

# Two Auction-Based Resource Allocation Environments: Design and Experience

# Two Auction-Based Resource Allocation Environments: Design and Experience*

Alvin AuYoung, Phil Buonadonna, Brent N. Chun, Chaki Ng,
David C. Parkes, Jeff Shneidman, Alex C. Snoeren, Amin Vahdat

February 20, 2008

## 1 Introduction

Many computer systems have reached the point where the goal of resource allocation is no longer to maximize utilization; instead, when demand exceeds supply and not all needs can be met, one needs a policy to guide resource allocation decisions. One natural policy is to seek *efficient* usage, which allocates resources to the set of users who have the highest utility for the use of the resources. Researchers have frequently proposed market-based mechanisms to provide such a goal-oriented way to allocate resources among competing interests while maximizing overall utility of the users.

The emergence of several large-scale, federated computing infrastructures [6, 2, 8, 4, 49], creates an opportunity to revisit the use of market mechanisms to allocate computing resources. There are several common characteristics of federated systems that make a market-based approach appropriate. First, the end-users of these systems commonly have synchronized demand patterns (e.g., due to shared conference deadlines or other external events), which makes excess resource demand a common occurrence. Second, resource ownership is shared by the community, and therefore, there are many parties with a stake in the allocation policies used by the system. Finally, there is a large number of resources, which means that resource contention cannot be resolved by human administration or simply by the result of an uncoordinated process. An allocation system designed around a market can lead to a socially optimal allocation outcome to all users, which is particularly important during times of heavy resource contention. Markets also create a natural structure for globally desirable incentives and user behavior and can relieve system administrators from a significant management burden.

In general, however, market-based allocation schemes have yet to catch on in practice. To understand why applying a market-based design to these systems is a challenging problem, and perhaps why markets have not been easily applied in the past, one can observe ways in which these computing environments differ from traditional markets. First, computing systems lack several natural market components. Since end-users typically do not engage in the purchase or production of resources, they are not typically required to use currency or otherwise barter for resource use. Therefore, many users are not accustomed to– or comfortable with –determining their utility for a resource in terms of currency. Also missing in these systems is a means of resource production. Since the supply of resources is often fixed– due to external constraints –and cannot easily be summarily increased or decreased in response to global demand, both under-demand and over-demand of resources are common. A second difference between computing systems and markets relates to the behavior of participants. Rather than engage in strategic, sophisticated behavior (which can require costly modeling of other participants), end users in systems may prefer the simple interfaces and allocation policies of today's systems (e.g., FCFS, best-effort), rather than needing to submit bids and otherwise provide richer information in order to achieve what may sometimes amount to only a slight utility gain [15].

*This chapter is based on the authors' following publications [44, 9, 17, 34]

We present two case studies of market-based mechanisms that were developed to support sophisticated usage policies in real computing environments, and designed to address some of these challenges. First, we present *Bellagio*, a market-based resource allocator deployed for 6 months on the worldwide PlanetLab research network [6]. Second, we describe *Mirage*, an allocator deployed on a shared sensor network testbed [25], which is still in use as of this writing. We discuss our experiences with each system, and focus on both the initial challenges these systems were designed to address, and the empirical successes and difficulties that we have observed since deployment. A key contribution of these systems is their user-centric design. Both Mirage and Bellagio use a resource discovery interface and a combinatorial bidding language to allocate resources; the former allows users to easily communicate domain-specific resource demand, and the latter allows users to express needs on bundles of resources, which is important in each of our applied domains. Both systems also use a virtual currency policy that attempts to create incentives similar to those created by currency in a real market economy.

Deployments such as these appear vital in understanding both the potential and limitations of market-based resource allocation in computer systems. While our deployments were mostly successful, our experience also highlights a few persistent challenges with applying markets to these systems. For example, the perishable nature of computer resources influenced the use of a sliding window-based allocation mechanism. This formulation has side effects of making the system unusable for a portion of users, and also allowing it to be manipulated by others. In addition, choosing a level of system transparency to increase the usability of market mechanisms also allowed users to exhibit globally undesirable market behavior. We conclude with a set of suggestions to help guide future deployments of market-based resource allocation systems.

## 2 Background

During the past decade, large-scale, federated computing infrastructures have emerged as the primary means for delivering significant network and computational resources. By time-sharing communal resources, users can harness the power of an otherwise unobtainable set of resources to perform a wide variety of tasks. Computation in grid and cluster environments [22, 8, 7, 3, 20], wide-area distributed service development and deployment [6, 2] of network testbeds and development in sensor networks [4, 49] are all examples of common tasks in these emerging environments. Resource allocation continues to be the fundamental challenge in these systems.

**Traditional allocation approaches.** Perhaps the most natural resource allocation scheme in *time-shared* systems is proportional share, which provides equal, simultaneous access to time-shared resources to all users. In this model, multiple applications can run on a machine simultaneously. It is well known, however, that proportional-share scheduling alone does not function well in systems where over-demand for resources is common. In particular, as demand for resources increases, the time-share per resource received by each user decreases, while overheads remain constant (or, in some cases, increase due to thrashing). In the end, the utility delivered to each user goes to zero as the number of competing users increases.

Batch scheduling is commonly used to schedule jobs in *space-shared* systems, such as supercomputers and grid computing environments. As opposed to time-shared resources, space-shared systems are well suited for applications which require a large share of a system's resources (e.g., a CPU-intensive job), and therefore, run a single application on a machine at a time. Most space-shared systems schedule jobs to optimize system-level metrics such as total throughput or average job-turnaround time. These approaches avoid the thrashing that can occur with high resource contention that occurs in proportional share schemes. However, these systems typically do not account for the heterogeneity of users in their needs in regard to scheduling jobs. For example, simple batch scheduling systems cannot distinguish between users with jobs that require timely completion and users with jobs that do not. More sophisticated schedulers like Maui [26] do include multiple priority queues, but lack an automated way to prevent users from artificially inflating job priority. Also, the number of available job priorities is usually fixed, which restricts the amount of utility information a user can express for a job. These systems offer no way to resolve contention for resources in the face of heavy demand—conflicts are generally resolved through human intervention.

**Job utility functions.** Maximizing aggregate *utility*, or value delivered by the system, requires eliciting user preferences for resources over time and space [32]. The fundamental shortcomings of existing approaches are rooted in the fact that they do not explicitly consider a user's utility for a job when making a scheduling decision. Consider the example of a student using a computing cluster who has a set of jobs associated with various course projects. The student has an internal prioritization for the relative importance of each job, based on the importance of each course, and the various deadlines for each project. Therefore, she may be able to carefully craft her job submissions to create an ideal (i.e., utility-maximizing) execution sequence. But, if she is sharing the computing cluster with a group of like-minded students– each with their own job utility functions –it is unlikely that the entire set of students can coordinate to craft a mutually agreeable schedule. A study of over one hundred users at the San Diego Supercomputing Center (SDSC) supports the notion that real jobs exhibit sophisticated utility information [32], information that cannot be explicitly communicated to the scheduling system. Despite the fact that SDSC uses a sophisticated priority-based scheduler, the study highlights the fact that without explicitly considering the utility function of a job, a scheduling system cannot make automated decisions that maximize the overall utility of its users. A job utility function captures the value a job has to its user, as well as how this value can vary over time, or other dimensions.

The idea of scheduling based on per-job utility functions is a well-trodden field: the Alpha OS [19] handles time-varying utility functions; Chen and Muhlethaler [14] discuss how to do processor scheduling for them; Lee et al. [31] look at trade-offs between multiple applications with multiple utility-dimensions; Muse [13] and Unity [48] use utility functions for continuous fine grained service quality control; and Petrou et al. [40] describe using utility functions to allow speculative execution of tasks. Siena [10] considers the impact of user aggregate utility functions to control service provider behavior across multiple jobs. Kumar et al. [29] uses a single utility function to aggregate data from multiple sources. These prior works assume the existence of mechanisms to elicit honest and comparable utility functions from a user. Similar to priority job schedulers, a shared scheduling system needs a mechanism to prevent self-interested users from artificially overstating their job utility.

**Economic approaches.** A well-designed economic approach to resource allocation can provide a natural framework for users to communicate their job utility functions, and for the system to elicit true utility functions from users. The basic idea is that resources are priced by looking to balance global supply and demand, with users required to purchase resources using a personal budget of currency. In this case, resource prices force users to self-select a resource allocation based on their utility for jobs, with the implication being that the resources are allocated to the jobs that value them the most. Because users must provide payment (perhaps in a virtual rather than real-world currency) in return for the allocation of jobs, users will tend to be straightforward in describing their utility (measured in units of currency) for different allocations.

Market-based systems have long been proposed as solutions for resource allocation in these environments, yet integration of these systems into real computer systems has garnered little momentum. A number of previous efforts have explored market-based approaches for resource allocation in computer systems. The earliest work that we are aware of is Sutherland's futures market for CPU time on Harvard's PDP-1 [46]. Subsequent work can be categorized into two-groups: pricing mechanisms and auction-based mechanisms. In pricing mechanisms, the system determines a price for each resource by using a combination of various signals, such as load and resource demand history; users acquire resources by purchasing them at market prices. In auction-based mechanisms, the system generates prices for resources dynamically by first collecting bids from users on resources that are then allocated, typically to maximize revenue to the bid-taker. Past efforts on market mechanisms work has been largely dominated by auction-based schemes and has been applied to resource allocation in a broad range of distributed systems including clusters [47, 18], computational grids [50, 30], parallel computers [45], and Internet computing systems [33, 41].

As with these systems, our designs (Bellagio and Mirage) also rely on an auction to allocate resources. An auction is an appealing model for dynamic domains in which resource demands can be expected to fluctuate rapidly [16]. Recognizing that users of large-scale computing environments often require the use

of combinations of resources [16, 21], we adopt the model of a *combinatorial* auction, in which users bid on bundles of resources as opposed to individual resources. This ability to bid on resource combinations in space and time allows users to more accurately express their preferences on different resources. For example, a user can express that some resources are interchangable ("substitutes") and some are required together ("complements"). To the best of our knowledge, Bellagio and Mirage are the first deployed systems that support allocation of combinations of heterogeneous computational resources via flexible auction methods.

Of course, there do exist many alternatives to the use of markets in serving the purpose of resource allocation in computer systems. These include first-come first served (FCFS) allocation and also *reservation* systems that permit scheduling into the future. Despite our general belief in the importance of market based methods, the ultimate success of a system is measured in usage, and usage depends on a number of factors that are easy to overlook. For example, ease of use may trump mechanism features. People may be willing to accept the limitations of simpler systems if market-based systems are seen as too complex, or if they fail in other ways. This is one reason why it seems especially important to deploy market-based systems and gain real experience. Through such a deployment, one gains a rare opportunity to deploy a market and examine the interactions between participants in an isolated economic system. Uncovering practical issues in a deployed computational market can lead to observations about the need for new theory, which may then help in bringing the benefits of efficient, market-based resource allocation to users of today's large-scale shared infrastructures.

# 3 System Design

In this section, we describe the high-level design used by both Bellagio and Mirage. We discuss both the basic method of allocation, and the virtual currency system used in both systems.

## 3.1 System Architecture

The goal of our market-mechanism is to determine an allocation that maximizes user utility (as measured, for example, in units virtual currency.) If the market is reasonably competitive and bids reflect the true utility of a user for a particular allocation, then by clearing a market to maximize the total bid value the market will tend to achieve this goal of maximizing total utility.

The basic design of both the Bellagio and Mirage systems is that of a repeated, sealed-bid combinatorial auction[1], which allocates resources to competing users over time. Given that the *winner-determination problem* in a combinatorial auction is known to be NP-hard [42, 43], we adopt a greedy algorithm to ensure that the auctions clear quickly irrespective of the number of size of user bids. For this purpose, we adopt a simple heuristic that greedily allocates (also called a "first-fit" schedule) bids in order of a bid's *value density*, which is the bid value divided by its size and length [11]. We repeat this allocation algorithm $k$ times, with a different ordering of the top $k$ bid value densities, and select the best allocation among these $k$ attempts. In our deployment we set $k = 10$. We adopt a first-price auction format rather than an auction with a more complex payment rule, in part because of our intent to make auction clearing as fast as possible.[2]

In this setting, the auction is run periodically. Once bids are received the auction must determine winning bids and the associated resource allocation. A user in our systems submits bids to the auction using a two-phase process (Figure 1). First, she adopts a *resource discovery service* to find candidate resources that meet her needs. Second, using the concrete resources identified from the first step, such a user can place bids using a system-specific bidding language.

[1]The initial deployment of Mirage used an open-bid English auction format, which was later changed to a sealed-bid format.

[2]Furthermore, it is well understood that the second-price, Vickrey style of payment rules do not immediately provide the incentive advantages that they enjoy in static settings when they are adapted to dynamic settings [37].

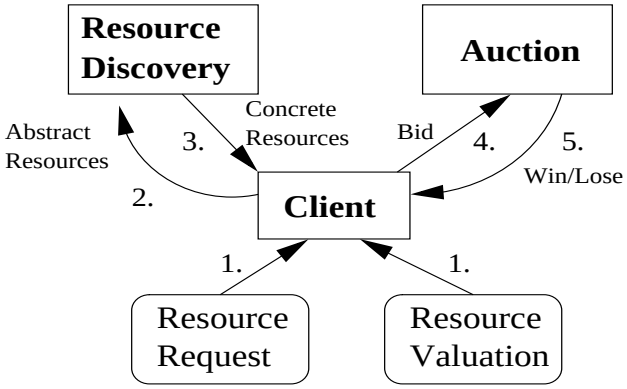


Figure 1: **Bidding and Acquiring Resources.**

### 3.2 Currency

Most existing computing systems lack any notion of currency. Because introducing real currency wasn't a feasible option in our target environments, we have designed a virtual currency system to be used in both Bellagio and Mirage.

The design of a good currency policy will encourage desirable behaviors and serve to discourage undesirable behaviors. The lack of proper policies can render the system useless. For example, if users can obtain large amounts of virtual currency very easily then they will of course bid arbitrarily high values all the time. Such a system reduces to resource allocation based on social conventions, since there is no disincentive for a user to not always bid the maximum possible value permitted within the bidding language. Simply stated, the currency must be "real enough" that users care not to spend it because they will be able to benefit from saving unspent currency for future use. In order to support virtual currency, we rely on a central bank that enforces a currency policy by controlling the aggregate amount and flow of virtual currency in the system.

Since users have no direct way of earning virtual currency, the system must decide how to distribute the currency. Because users enter the system with no virtual currency, we also need to provide new users with some initial amount of currency. In addition, as users spend currency over time, we also need a way to infuse their accounts with new currency. Clearly, there are many virtual currency polices one could employ to meet these requirements and different policies will result in very different economic systems and resource allocations.

Our virtual currency policy is based on two principles: (i) prioritizing users based on an exogenous policy (we will describe the specific policies for Bellagio and Mirage in the next sections); and (ii) penalizing/rewarding users based on usage or lack of usage during times of peak demand. Examples of factors that may affect prioritization include the *types of usage* (e.g., research vs. coursework), the *amount of contribution* made to a system, and *fairness* concerns, which may be defined in terms of the cumulative resource consumption of an individual. Allowing the prioritization metric to be determined exogenously permits flexibility in crafting a good policy for a specific domain. In addition, it seems natural to reward the user who refrains from using the system during times of peak demand (or, more generally, does not waste resources) and penalize the user who uses resources aggressively when resources are scarce. Consider a user who monopolizes the entire system for several days prior to a major deadline, for instance.

Each user is associated with an account at a central bank which stores virtual currency. Each bank account is assigned a baseline amount of currency based on priority, a number of "*currency shares*" which influences the rate that currency will subsequently flow into the account, and is also initialized with a baseline amount of currency. Given an initial currency allocation, users can begin to bid for resources in the auction. Each time the auction clears, trades are settled and revenue is collected from the accounts for winning bids. This currency is then distributed back to all accounts through profit sharing in a

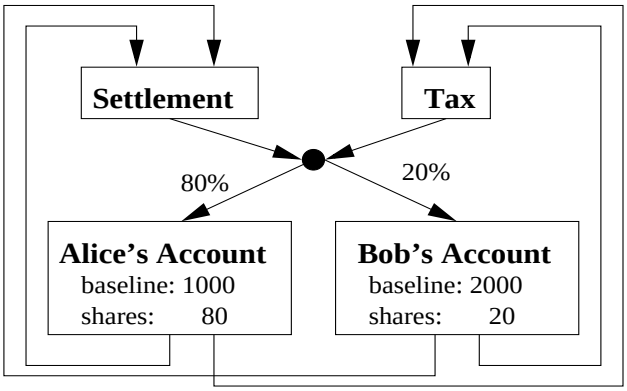


Figure 2: **Virtual currency policy.**

proportional-share fashion based on the number of shares in each account (Figure 2). It is this profit sharing policy that allows users who do not waste resources to save additional currency which can be used for a subsequent burst of activity later.

In addition to profit sharing, the system also imposes a fixed rate savings tax on all accounts that have excess currency above their baseline values, again with proportional-share distribution. The motivation for the savings tax is based on expected resource consumption. For example, in other distributed systems testbeds such as PlanetLab [38], it has been observed [16] that resource consumption is often highly imbalanced with a small fraction of the users consuming the majority of the resources and many users often going idle for long periods of time. Similarly, parallel batch computing systems often exhibit a diurnal load pattern [5] varying between extremes of light utilization and heavy utilization. Assuming similar resource consumption patterns, and in the absence of additional policy, the implication would be that heavy users would eventually be working out of accounts with very little currency even if there is little demand for resources in the system. To mitigate this effect, we impose a fixed rate *savings tax* that makes operational the concept of "use it or lose it" policy employed by agencies such as the Federal Aviation Administration in allocating scarce resources; i.e., it is fine to defer the consumption of currency for a while but at some point it is desirable to allow others to gain the benefit of resources that a user is not using by redistributing some of the currency via the savings tax. Users should be rewarded for not wasting resources, but such a reward should not last forever. In the absence of any activity in the system, the savings tax works such that all accounts eventually converge back to their baseline values. The savings tax is collected every 4 hours, at a rate of 5% of an account's savings. These parameters were chosen such that an exhausted bank account can recover half of its balance within a few days, and the full amount in a week.[3]

By controlling the distribution of wealth in a virtual economy, the systems are able to indirectly control the share of resources received by individual participants over different time-scales, despite artifacts introduced by the virtual currency and the continued presence of load imbalance. In order to control the distribution of wealth among users, there are two policies that we must determine: how will users be able to earn virtual currency, and how (if at all) will the system limit the wealth of users. We discuss the policies chosen for our specific implementations, Bellagio and Mirage, next.

# 4 Bellagio

In this section we describe the Bellagio architecture and our deployment experience on PlanetLab.

[3]Kash, Friedman and Halpern [23, 28] have recently initiated a research agenda on developing a theory for how to allocate virtual currency within "scrip" systems in which agents both contribute and consume resources.

### 4.1 Target platform

PlanetLab is a distributed, wide-area, federated testbed consisting of over 800 machines hosted by more than 400 sites across the world (Figure 3) [6]. Machines are owned and operated locally by each site, but primarily administered centrally by PlanetLab Central (PLC). PLC performs access control and maintains a consistent software image on each machine. Each site is in charge of the physical upkeep of its machines, such as uptime and providing adequate and persistent network bandwidth. Additionally, each site pays a periodic fee to PLC to support the central administration. In return, a site's members, such as research scientists and graduate students, are granted access to use any machine in the system. Each user can obtain a *slice*, which grants her access to the resources of any PlanetLab machine. The instantiation of a slice on a particular machine is called a *sliver*. A sliver shares basic information about its associated slice, such as authentication keys and basic configuration files, and is the physical manifestation of the user's isolated environment on a machine. A sliver is therefore intended to export an interface similar to that of a virtual machine or other virtualization technology. As of this writing, PlanetLab implements virtualization using the Linux V-Server technology.

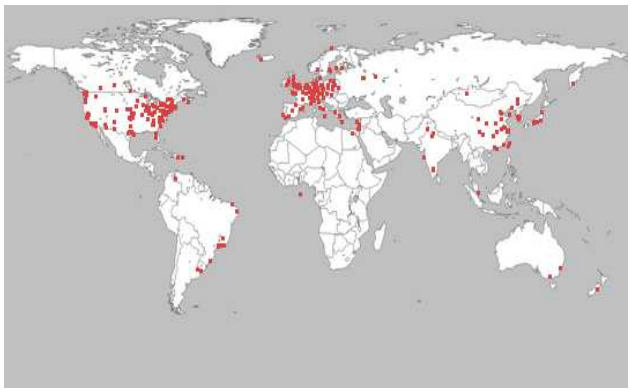

Figure 3: **Layout of PlanetLab resources as of 2008.**

A typical scenario for a PlanetLab user is that she will first obtain a slice from her local administrator (by way of PLC). Once she has a slice, she creates slivers on any number of machines. The most common use for the machines is as part of a wide-area network experiment, with each component machine acting as an end-point or providing some other function within the experiment. The value in these machines is not only in their local resources, but their geographic location; a wide-area configuration of machines provides a realistic setting for many distributed systems and network experiments. At any point in time, a particular machine can be multiplexing its resources across processes from many different slivers. These tasks can vary in length (i.e., a few minutes to months) and in size (i.e., a few nodes versus all nodes). Each machine's resources are time-shared among active slivers using an equal-weight, proportional-share scheduling policy. For example, if there are $n$ active slivers on a machine, each active sliver is expected to receive a $\frac{1}{n}th$ time-slice of a machine's CPU cycles and network bandwidth over every scheduling window. We call this type of scheduling *work-conserving*, since resource access is multiplexed among active slivers. PLC performs limited admission control such that the number of active slivers on a machine is limited to $n = 1000$ (i.e., no more than 1000 simultaneous slices can run on a machine). This constraint is imposed due to the physical limitations on memory and disk space on each machine.

The need for a more sophisticated resource allocation policy became apparent during deadlines for the major systems and networking conferences in 2003 and 2004, when resource contention became a persistent problem in PlanetLab. Figure 4 illustrates a common occurrence for PlanetLab users. Due to synchronized conference paper deadlines among users, the system experiences large spikes in resource demand in the days leading up to a deadline. The proportional-share resource allocation policy provides users with little control over their share of resources during such times, little incentive to reduce consumption, and even

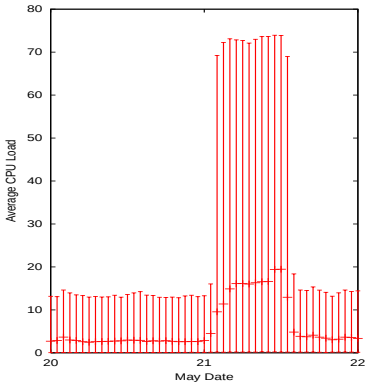


Figure 4: **5th, average and 95th percentile load on 220 PlanetLab nodes leading up to May 2004 OSDI deadline.**

less means to coordinate usage with one another. The end result is that significant resource contention leaves the system unusable for the majority of users[4]. Bellagio was designed to help address this problem.

## 4.2 Architecture

Bellagio makes allocation decisions using a combinatorial auction. We need to provide additional mechanisms on PlanetLab to allow users to bid for these resources. In this section, we describe these mechanisms.

### 4.2.1 Resource Discovery

The decision to use a combinatorial auction rather than a series of single-item auctions is motivated by the fact that most users on PlanetLab require simultaneous use of a large number of machines. Selecting these machines poses a challenge to users because there is a large set from which to choose, and machines – while largely homogeneous in resource capacity – often exhibit dynamic characteristics. For example, a typical usage scenario would involve selecting machines that exhibit low load and an abundance of network capacity. In addition, a user might want machines that have a particular physical network topology of interest, such as within the same continent, or explicitly spanning continents.

Distributed resource discovery allows end users to identify available resources based upon both of these characteristics. Bellagio uses SWORD [36] for this purpose, which is a resource discovery service implemented on PlanetLab. It exports an interface that allows PlanetLab users to locate particular resources matching various criteria. For example, users can search for resources based on resource-specific attributes (e.g., machines with low CPU load and large amounts of free memory), inter-resource attributes (e.g., machines with low inter-node latency), and logical (e.g., machines within a specific administrative domain) or physical attributes (e.g., geographic location). SWORD returns a set of candidate machines matching the user's description.

Once users submit a bid, there may be a substantial delay between bid submission and the actual allocation. Because resources can exhibit dynamic temporal characteristics, it might not be useful for a user to bid on specific machines that she has discovered from SWORD. Instead, the intended use of the resource discovery mechanism is to express abstract resource specifications. Therefore, Bellagio alternatively allows a user to specify a SWORD resource specification (i.e., a SWORD query) in her bid instead of providing details on specific machines, and Bellagio will resolve these queries immediately prior to determining a resource allocation for users.

[4]Since the Bellagio deployment and discontinuation, PlanetLab has continued to rely on Sirius, a resource reservation service. Current evidence suggests that resource contention may no longer be a severe problem [39].

### 4.2.2 Bidding Language

The previous section describes how resources bundles are identified by users. This section describes how users then communicate demands for resource bundles to Bellagio. In designing a bidding language, with which users will report bids for bundles to the market, one needs to make a number of tradeoffs. First, the expressiveness of the language and size of bids that can be submitted in a given language can directly affect the computational difficulty of solving the induced allocation problems [12, 35, 43]. But on the other hand, having a language that is both expressive and concise for users can make their life simpler. One consideration of relevance within Bellagio is the frequency with which auctions must be cleared. Very frequent auctions would require a restrictive bidding language with simple, included allocation problems while less frequent auctions would allow for a more complex bidding language and hard clearing problems. Of course, less frequent auctions would also make the system less reactive and less useful for very impatient users.

The particular choice of parameters were hand-tuned based on studies of PlanetLab usage [16] and our own experience. As a result, the basic unit of allocation in Bellagio is a 4-hour time slot of CPU shares, and the combinatorial auction correspondingly clears every 4 hours. In each period, there is a rolling window of $T = \{1, 2, 3..., 42\}$ time slots forward from the current time in the auction (where $T = 1$ denotes the immediately next time slot that will be available). We let $N$ denote the set of resources available to allocate (i.e., different PlanetLab nodes). Note that immediately prior to running the auction clearing algorithm, any bid that contains an abstract resource specification (SWORD query), Bellagio translates the resource specification to a set of concrete candidate machines. The bidding language allows a user to specify a required *allocation duration*, from the set $D = \{1, 2, 4, 8\}$ which represents the number of time-slots to reserve. A component of a bid $b_i$ from a user is constructed as follows:

$$b_i = (s_{i,0}, t_{i,1}, d_i, \{n_{i,1}, n_{i,2}, ...\}, \{ok_{i,1}, ok_{i,2}, ...\}, v_i),$$

where $s_{i,0}, t_{i,1} \in T$ denote the range of possible start times for the bid to be valid, $d_i \in D$ is the duration of the job, $\{n_{i,1}, n_{i,2}, \ldots, n_{i,k}\}$ and $\{ok_{i,1}, ok_{i,2}, \ldots, ok_{i,k}\}$ denote the required quantities $n_{i,j} \geq 1$ on resource equivalence classes $ok_{i,j} \subseteq N$. An example of equivalence class is $ok_{i,j} = \{*.\texttt{princeton.edu}\}$, where $ok_{i,j}$ represents the subset of PlanetLab nodes located in the `princeton.edu` domain. The corresponding $n_{i,j}$ parameter represents the desired quantity of nodes satisfying this equivalence class. $v_i \geq 0$ is the bid price (willingness to pay) for any bundle of resources that satisfy this bid. These parameters allowed PlanetLab users to reserve CPU shares for a duration of up to 32 hours, and up to a week in advance. For example, a user might request "any 10 nodes from Princeton, and any 10 nodes from Berkeley, for 32 consecutive hours anytime in the next 24 hours." A corresponding bid of value $v$ would be:

$$b_i = (1, 6, 8, \{10, 10\}, \{\{*.\texttt{princeton.edu}\}, \{*.\texttt{berkeley.edu}\}\}, \texttt{v}).$$

### 4.2.3 Currency Policy

The currency distribution policy in Bellagio is performed at a per-organization basis. Each site begins with an initial balance of virtual currency that is proportional to the number of machines contributed to PlanetLab. This policy was designed to encourage contributions to PlanetLab, and in particular to reward those who contribute the most. If new sites joined PlanetLab, they were assigned currency in this way. This is the only way that currency can be introduced in the economy. In PlanetLab, individual users and slices are associated with a particular site or institution. All users and slices associated with that institution have rights to the site's bank account. When a user makes a bid, her balance is temporarily frozen into an intermediate Bellagio account. Each time an auction clears, revenue is collected from the accounts of winning bidders.

As described earlier, each bank account is also associated with a currency share which determines the site's portion of profit sharing. Initially, the currency share for a bank account was the same as its initial balance – the number of machines their site has contributed to PlanetLab. However, we modified this policy

upon observation that certain machines in the system were used much more often than other machines. We wanted a policy that would reward organizations which contribute the most valuable (highly utilized) resources. Therefore, we revised profit sharing such that it is only divided among each of the organizations that owns a machine in a winning allocation; profit (revenue) from each auction-clearing period is funneled directly back to the account of the PlanetLab organization that owns the machine. One implication of this policy is the promotion of long-term system growth. Users wishing to receive a larger revenue share will now have incentive to contribute useful resources to the shared infrastructure.

Once the payment amount for each winning bid has been determined, the user's bank account is charged by the appropriate amount, and resource bindings are performed by returning resource capabilities in the form of tickets to the winning bidders using a system such as SHARP [24]. Access to all resources in Bellagio is guarded by resource capabilities. A resource capability represents the right for an application to use a resource, and an application must present the resource capability to the resource in order to use it.

### 4.3 Deployment

In the Summer of 2004, PLC provided the opportunity for research groups on PlanetLab to experiment with more sophisticated scheduling policies. Beginning in December 2004, timed with the release of PlanetLab Kernel version 3.0, Bellagio was given a fraction of each PlanetLab machine's CPU resources, which it could then allocate to different user slices. For example, if a machine has $n$ active slices, each slice receives $\frac{1}{n}$ time-share of the machine's CPU. A full allocation of Bellagio's share of resources to a single slice would provide the slice with $\frac{3}{n+2}$. For large $n$, the relative boost a slice could receive is 200% Therefore, for a single machine, the full allocation of Bellagio's resource can double or triples a slice's relative CPU scheduling share. We stress that the use of Bellagio was strictly optional: PlanetLab users automatically received a default proportional share allocation; our system only sold an increase in these soft resource shares. This particular resource boost is an artifact of the scheduling technology used by PlanetLab, and the amount of resources that we were provided to allocate within our market.

Because we were introducing a market-based allocation scheme to a live system which previously had no notion of markets, we anticipated the learning curve for new users would be high. The Bellagio user interface was designed to encourage early adoption by the PlanetLab user community and promote frequent use. In order to lower the barrier to for the user community, every registered PlanetLab user was automatically registered to use Bellagio; a user simply supplied her authentication credentials to the Bellagio Web interface, which were then verified against the central PlanetLab database. The Web interface was deployed on a cluster of Linux machines with a PHP front-end and a PostgreSQL database back-end. The database contained account balances for virtual currency and managed user authentication.

Once authenticated by the Web interface, a user could view the status of her account and previous bids and allocations. To facilitate the bidding process, we provided several useful guides. First, to address the issue of "valuation uncertainty," which is an issue related to whether or not users will understand how to value resources in units of the virtual currency, we provide historical prices in Bellagio as a reference point to the current level of demand in the system. Second, to help formulate bids, we provided several "one-click" bidding options, such as "bid on any $N$ nodes in my slice" or "any $N$ nodes from $K$ distinct Autonomous Systems." Users also had the option of formulating complex queries by defining equivalence classes of resources on which to bid. And, as mentioned, Bellagio provided an interface to SWORD to perform these queries, which is a tool familiar to many PlanetLab users [36].

Bellagio was released for public use on February, 2005. Each site was given a balance of 100 units of virtual currency per node; the initial balance of sites ranged from 100 to 2200. During its lifetime, it saw 23 bids from PlanetLab users representing 13 different organizations, which has resulted in a total of 242,372 allocated hours of CPU shares. Bellagio was taken off-line in the Summer of 2005.

# 5 Mirage

This section we describe the Mirage architecture and our deployment experience on the Intel Research Sensor Network Lab.

## 5.1 Target platform

The initial motivation for this work became apparent during the construction of a 148-node testbed at the Intel Research laboratory (IRB) in Berkeley, CA (Figure 5). This testbed is comprised of 97 Crossbow MICA2 and 51 Crossbow MICA2DOT series sensor nodes, or motes, mounted uniformly in the ceiling of the lab. The motes incorporate an Atmel ATmega128 8-bit microcontroller, 4KB of RAM, 128KB of flash memory, and a Chipcon CC1000 FSK radio chip. The MICA2 series devices in the testbed operate in the 433 MHz ISM band and incorporate a sophisticated sensorboard that can monitor pressure, temperature, light, and humidity. The MICA2DOT devices operate in the 916MHz ISM band and do not include sensorboards.

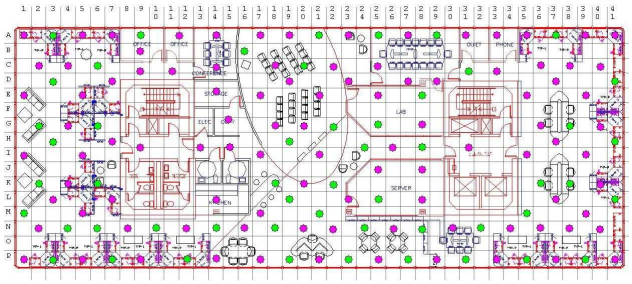

Figure 5: **The Intel Berkeley lab testbed layout.**

Users of a sensor-network testbed such as IRB's are frequently interested in acquiring combinations of resources; e.g., resources that meet certain constraints. For example, consider a machine learning researcher who is interested in testing distributed inference algorithms in sensor networks. Such a user might be interested in evaluating algorithms at a moderate scale while performing inference over temperature and humidity readings of the environment. The user's code might also be tailored to a particular type of device (e.g., a MICA2 mote) and needs to run on a different, appropriately-spaced frequency to avoid crosstalk from other experiments. Thus, a user's abstract resource requirement might be something like "any 64 MICA2 motes, operating on an unused frequency, that have both a temperature and a humidity sensor".

## 5.2 Architecture

As described in the Bellagio architecture, Mirage makes allocation decisions using a combinatorial auction. We describe in this section the methods that we provide in Mirage to allow users to bid for motes.

### 5.2.1 Resource Discovery

Similar to Bellagio, Mirage users specify the type of resources they are interested in using as abstract resource specifications. A resource discovery service then maps these specifications to concrete resources that meet the desired constraints. We use this level of indirection for three reasons. First, it frees the user from having to manually identify candidate resources. Second, it allows users to automatically take advantage of new resources as they are introduced into the system. Finally, as mentioned, testbed users are frequently interested in acquiring sets of nodes and are often indifferent to which specific nodes they are allocated as long as the candidate resources meet the user's constraints. The resource discovery service

allows users to discover all possible candidates and thus provide the system with the maximal amount of information on substitutes when clearing the auction.

Abstract resource specifications allow users to specify constraints on the types of resources they seek to acquire. For example, testbed users often need to specify constraints on per-node attributes. In the machine learning example earlier, for instance, a logical conjunction on per-node attributes (mote type and sensor board type) combined with a desired number of nodes is required. In other cases, constraints on inter-node attributes may be necessary. For example, a user might wish to acquire "8 motes where each pair of motes is at least 10 meters apart" to ensure that the network causes a multi-hop routing layer to form. Currently, Mirage supports resource discovery using per-node attributes including mote type, sensor board type, and supported frequency range.

### 5.2.2 Bidding Language

The bidding language allows users to express desired resources and the value for those resources. More specifically, a bid in the bidding language includes combinations of concrete resources, obtained via the resource discovery service, that are equally acceptable to the user and the maximum amount the user is willing to pay for those resources. Since time is a critical aspect of resource allocation (e.g., resources near a conference deadline), resource combinations specify resources in both space and time. Formally, a bid $b_i$ in Mirage is specified as follows:

$$b_i = (v_i, s_i, t_i, d_i, f_{min}, f_{max}, n_i, ok_i) \tag{1}$$

Bid $b_i$ indicates the user wants *any combination* of $n_i$ motes from the set $ok_i$ (obtained through resource discovery) for a duration of $d_i$ hours with a start time anywhere between times $s_i$ and $t_i$ and a frequency anywhere in the range $[f_{min}, f_{max}]$. The associated bid price is $v_i$, representing the units of virtual currency that the user is willing to pay for these resources. Continuing with the distributed inference example, a user thus might say: "any 64 MICA2 motes, which have both a temperature and a humidity sensor, operating on an unused frequency in the range [423 MHz, 443 MHz], for 4 consecutive hours anytime in the next 24 hours". Suppose the user used the resource discovery service and found 128 motes meeting the desired resource specification and valued the allocation at 99 units of virtual currency. The corresponding bid in this case would thus be:

$$b_i = (99, 0, 20, 4, 423, 443, 64, \text{list of 128 motes}) \tag{2}$$

Mirage uses a greedy heuristic algorithm to compute the set of winning bids, similar to the algorithm used by Bellagio.

The resources that a user of the IRB testbed cares about can exhibit both substitutes and complements. For example, in the machine learning example, the user does not care which specific MICA2 motes are allocated as long as a total of 64 of them are allocated. Hence, MICA2 motes are substitutes for one another. Similarly, the user does care that 64 motes are allocated simultaneously. A partial allocation of, say, 8 motes would not meet the user's needs in this case since the user's intention was to test at a moderate scale. (The extreme case would be a partial allocation of a single mote.) Thus, the 64 motes can be viewed as being complimentary to one another.

### 5.2.3 Currency Policy

As in Bellagio, each user is associated with a project that has an account at a central bank which stores virtual currency. Each project's bank account is assigned a baseline amount of currency based on priority, a number of currency shares, which influences the rate that currency flows into the account, and is initialized with a baseline amount of currency. Priority and currency shares are determined exogenously, and mostly based on the type of usage (e.g., research vs coursework). Most accounts have a baseline balance and currency share of 1000, while 2 local users have a balance and share of 2000. Note that the profit sharing

policy differs from the one used in Bellagio. In Mirage, the sensor nodes are all owned by the IRB lab, and therefore, profit cannot be associated with a particular project, whereas in PlanetLab a resource can be directly attributed to a site.

### 5.3 Deployment

We deployed Mirage on Intel Research Berkeley's 148-mote sensor-network testbed in December 2004, and it is still in operation as of this writing. The implementation is comprised of three types of components: clients, a server, and a front-end machine that users log in to and which provides controlled physical access to the testbed (Figure 6). Clients provide users with secure, authenticated command-line (the `mirage` program) and web-based access to a server (`miraged`) which implements a combinatorial auction, bank, and resource discovery service. The server provides service to clients by handling secure, authenticated XML-RPC requests using the SSL protocol with persistent state stored in a PostgreSQL database. Lastly, the front-end physically enforces resource allocations from the auction using Linux's per-uid `iptables` packet filtering capabilities. By default, all users are denied access to all motes. Based on the outcome of the auction, rules are added as needed to open up access to users of winning bids for specific periods of time.

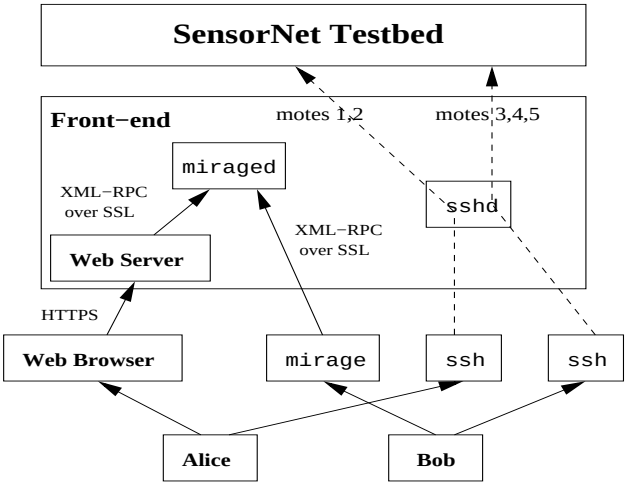


Figure 6: **Mirage implementation.**

The auction is parameterized by several variables: number of resource slots, resource slot size, and acceptable bid durations. Our deployment includes 148 motes where access to those motes is based on 1-hour slots, the minimum time unit of resource allocation. To accommodate users who might require a range of different times with the motes, users may bid for either 1, 2, 4, 8, 16, or 32 hour duration blocks. To allow users who wish to plan ahead (e.g., perhaps near a conference deadline), the auction will sell resources for up to three days into the future. Given our slot size of 1-hour, this works out to a total of 72 slots. Thus, we can view the resources being allocated as a matrix of 148 motes by 72 time slots. When the system boots, all slots are available. Over time, slots become occupied as bids are allocated resources and new slots become available as the window of slots opens up over time.

To use the system, users register for an account at a secure web site by providing identifying information, contact information, a project name, and by uploading an SSH public key. Each user is associated with a project and each project has an owner. An administrative user is responsible for enabling accounts for project owners and assigning each project a baseline virtual currency value and a number of virtual currency shares. Project owners can subsequently enable their own users, thereby eliminating the administrative user as a centralized bottleneck.

Users bid securely in the auction using either the command-line tool `mirage`, which acts as an XML-RPC/SSL client, or through the web-based interface, where PHP scripts on the back-end act as XML-RPC/SSL clients to the relevant servers. The command-line tool provides full access to the entire RPC interface exposed by `miraged`. Use of this program is useful for various types of scripting and automation. To accommodate users who prefer a graphical interface, the web-based interface provides a simple, integrated interface to the system where users specify what resources they want and how much they are willing to pay using an HTML form. The web server, in turn, maps the user's abstract resources to concrete resources using the resource discovery service and places a bid in the auction on the user's behalf.

To use testbed resources, each winning bid results in members of the associated project being given access to a specific set of motes for a period of time specified in the bid. Motes are made physically accessible to project users through the front-end by doing the following for each project member: (i) creating a temporary Unix login on the front-end machine using a global username (MD5 hash of the user's SSH public key), (ii) enabling access to the front-end via SSH authentication using an SSH `authorized_keys` file, and (iii) setting up firewall rules on the front-end such that only the user can access the particular motes assigned to the winning bid.

In the first six months of usage, 18 different research projects were registered to use the system and 322 bids were submitted resulting in a total of 312,148 allocated node hours.

# 6 Experiences

The goal of both Bellagio and Mirage is to provide an allocation policy that increases the aggregate utility of its users. But it can be challenging to measure our success in reaching this goal. Although we have real workload data from which we can quantify the aggregate utility (in units of virtual currency) of our scheduling policy on our target users, it is difficult to quantify the aggregate utility of a different scheduling policy using the same workload data. The problem is that there may have been bids that were withdrawn, repeated or simply not submitted in the Bellagio and Mirage workload data. If we cannot account for the absent (or duplicated) resource requests, we cannot accurately reconstruct a workload stream that would have been presented to a more traditional scheduling policy, such as FCFS or proportional-share schedulers. Nevertheless, our data contains sufficient information that demonstrates the our market-based systems likely *do* increase utility in both domains. We explain this next.

Figure 7 indicates that resource contention continues to persist on the IRB sensor network testbed, as it had prior to the deployment of Mirage. We can see that during times of heavy system load, the bidding process is able to allocate resources at a higher price (Figure 8), and resolves contention by allocating the scarce resource supply to users who valued them the most. Figure 9 demonstrates that individual users place bids that range over four orders of magnitude. From this data, we can conclude that users indeed place different levels of priority on resources (assuming that the bids don't represent grossly misstated valuations) at different times, which suggests that extracting this information can improve the allocation efficiency of a system, i.e. the total utility achieved by its user base.

Similar observations about resource valuation were made from our deployment on Bellagio, however the data is significantly less reliable because of the few data points available from deployment. As emphasized in the description of the deployment, use of Bellagio was strictly optional to PlanetLab users. As a result, the market did not capture the full demand of the system. In particular, when overall system demand was low, there was virtually *no* activity on Bellagio. This lack of activity was primarily due to the availability of free resources (i.e., those machine shares not allocated by the market), and in fact ultimately lead to Bellagio being taken offline. We see this result as an artifact of the opt-in nature of Bellagio, and not a negative result about the applicability of a market-based scheduler on the PlanetLab domain.

This example also serves to illustrate that users will often choose an easier-to-user approach, even if it results in slightly less utility. One particular burden of our auction-based approach is the artificial delay to node allocations. Since auctions are run every hour, allocations to winning bids are correspondingly delayed until the hour. There is a trade-off here between imposing little delay on users and maximizing the

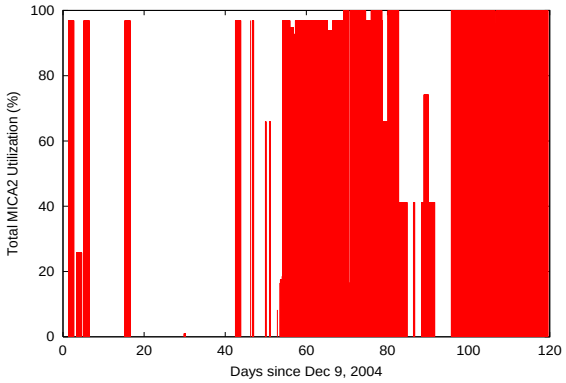


Figure 7: **Testbed utilization for 97 MICA2 motes.**

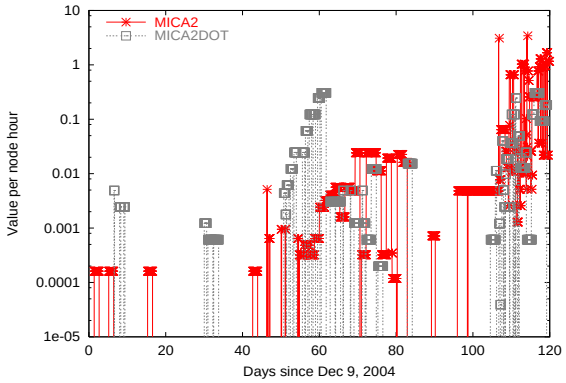


Figure 8: **Median node-hour market prices.**

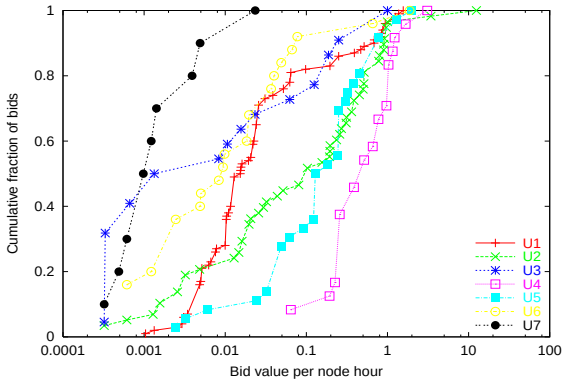


Figure 9: **Bid value distributions by user.**

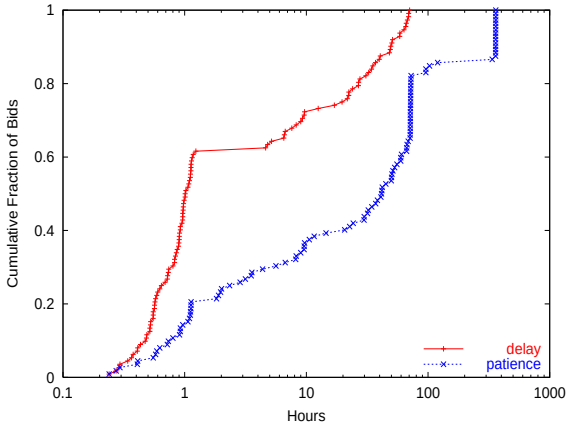


Figure 10: **Distribution of delay and patience from bids submitted to Mirage from January 20 to March 22, 2005.**

efficiency of an allocation: increasing the frequency of auctions reduces the delay imposed on users, but it also reduces the amount of demand information captured in an auction, potentially reducing the quality of the allocation.

In Mirage, the auction-clearing period of 1 hour was designed to mitigate the impact of the imposed auction delay. However, based on the available data, we observe that a significant fraction of the bids from users could not be satisfied given the 1-hour delay. Each bid in Mirage includes both a delay and patience field, indicating an earliest start time and latest start time allowed for an allocation. These fields are used to communicate time-based constraints such as a deadline for an allocation. Figure 10 indicates that 10% of user bids could not wait for the 1-hour duration. This effect was more obvious in Bellagio, where a majority of PlanetLab users chose not to bid at all.

Bellagio and Mirage were both designed to reduce the usage overhead imposed on the users. In Mirage, we provided some information transparency by using a first-price, open auction in order to help guide user bidding behavior. Despite the potential for strategic manipulation in an open auction, we decided to prioritize our goals for usability and efficiency improvement over incentive-compatibility. Perhaps not unexpectedly, users not only learned how to use the system effectively, but a few users eventually developed strategies to manipulate allocations in their favor. These results provide evidence that some end users may exhibit the sophisticated usage patterns of rational economic agents and serves to justify the use of market based methods in seeking to address these kinds of manipulations. The following are descriptions of the four primary bidding behaviors that we observed during the initial Mirage deployment.

**Behavior 1**: *underbidding based on current demand*. Since all outstanding bids in our initial deployment were publicly visible, users could observe periods with low demand and submit low bids in these periods. For example, one user would frequently bid a low value, such as 1 or 2 when no other bids were present. Underbidding is not necessarily a problem in this situation, because it can still result in a utility-maximizing allocation; e.g., if supply exceeds demand then all interested users receive an allocation. This can be a problem in for-profit systems, however, where it would be important to use a reserve price to provide good revenue properties. Moreover, it suggests that users need to be strategic in thinking about how and when to bid, which indicates that such a system might be difficult to use.

**Behavior 2**: *iterative bidding*. The possibility of user underbidding coupled with uncertainty about the bid values of other users results in "iterative" bidding, i.e. a behavior in which a user adjusts her bids while an auction remains open and in response to price feedback. This poses a problem for system performance in Mirage because the auctions need to have a definite closing time (because the associated resources are perishable), and users may still be adjusting their bids when an auction closes. The impact of this strategy is a potential efficiency loss in the allocation.

**Behavior 3**: *rolling window manipulation*. Unlike auctions for tangible goods, resource allocation in computer systems are not allocated permanently, but rather allocated for particular intervals of time. Since

many experiments by Mirage users can span several days, we permitted users to bid for allocation blocks of 1, 2, ..., or 32 hours in size. In order to allow users to plan in advance, we auctioned off resources over a rolling window of 72 hours into the future. In periods of overdemand (e.g., during the SenSys 2005 conference deadline) the entire window of resources becomes fully allocated, and we found that the design of the rolling window can lead to unintended consequences. For example, consider a scenario with only two users, A and B, where user A requires a 4-hour block of nodes, and user B requires only a 2-hour block. If the window of resources is fully allocated, user A must wait at least 4 hours before a contiguous 4-hour block is available. However, after only 2 hours, user B will win her allocation, regardless of her valuation relative to user A's. Furthermore, if user B continues this behavior, it is possible that user A will be starved indefinitely. During times of heavy resource load, we observed that no bids involving a block larger than 2-hours could be satisfied. It is possible that there were outstanding bids involving larger blocks and proportionally larger bid amounts that could not be allocated because of the rolling window.

Given the negative impact of these observed user behaviors, we responded by adjusting the auction protocol in Mirage. First, we instituted a sealed-bid auction format, thereby discouraging Behaviors 1 and 2. We also responded to Behavior 3 by increasing the allowable time window to be 104 hours, with bid start times constrained to be within the next 72 hours. To understand the rationale for this change, consider the following example. Assume that we have two users, A and B, where user A requires a 32-hour block of nodes (the maximum allowed in Mirage), and user B requires a 16-hour block of nodes. If the entire 72-hour window is allocated, the next available 32-hour block occurs 104 hours in the future. By expanding the rolling window to 104 hours and restricting the last 32 hours (of the window) from being reserved, bids from both user A and B will be considered for the next available 32-hour block. Under the original auction format, the bid from user B would win the allocation before the bid from user A could even be considered.

| **Time** | **Project** | **Value** | **#Nodes** | **#Hours** |
|---|---|---|---|---|
| 04-02-2005 03:58:04 | user B | 1590 | 97 | 32 |
| 04-02-2005 05:05:45 | user A | 5 | 24 | 4 |
| 04-02-2005 05:28:23 | user A | 130 | 40 | 4 |
| 04-02-2005 06:12:12 | user A | 1 | 33 | 4 |

Table 1: **Behavior 4 on 97 MICA2 motes.**

**Behavior 4**: *auction sandwich attack*. While our changes eliminated Behavior 3 and significantly reduced Behaviors 1 and 2, a fourth behavior exploited the available information about awarded allocations. In the so-called "auction sandwich" attack, a user exploits two pieces of information: (i) historical information on previous winning bids to estimate the current workload and (ii) the greedy nature of the auction clearing algorithm. In this particular case, we observed a user employing a strategy of splitting a bid for 97 MICA2 motes across several bids, only one of which has a high value per node hour. During times of high resource demand, most users request a majority of nodes (most often all 97 motes, since a conference deadline requires large-scale experiments). Since Mirage uses a greedy (first-fit) heuristic in its auction-clearing algorithm, the user's single high value bid is likely to win and because the bids from other users are then blocked and unable to fit in the remaining available slots, her other low-valued bids fill the remaining slots. We produce an actual occurrence of this behavior in Table 1. Here, user A submits three bids, the main one being a bid with value 130 (value per node hour $130/(4 \cdot 40) = 0.813$) and used to outbid a bid from user B, with value 1590 (value per node hour $1590/(32 \cdot 97) = 0.0512$). Once the high valued 40-node bid has occupied its portion of the resource window, no other 97-node bids can be matched. Consequently, the user wins the remaining 57 nodes using two bids: a 24-node bid and a 33-node bid, both at low bid prices.

# 7 Concluding Remarks

From our deployments, we have seen that a market-based allocation system can significantly reduce the management burden and simultaneously improve user satisfaction on large-scale federated infrastructures.

The allocation decisions in Bellagio and Mirage are autonomously driven by user-provided job utility information, with very limited human intervention; when user behavior is properly constrained, the scheduling policies can lead to utility-maximizing resource allocations. While market-based systems have not yet seen widespread adoption, we believe that further efforts to deploy and study live systems will promote more support in the research and systems community, and can ultimately achieve the goal of more efficient utilization of today's computing platforms.

Overall, we made the following observations in this chapter:

**Markets can increase allocation efficiency, but only if users behave properly.** It can be difficult to rigorously compare the efficiency of two different allocation policies because the workload data that would be generated by each policy are different. However, from the Mirage deployment, we have demonstrated several scenarios where markets can be expected to outperform traditional allocation policies. One important aspect of this is the observation that even in large-scale infrastructures, resource demand often exceeds resource supply: in both Mirage and Bellagio, there are unavoidable periods of time where users' resource demands overlap, creating a situation where the system must arbitrate among them. We have observed user bids varying by several orders of magnitude, indicating that there is room to make intelligent (and therefore, unintelligent) allocation decisions. Well-designed markets will promote a socially optimal allocation for the actual supply and demand conditions. In the case of Mirage, we saw that strategic behavior from a subset of users could potentially decrease the utility of the allocation to other users. By adjusting the allocation policy, we were able to limit the opportunities for strategic manipulation, and mitigate this negative impact on well-behaved Mirage users.

**Users are able to utilize market-based methods and will follow sophisticated behaviors if given sufficient incentive.** There exist many simpler alternatives to markets for the design of methods for resource allocation; e.g., traditional methods that do not require the provision of bid information to describe demands by users. We have hypothesized that users might sometimes prefer this simplicity over the potential benefits of a market. But we have also observed that given enough incentive, users are willing to put forth the effort to use a market system. In Bellagio, we saw that there was neither enough pressure or sufficient incentive to use the market, and the vast majority of users opted for the simpler proportional allocation scheme in PlanetLab and were willing to forego any additional resources that Bellagio could provide. On the other hand, users in Mirage were given no alternative for obtaining a sensor node allocations. While finding that users were able to utilize the market-based system, we also observed both simple and strategic behavior, with the latter providing a (sometimes successful) attempt to manipulate allocations in their favor.

Our deployment experience has also highlighted a few key obstacles that can be addressed by future research:

**Mechanisms with faster response time.** In the course of the Bellagio deployment, we learned that many users prefer immediate use (i.e., standard best-effort, proportional-share) over a potentially larger resource allocation (acquired from bidding). In Mirage, we also observed that many users required more immediacy for a significant fraction of their resource requests. As alluded to earlier, there is a trade-off between offering a resource immediately and the efficiency gained from delaying an allocation decision. A hybrid mechanism, such as buy-it-now pricing on eBay [1], may provide a compromise between these desired properties.

**Low-cost mechanism or highly-valued resources.** In the Bellagio deployment, we learned that the resources made available through the auction were not valuable enough to users to justify expending the effort to obtain them. Many PlanetLab experiments are bottle-necked by other resources such as memory or disk bandwidth and obtaining a few extra soft CPU shares is of little value. A market mechanism must either provide a resource of obvious and significant value to the end-user, or impose a low enough cost to

a user to justify its adoption. In Mirage, the market is the only means to obtain a sensor node, but having a market with a low cost of use was nevertheless important in promoting its adoption in the first place.

**Strategyproof methods.** As we saw in Behavior 4 from the Mirage deployment, some users will try and can also succeed in exploiting the particular characteristics of a market design. In promoting simplicity for users and also robustness and predictability for system designers, it can be helpful to deploy non-manipulable are "strategyproof" methods. There is a developing literature on the design of such mechanisms for dynamic environments (see Parkes [37] for a recent survey), and it will be of interest to continue to seek to adopt these methods within computer systems [27].

## References


[1] eBay, inc. `http://www.ebay.com/.`

[2] Emu-lab. `http://www.emulab.net/.`

[3] HP utility data center. `http://www.hpl.hp.com/news/2003/apr_jun/udc.html?jumpid=reg_R1002_USEN`.

[4] Millennium sensor-network cluster. `http://www.millennium.berkeley.edu/sensornets/.`

[5] Parallel Workloads Archive. `http://www.cs.huji.ac.il/labs/parallel/workload/.`

[6] Planet-lab. `http://planet-lab.org/.`

[7] San Diego Supercomputing Center. `http://www.sdsc.edu/.`

[8] TeraGrid. `http://teragrid.org/.`

[9] A. AuYoung, B. N. Chun, A. C. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *ACM Workshop on Operating System and Architectural Support for the On Demand IT Infrastructure (OASIS)*, October 2004.

[10] A. AuYoung, L. E. Grit, J. Wiener, and J. Wilkes. Service contracts and aggregate utility functions. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, 2006.

[11] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4:125–144, 1992.

[12] C. Boutilier. Bidding languages for combinatorial auctions. In *In Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 1211–1217, 2001.

[13] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centres. In *18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[14] K. Chen and P. Muhlethaler. A scheduling algorithm for tasks described by time value function. *Real-time Systems*, 10(3):293–312, May 1996.

[15] N. Christin, J. Grossklags, and J. Chuang. Near rationality and competitive equilibria in networked systems. Technical Report 2004-04-CGC, apr 2004.

[16] B. Chun and A. Vahdat. Workload and failure characterization on a large-scale federated testbed. Technical Report IRB-TR-03-040, Intel Research Berkeley, November 2003.

[17] B. N. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *Proceedings of 2nd IEEE Workshop on Embedded Networked Sensors (EmNetsII)*, 2005.

[18] B. N. Chun and D. E. Culler. User-centric performance analysis of market-based cluster batch schedulers. In *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid*, May 2002.

[19] R. K. Clark, E. D. Jensen, and F. D. Reynolds. An architectural overview of the alpha real-time distributed kernel. In *Winter USENIX Technical Conference*, April 1993.

[20] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[21] D. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel Job Scheduling – A Status Report. In *10th Workshop on Job Scheduling Strategies for Parallel Processing*, June 2004.

[22] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers Inc., 1999.

[23] E. J. Friedman, J. Y. Halpern, and I. Kash. Efficiency and nash equilibria in a scrip system for p2p networks. In *EC '06: Proceedings of the 7th ACM conference on Electronic commerce*, pages 140–149, New York, NY, USA, 2006. ACM.

[24] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. Sharp: an architecture for secure resource peering. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 133–148. ACM Press, 2003.

[25] Intel Research Berkeley. Mirage: Microeconomic resource allocation for sensornet testbeds. `http://mirage.berkeley.intel-research.net/`.

[26] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 87–102. Springer Verlag, 2001. Lect. Notes Comput. Sci. vol. 2221.

[27] L. Kang and D. C. Parkes. A decentralized auction framework to promote efficientresource allocation in open computational grids. In *Proc. Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing (NetEcon-IBC)*, San Diego, CA, 2007.

[28] I. A. Kash, E. J. Friedman, and J. Y. Halpern. Optimizing scrip systems: efficiency, crashes, hoarders, and altruists. In *EC '07: Proceedings of the 8th ACM conference on Electronic commerce*, pages 305–315, New York, NY, USA, 2007. ACM.

[29] V. Kumar, B. F. Cooper, and K. Schwan. Distributed stream management using utility-driven self-adaptive middleware. In *Proceedings of 2nd International Conference on Autonomic Computing (ICAC'05)*, June 2005.

[30] K. Lai, B. A. Huberman, and L. Fine. Tycoon: A distributed market-based resource allocation system. Technical Report cs.DC/0404013, April 2004. Available at `http://arxiv.org/abs/cs.DC/0404013`.

[31] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen. A scalable solution to the multi-resource qos problem. In *20th IEEE Real-Time Systems Symposium (RTSS'99)*, December 1999.

[32] C. B. Lee and A. Snavely. On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions. In *International Journal of High Performance Computing Applications*, volume 20, No.4, pages 495–506, 2006.

[33] L. Levy, L. Blumrosen, and N. Nisan. On line markets for distributed object services: the majic system. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, March 2001.

[34] C. Ng, P. Buonadonna, B. N. Chun, A. C. Snoeren, and A. Vahdat. Addressing strategic behavior in a deployed microeconomic resource allocator. In *Proceedings of 3rd Workshop on the Economics of Peer to Peer Systems (p2pecon)*, 2005.

[35] N. Nisan. Bidding and allocation in combinatorial auctions. In *In Proceedings of the 2nd ACM Conference on Electronic Commerce*, pages 1–12, 2000.

[36] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable wide-area resource discovery. Technical Report UCB//SD-04-1334, UC Berkeley and UC San Diego, 2004.

[37] D. C. Parkes. Online mechanisms. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 16. Cambridge University Press, 2007.

[38] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of HotNets-I*, October 2002.

[39] L. Peterson, V. Pai, N. Spring, and A. Bavier. Using PlanetLab for Network Research: Myths, Realities, and Best Practices. Technical Report PDN–05–028, PlanetLab Consortium, June 2005.

[40] D. Petrou, G. R. Ganger, and G. A. Gibson. Cluster scheduling for explicitly speculative tasks. In *Proceedings of International Conference on Supercomputing (ICS'04)*, June–July 2004.

[41] O. Regev and N. Nisan. The popcorn market – an online market for computational resources. In

*Proceedings of the 1st International Conference on Information and Computation Economies*, October 1998.

[42] M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinational auctions. *Management Science*, 44(8), August 1998.

[43] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Winner determination in combinatorial auction generalizations. In *In Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 69–76, 2002.

[44] J. Shneidman, C. Ng, D. Parkes, A. AuYoung, A. C. Snoeren, A. Vahdat, and B. N. Chun. Why Markets Could (But Don't Currently) Solve Resource Allocation Problems in Systems. In *Proceedings of HotOS: 10th USENIX Workshop on Hot TOpics in Operating Systems*, 2005.

[45] I. Stoica, H. Abdel-Wahab, and A. Pothen. A microeconomic scheduler for parallel computers. In *Proceedings of the 1st Workshop on Job Scheduling Strategies for Parallel Processing*, April 1995.

[46] I. E. Sutherland. A futures market in computer time. *Communications of the ACM*, 11(6):449–451, 1968.

[47] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–177, February 1992.

[48] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proceedings of 1st International Conference on Autonomic Computing (ICAC'04)*, May 2004.

[49] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A Wireless Sensor Network Testbed. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks (IPSN'05), Special Track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS)*, 2005.

[50] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik. G-commerce: Market formulations controlling resource allocation on the computational grid. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, March 2001.