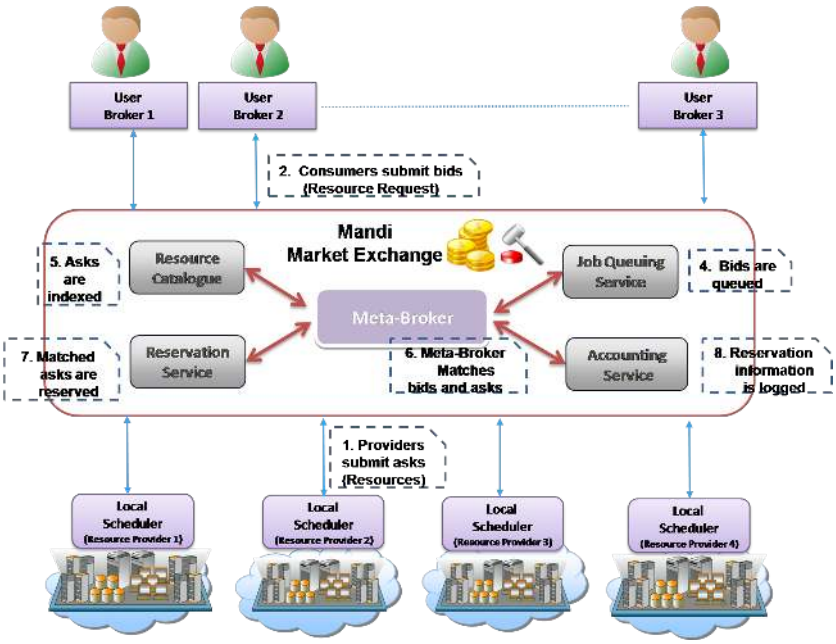


Fig. 1 Trading Scenario of Mandi

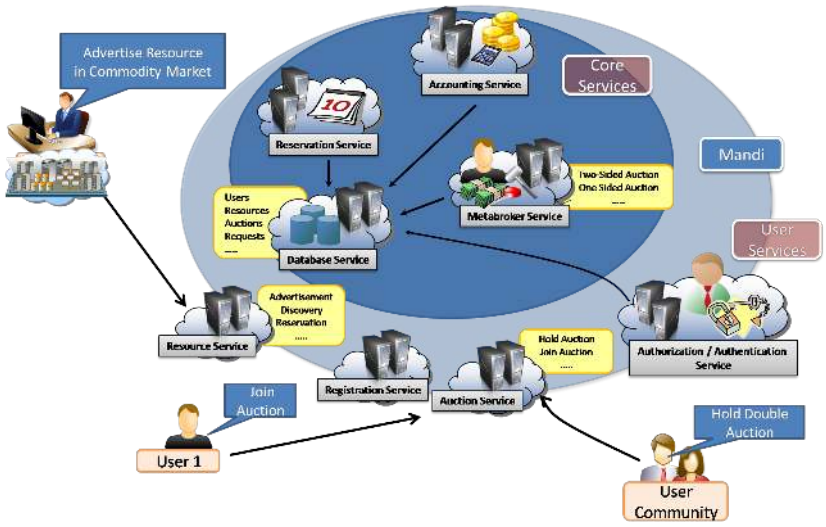


Fig. 2 Mandi Architecture

by communicating through the database service. The functionality of each service is described in the following sections.

4.1.1 User Services

The user services hide all the internal components of Mandi and implement all the services visible to market participants. It is implemented using Web Services which provide the consumers and resource providers platform independent access to the market exchange services. The following market services are provided to users:

1. **Registration Service:** The users need to register before they can access the exchange's services. The users' details are maintained in the storage layer, and are used for authentication and authorization.
2. **Auction Service:** This service allows a user to join any auction and bid for the items. The Hold Auction Service allows users to specify the auction types which they are allowed to initiate. Mandi can conduct two classes of auctions, i.e., one-sided auction and two-sided auction. In the case of two-sided auctions, multiple consumers and providers can choose to participate and get matched.
3. **Resource Services:** Resource Discovery and Reservation Service allow consumers to find services of their requirements and reserve them. This feature is added to integrate the commodity market model within Mandi. Advertisement Service allows resource providers to advertise their cloud resources (number of CPUs and time at which they will be available).
4. **Authentication and Authorization Service:** This service allows users to login into the market exchange and authorized them to access other Mandi services.

4.1.2 Core Services

These services consist of the internal services of the market exchange.

1. **Meta-Broker Service:** The initiation of any auction is managed by the Meta-Broker Service. It conducts the auction and announces the auction winner through Reservation Service. The auction can either be one-sided or two-sided. Thus, two components i.e. Two-Sided Auction and One-Sided Auction are provided to add customised auction protocols within Mandi.
2. **Reservation Service:** It informs the resource providers 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:** stores the trading information of each user. It also stores the information about the failed and successful transactions.
4. **Database Service:** This service is the interface between the persistence database and other services such as the web interface, the advance reservation and the meta-broker. Its main objective is to maintain all the trading information such as transaction history, users' details, auctions, compute resources for leasing, and user requests. This enables the recovery of Mandi in the case of unexpected failures, and is also useful in synchronizing various components of Mandi.

4.2 User Interaction Phases

To understand the interrelationships between Mandi's services, it is necessary to see how they interact in different use cases. There are several important use-cases of interaction with Mandi. There are two types of users trading in Mandi: a) consumers who

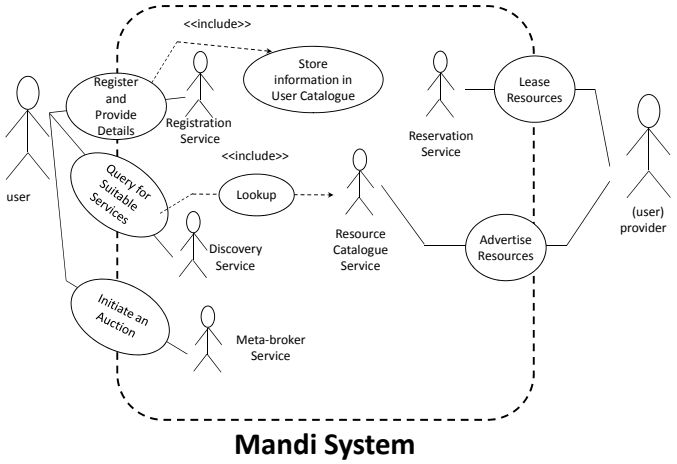


Fig. 3 Use Cases of Mandi

need compute resources to execute their applications, and b) resource providers who are providing their infrastructure as service. The figure 3 gives a comprehensive view of the uses cases addressed by Mandi. In particular it shows the operations that are performed by the users by differentiating their double role of consumers and providers of resources. The figure also shows which are the specific components of Mandi that are in charge of managing the requests associated to the specific operation displayed. Only the "Initiate an Auction" use case has been reported briefly due to its interactions with other components. A complete description of each of these use cases follows.

1. **Registration:** Every user needs to register and submit their personal details such as name, organization address, credit card details and contact phone number. After registration, users are issued a login id and password to access Mandi's services. The user details are stored in the persistence database and used mostly for the accounting and logging purposes. These details can be accessed by other entities through the User Catalogue component of the Database Service.
2. **Resource Advertisement:** Any user can advertise the compute resource available for leasing. Each resource is assigned a unique identifier that is registered in the Catalogue Service for discovery purposes. The user is required to give information such as how many CPUs/VMs are available for processing and what are their configurations. The user also needs to inform Mandi about the trading model to use for the these resources. If the user selects the commodity market model, then the leasing price of the resource should be given while advertising the resource. If the user selects the auction model, then the auction type is needed to be specified by the user. All the information is stored in the storage layer that is accessed by other components of Mandi using Resource Catalogue Service for allocation purposes.
3. **Service Discovery:** To discover resources that are advertised in Mandi for lease, users can use the Service Discovery Service. To find the resource of their choice, users just need to give the configuration of compute resources they are interested in and when they want to lease. Mandi will search the Resource Catalogue Service for

the required resources, and send the matched resources with their trading protocol information to users. Users can then select the resources of their choice and use either service reservation or join auction for leasing the resource.

4. **Leasing Resources in Commodity Market:** To allow the integration of commodity market model in the Mandi market exchange, the Reservation Service is added to allow users to directly lease the available resources. A user provides the identifier of the resources to be leased through Mandi. The Reservation Service adds the user's lease request in the Lease Request Catalogue Service which is regularly accessed by the Reservation Service. The reservation service does the final allocation of resources by informing the resource provider and adding the information in the persistence database for accounting purposes. After allocation, the leased resource and request are removed from the respective catalogues.
5. **Conducting an Auction:** To hold an auction, first a user needs to get the types of auctions currently supported by the market exchange. Then, the user can send a request to hold the particular type of auction with the relevant details such as auction item, minimum bid, and auction end time. If the auction item is a compute resource, the user is required to specify the time that the CPU/VM will be leased. All the auction requests are stored in the database. Depending on the chosen auction protocol, an auction thread (with a unique identifier) is initiated by the Meta-Broker Service. After the initiation of the auction thread, the unique identifier (AuctionID) is sent back to the user (auction holder). The auction thread waits till the auction end time is reached. Users who want to bid in the auction need to provide the AuctionID that can be discovered using Join Auction Service. Depending on the auction rules, the user is also allowed to resubmit updated bids. At the end of the auction, the auction thread collects the bids, executes the winner selection algorithm and sends the reservation request to the Reservation Service. The Reservation Service creates a contract for accounting purposes, and informs the participants about the auction outcome.

4.3 Implementation Details

The Class diagram which illustrates the relationship between Mandi's Objects is depicted in Figure 4. The objects in Mandi can be broadly classified into two categories - entities and workers. This terminology is derived from standard business modelling processes [10]. 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 updated periodically. Workers represent the internal functionality of Mandi, that is, they implement the actual logic and manipulate the entities in order to allocate resources to applications. Therefore, workers can be considered as active objects and the entities as passive objects. The next sections (Section 4.3.1and 4.3.2) take a closer look at the majore entities and workers within Mandi.

4.3.1 Entities

1. **User:** The User class is used to store information about the participants (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 no special need to differentiate between them in

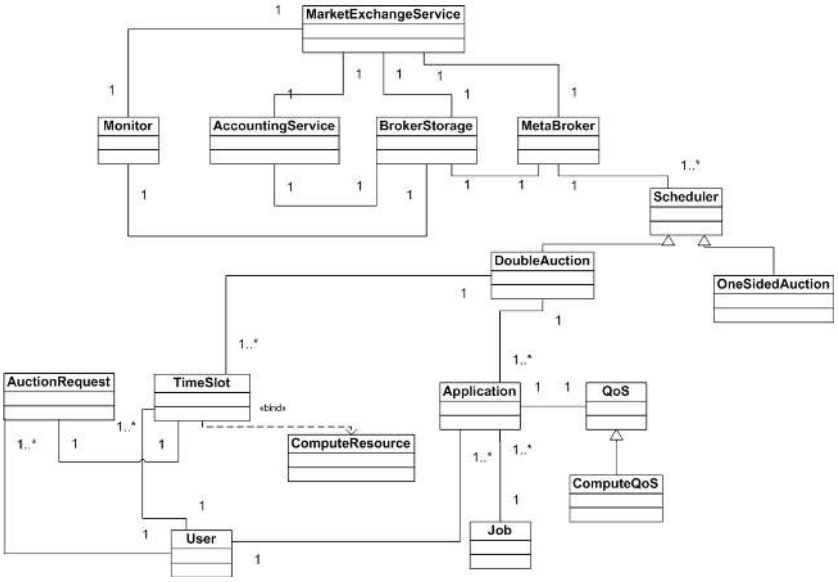


Fig. 4 Mandi Class Design Diagram

Mandi. Each user can advertise multiple compute resources, and submit multiple lease and auction requests.

2. **TimeSlot and ComputeResource:** The TimeSlot class is used to represent compute resources available for leasing by indicating how many CPUs are available for how much time. Each "TimeSlot" is associated with one compute resource that is a representation of a set of CPUs or VM advertised by the resource provider. 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 the auction's information. Each "TimeSlot" can be associated with only one auction.
3. **Auction Request:** All the information for holding an auction for any commodity advertised by a user is represented using the AuctionRequest class. 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. Mandi provides facilities to define different types of auctions which can be implemented by extending the TwoSidedAuction and OneSidedAuction classes. Each auction mechanism is specified by its winner selection algorithm.
   To enable the co-existence of multiple auction based negotiation with different matching and pricing strategies, the AuctionRequest class contains the "auctionType" attribute informing Mandi which auction the user wants to hold.
4. **Application:** The Application class abstracts the resource requests of the user's application that consists of the total number of CPUs required, QoS requirements, deadline, and budget. The "deadline" attribute represents the urgency of the user to complete his/her application. The "QoS" is an abstract class that 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 in leasing resources for its application. Each application is allowed to participate in only one auction.

4.3.2 Workers

1. **MetaBroker:** The MetaBroker is the first component in Mandi to be started that instantiates other worker components, and manages their life cycles 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.

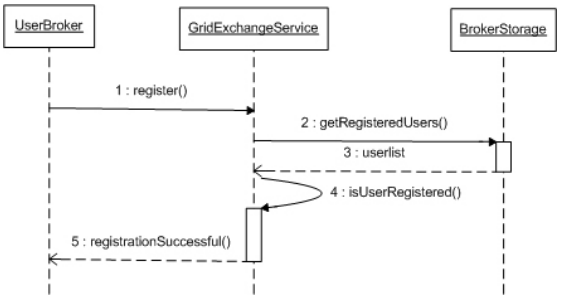


Fig. 5 Registration Process

2. **GridExchangeService:** The GridExchangeService is a Web Service interface that 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 5. The UserBroker sends a registration request to the exchange using the GridExchange web service with the preferred login name and password. The GridExchangeService 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 the UserBroker with "registration success" message.

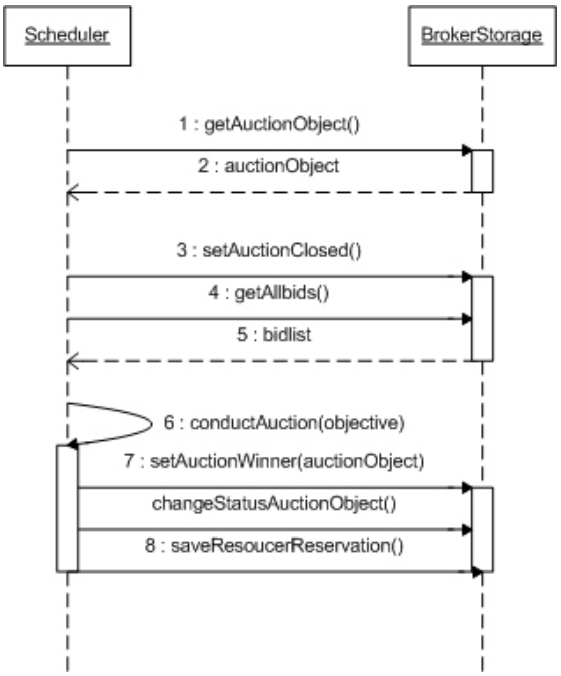


Fig. 6 Scheduling Sequence

3. **Scheduler:** For each market model, the Scheduler matches the user application to the advertised compute resources and decides the price for executing the application. Figure 6 shows the basic steps that are performed by the Scheduler. The Scheduler gets the auction object (in the form of a time-slot or an application) from the persistent database and the list of all the bids submitted for the auction. Then, 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 winning bid is chosen and the trading price is calculated. The status of the winning bid is changed to "matched" from "unmatched". The match is saved to the database in the form a reservation request which will be used by the Monitor to inform and reserve resources on the compute resource. The function of the Monitor is described in detail below.
4. **Monitor (*aka* advance reservation):** The Monitor keeps track of all the reservation requests in the database, as shown in the Figure 7. The Monitor periodically requests all the reservation requests from the persistent database. It uses Web Services to send SOAP messages for the resource provider to inform them about the matching of the user application to the advertised time-slot (compute resource). 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 acts as 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 GridExchangeService.

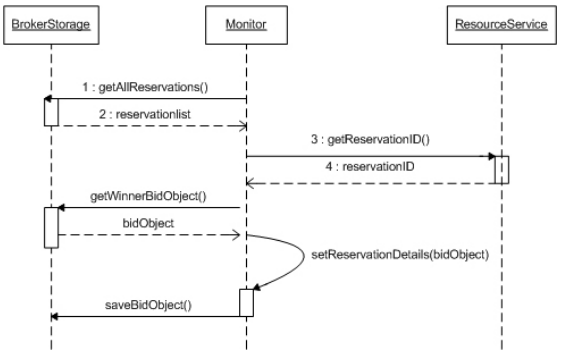


Fig. 7 Reservation Process

5 Prototype Details

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 [11] as a service provider. In this section, we describe the prototype implementation and discuss its performance results.

5.1 Mandi

Mandi has been implemented in Java in order to be portable over different platforms such as the Windows and Unix operative systems. From the implementation point of view, Mandi is composed of a collection of services 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). The Web Service interface makes the interaction with Mandi platform independent. The current prototype support three types of trading mechanisms: i)*First Bid Sealed Auction*; ii) *Double Auction*, and iii) *Commodity market*.

5.2 Aneka (for Automated Advertisement and Reservation of Resources)

On the provider side, Aneka [11] 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 resource 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 of 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 that keep the track of local map and ensure the exclusive execution on 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.

5.3 Client Components

The client are constituted by a simple Web Service client that generates all the resource requests to Mandi.

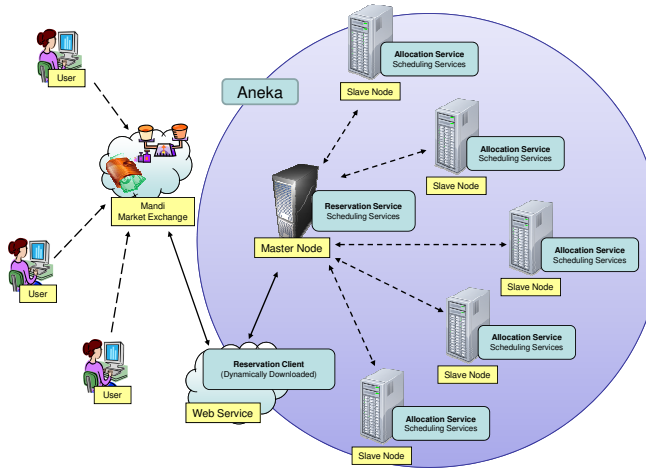


Fig. 8 The Topology of Testbed

6 Performance Evaluation

To evaluate the performance of Mandi, two sets of experiments are conducted:

1. **Comparison of Mandi with other approaches discussed in the related works:** We compared Mandi with two approaches used in other market exchanges:
   – *Continous Double Auction (CDA):* In this market model, the market place matches the bids and asks from users continuously. If any bid or ask is unmatched it is queued up unless matched. This market model is a core part of many Grid market places such as Sorma project [2].

- *Commodity Market:* In this market model, prices are considered to be fixed. This model is more reliable since users can know more quickly whether he will get resources or not. GridEcon [1] is based on the commodity market. To compare the above approaches with ours, we use three metrics: a) number of successful scheduling b) delay in scheduling per user, and c) cost incurred per consumer.

2. **Stress Analysis of the Mandi Prototype:** In these experiments, we evaluated the overhead generated by 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 CPU and memory load generaged by Mandi as the number of simultaneous negotiation increases. 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 the delay in resource request submission (which is initiated from the client machine) and resource reservation (which involve the 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 memory running Windows operating system and Java 1.5. The HSQL Database was configured to run on the same machine. The performance of Mandi evaluated using the JProfiler [12] 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 support ten concurrent VMs (one per core). The topology of resources is given in Figure 8.

6.1 Comparison of Mandi with other Market Exchange Approaches

In this experiment, since our aim is not to evaluate a particular trading protocol but to evaluate why Mandi like market exchange is much more beneficial, we have used simple configurations. We consider 5000 consumers who requests market exchange for scheduling their jobs. Each job has a deadline constraint. Each user is assumed to submit their jobs with a delay between one to two seconds. We consider 5 resource providers to advertise their resources to market exchange. For simplicity, it is assumed that each provider advertises similar resources; each of which is assumed to be capable of all the submitted jobs. The fixed price asked by provider for their resources is assumed to $2 per second. Since in general, the commodity prices are higher than auction prices, it is assumed that both consumer and provider's bids and asks are uniformly distributed between ($1,$2). The delay incurred in submitting the request with resource reservation and accounting to market exchange is considered to be 10 seconds. In Mandi, we gave each consumer and provider three options: a) participate

in CDA conducted by Mandi b) buy or sell in the commodity market or c) start a new first sealed price auction and invite bids from other market participants. Thus, several experiments are conducted with varied percentage of jobs with tight deadline. The results of these experiments are presented in Table 2, 3, and 4. From these tables, we can clearly observe that Mandi resulted in minimum cost to its participants with minimum delay in comparison to other approaches. CDA resulted in second best cost for consumers, but due to delay in finding appropriate match many jobs missed their deadline. The commodity market provides access to resources immediately but does not guarantee the best price for consumers. Thus, this experiment clearly shows that when users has complex QoS of service requirement such as deadline or budget, then a market exchange which provide multiple choices to them is much more viable.

Table 2 Cost incurred to consumers per job ($)

% of Tight Job Deadlines	Mandi	Only CDA	Only Commodity Market
10%	1.285364838	1.36785296	2
30%	1.293125863	1.37674317	2
50%	1.290176674	1.372885465	2
70%	1.293281083	1.377506229	2
90%	1.290952776	1.373555695	2

Table 3 Delay incurred to each consumer in job scheduling(sec)

% of Tight Job Deadlines	Mandi	Only CDA	Only Commodity Market
10%	10	10.8694786	10
30%	10	10.93667214	10
50%	10	10.89998991	10
70%	10	10.94208023	10
90%	10	10.90817005	10

Table 4 Jobs missed their deadline

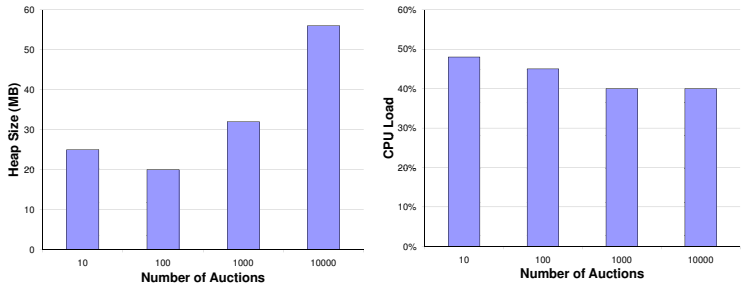
% of Tight Job Deadlines	Mandi	Only CDA	Only Commodity Market
10%	0	396	0
30%	0	446	0
50%	0	427	0
70%	0	447	0
90%	0	437	0

6.2 Stress Analysis of Mandi Prototype

In order to provide a global evaluation of our system we also conducted a stress analysis. The analysis of the prototype is aimed at investigating the impact on handling multiple

concurrent market models in terms of overhead incurred in terms of memory and CPU usage, and the communication of Mandi components with user brokers.

6.2.1 Memory Usage and CPU Load



(a) Memory Required (b) CPU Usage

Fig. 9 Performance of Mandi for 50,000 clearance requests

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. Since, the performance of Mandi is highly dependent on the number of auctions conducted concurrently, we varied the number of auctions from 10 to 10,000 that are conducted over a 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 9 shows the heap size (memory usage) and CPU usage by the broker over a period of 5 Second run. In Figure 9(b), the variation in CPU usage (load) is about 10% with the increase in number of auctions. This is because scheduler threads conducting auctions are short lived and has comparable CPU needs. The little higher value of CPU usage registered for the case of 10 auctions is due to the large number of resource request per auction (50,000/10) needed to be matched.

In Figure 9(a), we can see how the 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 the 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 is completed. This reduces the memory occupied by resource request objects waiting to be matched.

6.2.2 Overhead in Interaction with Resource Provider and Consumer

Two experiments were performed; one for measuring the resource request submission time and the other for free slot advertisement and reservation time by the provider middleware. All interactions between different entities, i.e Mandi, consumer, and provider

middleware are using web service interfaces. We used JMeter to generate SOAP messages for testing the performance of web services. SOAP messages are generated until no more connection can be initiated with the web service located at Mandi and resource provider's site. To stress test the 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 5 shows the time taken to serve a request by web service in milliseconds. The time overhead due to resource request submission is only 11.75 ms. The time taken by Aneka web service to locate free resource and confirm the reservation 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 involves the communication between two different containers and varies sensibly depending on the network topology.

6.3 Discussion

The performance results indicate good scalability for the 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, the database can be the cause of single point failure of whole system. Distributed databases which use replication and load balancing techniques can be helpful in increasing the scalability of the system.

Table 5 Overhead due to Interactions of Mandi

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

7 Related Work

As discussed earlier, there are many market solutions proposed for trading grid and cloud resources both from academia and industries. Various industrial solutions from companies such as Cloud Market and Rightscale more or less act as information and deployment systems allowing users to search their appropriate resources. While Mandi is a market place which allow any provider or user to trade using the negotiation protocol of his/her own choice.

Many research projects focussed on building a market exchange for Grid and Cloud infrastructures. Among them, the most prominent, which are related to our work,

are GridEcon [1], SORMA [2], Ocean Exchange [13], Tycoon [8], Bellagio [14], and CatNet [4].

Table 6 Comparison between Mandi and Other market exchanges

Characteristics	GridEcon	Sorma	Ocean-Exchange	Catnet	Bellagio	Mandi
Negotiation Protocol	Commodity Market and Double Auction	Combinatorial Auction	Bilateral Negotiation	Bargaining	Combinatorial Auction	Commodity market, One-sided auction, and Two-sided auction
Pricing	Static and spot pricing	K-Pricing	Static	Static and dynamic pricing	K-pricing	Static and dynamic pricing such as spot pricing
User/Provider Role	Bidding	Bidding	Discover and negotiate	Bidding	Bidding	Discover, initiate or bid in an auction, buy in commodity market
Job Submission and Monitoring	yes	yes	yes	yes	yes	no
Flexibility of Market Model	no	no	no	no	no	yes

Many existing systems (such as Bellagio [14] and Tycoon [8]) 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 [3]). More generic market architectures such as CatNet, Ocean Exchange, GridEcon and SORMA also 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 resources. The GridEcon project started with a vision to research into a viable business model for trading services in an open market. The current implementation of GridEcon only supports the commodity market model. Ocean Exchange currently supports the commodity market model, while CatNet supports only the bargaining and contract/net models. Thus, in the previous work, the choice of market model is decided by the 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 the utility gained. Even major cloud companies such as Amazon [15] currently offer multiple services based on commodity using different pricing. Thus, Mandi acts as a neutral entity or middleman giving the flexibility to participants to use any market model or negotiation protocol for trading their service. Mandi also allows concurrent and multiple negotiations between market participants.

Moreover, these systems also handle the management of job execution after matching it to appropriate resource. Thus, if a new resource provider wants to participate in the market, a special plugin is required to be implemented to allow the management of job submission and execution. In Mandi, the responsibility of actual job submission and execution is delegated to the user brokers and provider's resource management system. In addition, consumers and providers can access Mandi's services using a platform independent Web Service interface. Thus, our main contribution in this paper is to propose a novel market-exchange architecture which reflects real-world markets

where different participants interact with each other using a trading mechanism of their choice. The comparison with related work is summarized in Table 6.

8 Conclusion and Future Directions

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 a 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 the overhead time incurred in the interaction between the consumer and the provider through Mandi. The experiments show 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. 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 services 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 [6] and integrate various resource providers such as Amazon. In addition, since in reality there will be multiple exchanges, we will research how they will intercommunicate and trade with one another.

References

1. J. Altmann, C. Courcoubetis, J. Darlington, and J. Cohen, "GridEcon-The Economic-Enhanced Next-Generation Internet," in *Proceedings of the 4th International Workshop on Grid Economics and Business Models, Rennes, France*, 2007.
2. D. Neumann, J. Stoesser, A. Anandasivam, and N. Borissov, "Sorma-Building an Apen Grid Market for Grid Resource Allocation," in *Proceedings of the 4th International Workshop on Grid Economics and Business Models, France*, 2007.
3. 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.
4. T. Eymann, M. Reinicke, O. Ardaiz, P. Artigas, F. Freitag, and L. Navarro, "Decentralized Resource Allocation in Application Layer Networks," in *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid, Tokyo, Japan*, 2003.
5. D. Abramson, R. Buyya, and J. Giddy, "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker," *Future Generation Computing System*, vol. 18, no. 8, pp. 1061–1074, 2002.
6. S. Venugopal, K. Nadiminti, H. Gibbins, and R. Buyya, "Designing a Resource Broker for Heterogeneous Grids," *Software-Practice and Experience*, vol. 38, no. 8, pp. 793–826, 2008.
7. E. Huedo, R. Montero, I. Llorente, D. Thain, M. Livny, R. van Nieuwpoort, J. Maassen, T. Kielmann, H. Bal, G. Kola *et al.*, "The GridWay Framework for Adaptive Scheduling and Execution on Grids," *Software-Practice and Experience*, vol. 6, no. 8, 2005.
8. 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.

9. J. Altmann, M. Ion, A. Adel, and B. Mohammed, "A Taxonomy of Grid Business Models," in *Proceedings of the 4th International Workshop on Grid Economics and Business Models, Rennes, France*, 2007.
10. H. Eriksson and M. Penker, "Business Modeling with UML: Business Patterns at Work, John Wiley&Sons," 2001.
11. X. Chu, K. Nadiminti, C. Jin, S. Venugopal, and R. Buyya, "Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications," in *Proceedings of the 3th IEEE International Conference on e-Science and Grid Computing*, 2007, pp. 10–13.
12. J. Shirazi, *Java performance tuning*. O'Reilly Media, Inc., 2003.
13. P. Padala, C. Harrison, N. Pelfort, E. Jansen, M. Frank, and C. Chokkareddy, "OCEAN: The Open Computation Exchange and Arbitration Network, A Market Approach to Meta Computing," in *Proceedings of the 2nd International Symposium on Parallel and Distributed Computing, Ljubljana, Slovenia*, 2003.
14. 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, NV, USA*, 2004.
15. Amazon, "Amazon. Elastic Compute Cloud (EC2)," http://www.amazon.com/ec2/., 2009.

Author/s:
Garg, SK;Vecchiola, C;Buyya, R

Title:
Mandi: a market exchange for trading utility and cloud computing services

Date:
2013-06-01

Citation:
Garg, S. K., Vecchiola, C. & Buyya, R. (2013). Mandi: a market exchange for trading utility and cloud computing services. JOURNAL OF SUPERCOMPUTING, 64 (3), pp.1153-1174. https://doi.org/10.1007/s11227-011-0568-6.