

# Mercatus: A Toolkit for the Simulation of Market-Based Resource Allocation Protocols in Grids

Daniel Grosu and Umesh Kant

Department of Computer Science,  
Wayne State University, 5143 Cass Avenue,  
Detroit, MI 48202, USA  
dgrosu@cs.wayne.edu  
ap5651@wayne.edu

**Abstract.** Grid technologies enable the sharing and coordinated use of diverse resources distributed all over the world. These resources are owned by different organizations having different policies and objectives which need to be considered in making the resource allocation decisions. In such complex environments market-based resource allocation protocols are a better alternative to the classical ones because they take into consideration the policies and preferences of both users and resource owners. The only suitable solution for investigating the effectiveness of these resource allocation protocols over a wide range of scenarios with reproducible results is to consider simulations. Thus in this paper we present Mercatus, a simulation toolkit that facilitates the simulation of market-based resource allocation protocols. We describe the model and the structure of Mercatus and present experimental results obtained by simulating three types of auction-based resource allocation protocols.

## 1 Introduction

Grid technologies enable the sharing and coordinated use of diverse resources distributed all over the world [1]. Resources may provide computing services, data storage services, or may be sensors providing data capture services. The resources in a grid environment are typically heterogeneous and are operated by their owners under different policies. Moreover users and resource owners have different objectives, sometimes contradictory. The resource management mechanisms used in traditional computing systems cannot be simply applied to these complex environments because they assume complete control over resources. Thus we need new resource allocation protocols that take into account the objectives of both users and resource owners. The solution is to consider market-based resource allocation protocols [2, 3] which are based on trading and resource brokering policies between resource owners and users. These protocols are suitable for grid environments because of their decentralized structure and the use of incentives for resource owners to contribute resources.

There exist two broad categories of market-based models for resource allocation in grids: commodities markets and auctions [2]. In the *commodities markets model* various services provided by the resource owners are treated as interchangeable commodities. The resource owners determine the price for their services and charge users depending on the amount of resource they consume. In the *auction model* resource owners auction resources using different types of auction mechanisms for establishing the price. Auctions are easier to implement than commodities markets models because they require very little knowledge about the global price. Auctions can be classified into two classes depending on the type of interactions between sellers and buyers. In *one-sided auctions* bids are submitted by the grid users to a central auctioneer. The auctioneer decides the winner based on different auction mechanisms. Examples of one-sided auctions are: First-price auction (the highest bidder wins and pays the amount he bid) and Vickrey auction (the highest bidder wins and pays an amount equal to the second highest bid) [4]. In *two-sided auctions*, also called double auctions, both users and resource owners submit bids. To distinguish the bids we call ‘asks’ the bids submitted by the resource owners. The selling price and the users and the resource owners that trade are decided by the central auctioneer according to different types of double auction mechanisms.

The only suitable solution for investigating the effectiveness of these resource allocation protocols over a wide range of scenarios with reproducible results is to consider simulations. Thus in this paper we present Mercatus, a simulation toolkit which provides a set of core abstractions and functionalities which allow researchers to easily build simulators for the study of market-based resource allocation protocols in grids.

**Related Work.** There exist a large body of work on economic-based resource management models in distributed systems [3, 5, 6]. For a comprehensive survey of economic models for resource management see [2]. A number of research projects addressed the simulation of scheduling and resource allocation protocols in grids [7, 8, 9]. We discuss here three projects, SimGrid [9], GridSim [8] and OptorSim [7] which are closely related to our work.

SimGrid [9], developed at UCSD, is a simulation toolkit which targets the simulation of specific application domains and computing environments topologies. SimGrid’s C API provides functions for generation of resources and tasks, and functions to schedule and unschedule tasks on resources. The main difficulty when simulating market-based resource allocation protocols using SimGrid comes from the fact that there is no support for implementing market-based allocation protocols and the time accounting during the execution of these protocols is not implicit. In Mercatus time accounting is implicit from the design of the toolkit and it does not require special actions from the user.

GridSim [8], developed at the University of Melbourne, is a Java-based simulation toolkit which supports the simulation of application scheduling on heterogeneous Grid resources. GridSim API provides methods for task creation, resource management and scheduling. The main difficulty when simulating auction-based resource allocation protocols comes from the fact that the resource broker

which acts on behalf of the user and performs scheduling of user's work on suitable resources, does not support mechanisms for continuous bidding as required in the case of auction-based protocols. Also, the base class in GridSim does not have support for implementing auctions. Thus in our simulation toolkit we provide a grid market auctioneer class which supports registration of resources and conducts auctions supporting various bidding policies.

OptorSim [7], developed within the EU Data Grid project, is a Java based simulator which allows the simulation of data replication algorithms in data grids. It provides support for simulating peer to peer auctions for optimal replica selection. The main focus is on data transfer and replication and not on job execution.

**Our Contributions.** In this paper we introduce a new simulation toolkit, Mercatus, which provides basic constructs necessary to building simulators for the study of market-based resource allocation protocols in grids. Currently, Mercatus supports the simulation of several types of resource allocation protocols based on one-sided and two-sided auctions. It allows the study of economic efficiency in terms of costs, profits and payments and the study of system efficiency in terms of execution time and resource utilization.

**Organization.** In Section 2 we present the simulation model and the resource allocation protocols. In Section 3 we describe the design and structure of Mercatus and show how the resource allocation protocols are implemented. In Section 4 we show how to build simulations using Mercatus. In Section 5 we present several simulation experiments showing the capabilities of our simulation system. In Section 5 we draw conclusions and present future work.

## 2 Mercatus Simulation Model

Mercatus assumes a grid system model in which there exist several heterogeneous resources each offering services and several users that consume these services. The resources are characterized by the following parameters:

- (i) *Processing rate*: It is given in million instructions per seconds (MIPS).
- (ii) *Reservation price*: It is defined as the minimum price accepted by a resource for one second of task execution.
- (iii) *Cost*: Represents the cost incurred by a resource for one second of task execution.

Users are characterized by the following parameters:

- (i) *Work*: It is defined as the total amount of work in millions of instructions for all the tasks of a user.
- (ii) *Number of tasks*: Represents the number of tasks of one user.
- (iii) *Budget*: It is the maximum amount of 'grid dollars' (G\$) a user can pay to resources for executing his tasks.

The goal of the users is to finish their work at the earliest time and to pay as little as possible out of their budget, whereas the resources are concerned with maximizing their utilization and hence their profit. The price at which the trade

takes place is decided by the auction-based protocols. In Mercatus these auctions are conducted by a Grid Market Auctioneer (GMA). Depending on the type of auction, users or/and resources send bids to GMA. GMA decides the winner depending on the auction mechanism used and declares the auction results to users so that a winning user can send his task to be processed at an appropriate resource. Auction results are also sent to resources so that they can identify the winning user. The GMA keeps on conducting auctions until there is no user with any task left. A user continuously participates in consecutive auctions until he has no tasks left or he exhausted his budget. A resource continuously participates in the auction until there is no user left with any tasks not executed. GMA, users and resources are continuously interacting with each other. Mercatus does not support the simulation of data transfers and data replications.

Mercatus allows the programmer to specify the bidding policy of each grid user. The bidding policy can be constant or variable (depending on the budget remaining, deadline, resource processing rate and a percentage increase specified by the programmer). Also in two-sided auctions the asks of resources can be constant or variable (a resource can decrease its asks if it didn't win in the last auction). Currently, Mercatus supports three types of resource allocation protocols based on First-price, Vickrey [4] and Double auctions [10]. All these protocols are implemented as part of the GMA. All the bids submitted by the users are for a second of task execution at a given resource. In the following we briefly describe these protocols.

**First Price Auction Protocol (FPA):** This protocol is run by GMA in behalf of one resource. Users participating in this protocol bid without knowing what the bid values of the other users are. The user who bids the highest wins the auction and pays the amount he bid. The winning user sends the task to the resource involved in the auction.

**Vickrey Auction Protocol (VA):** This protocol is run by GMA in behalf of one resource. This protocol is based on Vickrey auction which is also called the second-price auction. Users participating in this protocol bid without knowing what the bid values of the other users are. The user who bids the highest wins the auction and pays an amount equal to the second highest bid. The winning user sends the task to the resource involved in the auction.

**Double Auction Protocol (DA):** Users send bids for a group of resources where each resource has the same characteristics. We assume that  $m$  users decided to participate in a double auction for a group of  $n$  resources. After GMA collects all the bids  $\{b_1, b_2, \dots, b_m\}$  and all the asks  $\{a_1, a_2, \dots, a_n\}$ , it does the following:

- (i) Sorts bids in decreasing order and asks in increasing order:
 
$$b_{\pi(1)} \geq b_{\pi(2)} \geq \dots \geq b_{\pi(m)}$$

$$a_{\sigma(1)} \leq a_{\sigma(2)} \leq \dots \leq a_{\sigma(n)}$$
 where  $\pi$  and  $\sigma$  are the permutations defining the orders statistics above.
- (ii) Finds  $k$  such that  $b_{\pi(k)} \geq a_{\sigma(k)}$  and  $b_{\pi(k+1)} < a_{\sigma(k+1)}$ .
- (iii) Determines the trading price,  $t = \frac{1}{2}(b_{\pi(k+1)} + a_{\sigma(k+1)})$

- (iv) If  $a_{\sigma(k)} \leq t \leq b_{\pi(k)}$  notifies resource  $\sigma(i)$  and user  $\pi(i)$ ,  $i = 1, 2, \dots, k$ , that they can trade at price  $t$ .
- (v) If  $t \geq b_{\pi(k)}$  or  $t < a_{\sigma(k)}$  notifies resource  $\sigma(i)$  and user  $\pi(i)$ ,  $i = 1, 2, \dots, k-1$ , that they can trade. Each resource gets  $a_{\sigma(k)}$ , and each user pays  $b_{\pi(k)}$ .
- (vi) GMA sends reject messages to resources and users that do not trade.

Users that trade send tasks to the corresponding resources and they execute them. After receiving the results of the execution, users send payments to the corresponding resources. If the condition in (v) holds, resource  $\sigma(i)$  receives  $a_{\sigma(k)}$  and user  $\pi(i)$  pays  $b_{\pi(k)}$ , for  $i = 1, 2, \dots, k-1$ . As a result of this trade there is a surplus of  $(k-1)(b_{\pi(k)} + a_{\sigma(k)})$ . We assume here that this surplus is kept by GMA which plays the role of a budget balancer. We also assume that  $b_{\pi(m+1)}$  is the lowest possible valuation of the users,  $a_{\sigma(n+1)}$  is the highest possible valuation of the resources and that  $b_{\pi(m+1)} < a_{\sigma(n+1)}$  holds.

### 3 Architecture and Implementation of Mercatus

Mercatus is a toolkit for the simulation of market-based resource allocation protocols in grid environments. It is a discrete event simulation toolkit written in Java, based on the SimJava discrete event model [11]. We will first briefly describe the SimJava package and then present the architecture and implementation of Mercatus.

**SimJava Simulation Package.** SimJava simulation package [11], implements a discrete event model suitable for implementing general simulations. Events are used at implementation level to model the behavior of respective entities in terms of interactions. The entities interact with each other by passing events. These events may be just simple notifications or may carry additional information related to the notification. Each entity runs in a separate Java thread and has a `body()` method which encodes its behavior. These threads run concurrently and entities communicate to each other through ports. The `Sim_system` object is responsible for initializing the simulation environment and linking the various entities through their ports. It also takes care of the events ordering by guaranteeing that the events dispatched by various entities are received by other entities in the same order. `Sim_system` also takes care of time accounting. More details about SimJava discrete event model can be found in [11].

**Mercatus Architecture.** There are three major players in our grid simulation model: users, resources and GMA. We model each of these three grid players as SimJava entities, where `User` entity represents users, `Resource` represents resources, and grid market auctioneer, `GMA` is the trusted party which resolves the trading between users and resources. Each entity has two ports, ‘in’ and ‘out’. An entity uses the ‘in’ port to receive events from other entities and the ‘out’ port to send events to other entities.

We describe the sequence of interactions among `User`, `Resource` and `GMA` entities and the events passed during these interactions. The events being passed

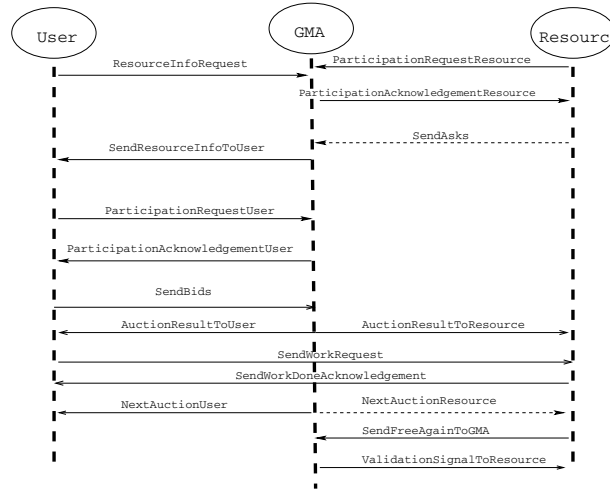


Fig. 1. Events passed between entities

can contain data corresponding to that particular event. The sequence of interactions for *one-sided auctions* is as follows:

*Phase I: Information collection and participation decisions*

1. Resources notify **GMA** of their willingness to participate in an auction by sending the **ParticipationRequestResource** event. The resource information is sent to GMA together with this event. GMA acknowledges the receipt of resource information by sending **ParticipationAcknowledgementResource**.
2. Users request information about resources willing to participate in auctions by sending **ResourceInfoRequest**. GMA provides resource information to users by sending **SendResourceInfoToUser**. Users receive the information and after analyzing it they decide to participate in some of the auctions. A user notifies **GMA** about his decision to participate by sending **ParticipationRequestUser** to GMA. After the event **ParticipationAcknowledgementUser** is received by the user, he is ready to participate in the auction.

*Phase II: Conducting the Auction*

1. Participating users send bids by using the event **SendBids**.
2. **GMA** runs the auction and notifies **User** (by sending **AuctionResultsToUser**) and **Resource** (by sending **AuctionResultToResource**) about the result of the auction.
3. The winning user sends the task to the corresponding resource using **SendWorkRequest**. The resource acknowledges the receipt of the work by sending **SendWorkDoneAcknowledgment**.
4. A new auction starts which involves the resources that are idle and willing to participate. **GMA** informs users and resources that it is ready to start the next auction by sending **NextAuctionUser**. **SendFreeAgainToGMA** is used by

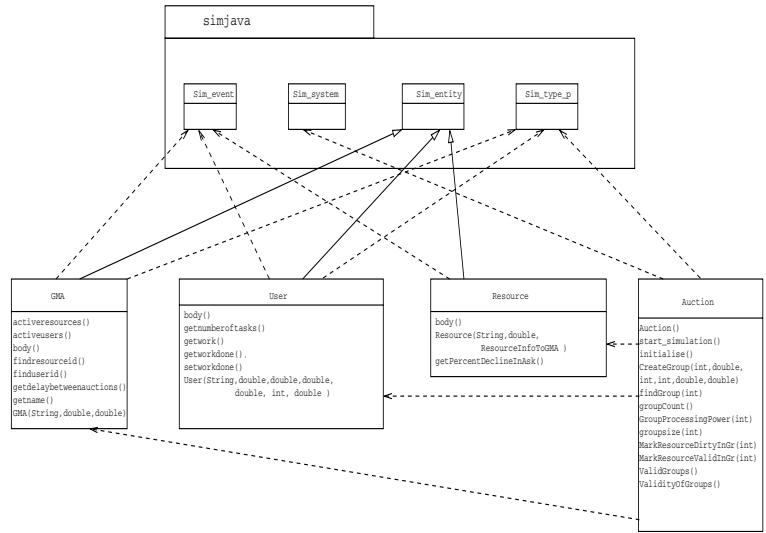


Fig. 2. Mercatus class diagram

a resource to inform GMA that it is idle again and will participate in the next auction. **ValidationSignalToResource** acknowledges that the resource is validated and it can participate in the next auction.

The sequence of event passing is shown in Figure 1. In case of the two-sided auctions two other events are needed, **SendAsks** and **NextAuctionResource**. They are represented in Figure 1 using dotted lines.

The main SimJava classes used to build the Mercatus toolkit are: **Sim\_system**, **Sim\_entity**, **Sim\_event**, and **Sim\_type\_p**. Based on these classes provided by SimJava our toolkit implements nine classes. We can categorize these nine classes into three groups: (i) **Auction** class (main class), (ii) entity classes, and (iii) wrapper classes. In Figure 2 we present the class diagram of the Mercatus toolkit.

(i) **Class Auction**: This class is responsible for running the simulation. The **initialise()** method initializes the simulation and defines various parameters such as the number of users and the number of resources. It also declares objects of these entities each with different bidding policy parameters. The Auction class provides a method for starting the simulation (**start\_simulation()**) and methods for managing groups of resources: **CreateGroup()**, creates a group of resources all having the same characteristics; **findGroup()**, finds and returns the group number of a resource with a given id; **groupcount()**, returns the total number of groups of resources; **GroupProcessingPower()**, returns the processing power of a resource belonging to a group; **groupsize()**, returns the number of resources belonging to a particular group; **MarkResourceDirtyInGr()**, marks resource availability as false; **MarkResourceValidInGr()**, marks resource availability as true; **ValidGroups()**, returns the number of valid group of resources;

and `ValidityOfGroup()`, returns true or false depending on whether the group has at least one resource available or not.

(ii) *Entity classes*: These classes model the users, resources and GMA.

**Class User**: The `body()` method is used to model the behavior of `User`. This method is inherited from `Sim_entity` class of `SimJava`. This class provide methods for: getting the number of tasks of a user, `getnumberoftasks()`; getting and updating the work completed, `getworkdone()` and `setworkdone()`; and getting the total work of a user, `getwork()`.

**Class GMA**: The `body()` method is used to model the behavior of `GMA`. This method is inherited from `Sim_entity` class of `SimJava`. `GMA` provides methods for: getting the number of resources currently available, `activeresources()`; getting the number of users currently willing to participate, `activeusers()`; finding a resource in the list of all resources, `findresourceid()`; finding a user in the list of all users, `finduserid()`; returning the delay between two successive auctions (the delay is set in the constructor of `GMA`), `getdelaybetweenauctions()`; and getting the name of `GMA`, `getname()`.

**Class Resource**: The `body()` method is used to model the behavior of the resource. This method is inherited from `Sim_entity` class of `SimJava`. It provides a method for getting the percent decline parameter in asks of a resource, `getPercentDecline InAsk()`.

(iii) *Wrapper classes*: We implemented several wrapper classes which are used to pass information between entities along with an event. Due to space limitation we are not able to describe these classes here.

## 4 Building Simulations with Mercatus

In Figure 3 we present a simple example of Mercatus code fragment in which we simulate ten resources and five users. This example shows that Mercatus is a very efficient tool and it is very simple to use.

The first step of any simulation is to initialize the simulation environment by calling `initialise()` method with the following parameters: auction type, num-

```

public static void main(String[] args) {

    Auction.initialise(0, 5, 10, 25, 0, 0.1 );

    Auction.addUser("User0", 4.2, 5, 100, 4, 56);
    Auction.addUser("User1", 6.5, 6, 150, 4, 80);
    Auction.addUser("User2", 3.0, 4, 180, 5, 70);
    Auction.addUser("User3", 4.0, 10, 100, 3, 61);
    Auction.addUser("User4", 4.25, 5, 100, 5, 76);

    Auction.CreateGroup(6, 50, 400, 30);
    Auction.CreateGroup(4, 100, 800, 80);

    Auction.start_simulation();

}
}

```

**Fig. 3.** Sample code fragment for creating simulations

ber of resources, number of users, number of rounds, bidding policy and network latency. Next we add as many users as we specified in the above step by calling `addUser()`. The parameters of `addUser()` method are: user name, total work, deadline, budget, number of tasks, and percentage increase of bids. Resources are added by calling `CreateGroup()` method once for each group of resources with identical characteristics. The parameters of this method are: number of members, processing rate, cost and percentage decrease in asks. The total number of resources over all groups should be equal to the total number of resources specified in the `initialise()` method. Finally, we call `start_simulation()` which links the ports of all entities and starts the simulation.

## 5 Simulation Experiments

In order to show the capabilities of Mercatus we present a case study in which we simulated a grid environment consisting of 15 resources shared by 10 users. The resources are divided into two groups. The parameters of users and resources are given in the Table 1 and Table 2. The work of each user is in millions of instructions. Members indicate the number of resources belonging to the group. In these experiments we assume that the reservation price is equal to the resource cost. All the simulation experiments are run for 25 auction rounds and considering a network latency of  $10^{-5}$  seconds.

Mercatus facilitates the evaluation of market-based resource allocation protocols in terms of their economic efficiency by reporting user spending, resource cost and resource profit. It also allows the study of system efficiency in terms of execution time and resource utilization. In the following simulation study we present a number of plots showing some of these metrics.

**Resource Profit.** In Figure 4 we present the profit of each resource as percentage of the cost. For most of the resources the profit obtained in VA is less than that in FPA. This is because in VA the winning user pays an amount equal to the second highest bid, while in FPA the winning user pays an amount equal to the highest bid. The exceptions are resources 4, 10, 11 and 12 which gain a higher profit than in FPA. This is because resources register in random order and the order of resources in FPA and VA is not the same.

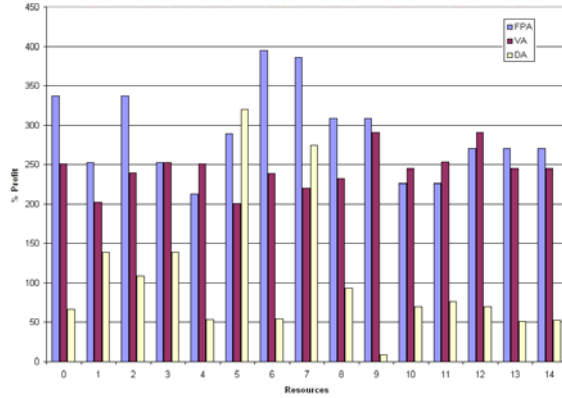
In DA we observe that resource 9 gains a very low profit. This is because resource 9 won three times during the 25 rounds, but every time it won according to the second case of DA protocol where the seller gets  $a_k$ . Resources 5 and 7 gain high profit in DA because both resources won only one task each during the 25 rounds of DA. Furthermore they won according to the first case of DA

**Table 1.** Users' parameters

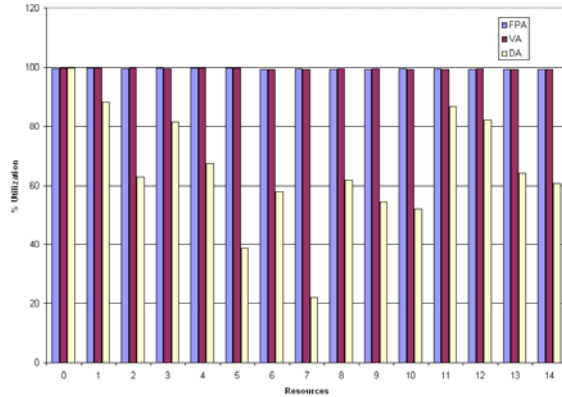
User	0	1	2	3	4	5	6	7	8	9
Work	4.2	6.5	3.0	4.0	4.25	5.9	4.0	5.0	4.6	3.1
Budget	100	150	180	100	100	140	100	160	200	140
Number of tasks	4	4	5	3	5	5	6	7	4	5

**Table 2.** Resources' parameters

Group	Members	Processing rate (MIPS)	Cost (\$G)
0	6	50	400
1	9	100	800



**Fig. 4.** Resource Profit



**Fig. 5.** Resource Utilization

protocol where winner gets  $t$ . Since the budget of the users is high the value of the trading price  $t$  is also high.

**Resource Utilization.** The resource utilization for each resource in the system is presented in Figure 5. We observe that for FPA and VA the utilization is very high because during most of the auction rounds of these protocols all available

resources get a user task. In the first auction one of the users has seven tasks, so seven out of fifteen resources will get a task. In the next auction these seven resources will be busy servicing user tasks, but remaining eight resources will be available and will get and execute other user tasks. In this way during most of the auction rounds the number of available resources is less than the total number of tasks a user has and hence during most of the auctions either a resource is winning a task or is servicing a user task it won during the last auction. Thus the resource utilization is high in case of FPA and VA.

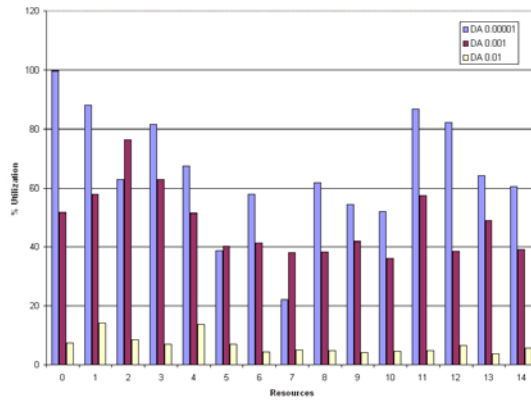


Fig. 6. Effect Of Network Latency On Resource Utilization

For DA the resource utilization is not always high and is less than that of FPA and VA. This is because in case of DA at least two resources should be in a group for the DA protocol to be applied. Also DA protocol is based on the equilibrium of buyers bid and sellers ask and any resource which is beyond this equilibrium point does not win the auction. So in case of DA it is not guaranteed that a resource will win a task even if there are sufficient users.

**Effect of Network Latency on Resource Utilization.** In Figure 6 we present the resource utilization for different network delay values and considering the DA protocol. In most of the cases the resource utilization decreases as network latency increases. This is because more time is spent in communication as compared to the time spent executing tasks. There are two exceptions at resources 2 and 7 where the utilization is high even though the network latency is increased from  $10^{-5}$  to  $10^{-2}$  seconds. This is because DA requires that at least two resources should be available and as latency is high it is highly probable that during the next auction more resources will be available. Hence it is more probable that for each auction there are enough resources to apply DA and thus utilization is increased.

## 6 Conclusion

In this paper we introduced Mercatus, a simulation toolkit that facilitates the simulation of market-based resource allocation protocols. We described the model and the structure of Mercatus and presented experimental results obtained by simulating three types of auction-based resource allocation protocols. In the future versions of Mercatus we will provide support for implementing other market-based resource allocation protocols (e.g. combinatorial auctions, commodities markets) and also more features that will make it more flexible. Mercatus v1.0 has been released and it is available at <http://mercatus.cs.wayne.edu>.

## References

1. Foster, I., Kesselman, C.: *The Grid: A Blueprint for a New Computing Infrastructure*. 2nd edn. Morgan Kaufmann, San Francisco, CA (2003)
2. Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic Models for Resource Allocation and Scheduling in Grid Computing. *Concurrency and Computation: Practice and Experience*. **14** (2002) 1507-1542
3. Wolski, R., Plank, J. S., Brevik, J., Bryan, T.: Analyzing Market-based Resource Allocation Strategies for the Computational Grid. *Int. J. of High Performance Computing Applications*. **15** (2001) 258-281
4. Vickrey, W.: Counterspeculation, Auctions, and Competitive Sealed Tenders. *J. of Finance*. **16** (1961) 8-37
5. Abramson, D., Buyya, R., Giddy, J.: A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. *Future Generation Computing Systems*. **18** (2002) 1061-1074
6. Gomoluch, J., Schroeder, M.: Market-based Resource Allocation for Grid Computing: A Model and Simulation. *Proc. of the 1st Int. Workshop on Middleware for Grid Computing*. June (2003) 211-218
7. Bell, W. H., Cameron, D. G., Capozza, L., Millar, A. P., Stockinger, K., Zini, F.: OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *Int. J. of High Performance Computing Applications*. **17** (2003) 403-416
8. Buyya, R., Stockinger, M.: GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience*. **14** (2002) 1175-1220
9. Casanova, H.: SimGrid: A Toolkit for the Simulation of Application Scheduling. *Proc. of the IEEE/ACM Int. Symp. on Cluster Computing and the Grid*. May (2001) 430-437
10. McAfee, R. P.: A Dominant Strategy Double Auction. *J. of Economic Theory*. **56** (1992) 434-450
11. Howell, F., McNab, R.: SimJava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modeling. *Proc. of the 1st Int. Conf. on Web-based Modeling and Simulation*. January (1998) 252-259