Virtual Worlds: Fast and Strategyproof Auctions for Dynamic Resource Allocation

Chaki Ng David C. Parkes Margo Seltzer
Division of Engineering and Applied Sciences,
Harvard University, Cambridge MA 02138
{chaki,parkes,margo}@eecs.harvard.edu

November 16, 2002

Abstract

We consider the problem of designing fast and strategyproof auction mechanisms for dynamic resource allocation problems in distributed systems. We propose a scalable design for an exchange for data staging between multiple request and service agents. Incoming jobs are distributed into auction pools and matched against sequences of auctions. We introduce a novel virtual worlds construction to extend a fast and strategyproof mechanism to our setting, and retain strategyproofness over a sequence of auctions. The goods for sale in an individual auction are selected by partitioning a service provider's available capacity into smaller consignments. A fair sharing method is used to assign an individual seller to each auction. We present experimental results to demonstrate the efficiency and strategyproofness of the mechanism in an implemented system.

1 Introduction

Computation is increasing distributed, and performed by devices representing multiple users, including individuals and businesses. Moreover, in many settings the devices are likely to represent self-interested users, and as such cannot be expected to blindly follow suggested protocols as has been assumed in traditional work on distributed systems [FS02].

Consider a data staging scenario, in which multiple users with PDAs are in Times Square and trying to read and process email messages and access corporate databases. Each user would like to stage their (encrypted) data within their physical environment to reduce latencies. A socially-efficient allocation of data staging capabilities in Times Square would allocate capacity to maximize the total value across all users. With cooperative users, and other computational considerations aside, one could simply ask devices to state their utilities for various outcomes and then implement this efficient allocation. However, rational and self-interested users would overstate their utility for the ability to stage their own data, and cause the system performance to quickly unravel.

In this paper we formulate a simplified model of this data staging scenario as a problem in mechanism design, and propose an approximately-efficient auction-based method to match requests for storage with seller capacity and perform dynamic resource allocation. In addition to adopting the goal of maximizing the total value, or *allocative efficiency*, within a system, by allocating resources to the users with the highest value, we choose to focus on the following three important properties:

strategyproof We seek a mechanism in which truth-revelation is the optimal strategy for all users, whatever the state of the system and whatever the strategy of other users. Equilibrium in strategyproof mechanisms are robust and simple to compute. Users can simply state their presence, and requirements and capabilities, upon arrival into such a system.

fast We seek a mechanism that is *fast* and *scalable*, to allow widescale deployment in decentralized systems. From the perspective of strategyproof mechanism design this is especially challenging, because traditional solutions, such as Vickrey-Clarke-Groves mechanisms, often require the optimal solution of hard optimization problems and approximations can easily break strategyproofness [NR00].

dynamic The seek a mechanism that must operate in a dynamic environment, with agents arriving and departing over time. This provides an additional challenge, because mechanism design traditionally assumes worlds in which all the agents are present for all time.

Our solution contains some novel design ideas, that arise from handling these three desiderata. We can make some interesting theoretical claims about our system, and initial experimental results are quite promising.

However, we view this work as but a first step towards the development of general infrastructure-level support for *plug-and-play negotiation* in computational environments. In this vision, rational self-interested computational devices that move around in ad hoc and pervasive systems are freed from the need to perform complex game-theoretic reasoning. Instead, we seek to design a family of rules of interaction to promote simple truth-revealing strategies, across a number of different types of negotiation scenarios.

The system we propose is strategyproof for agents that request service, at least with respect to information about the value, required storage size, and required storage time. For the moment we ignore temporal strategies, and assume that all agents truthfully announce their arrival into the system. The system is also robust with respect to overstatement of capacity by service providers. Although it is not strategyproof with respect to understatement of capacity, and we characterize conditions when this can help, the experimental results suggest that this effect is negligible.

Although the data staging scenario has characteristics of an exchange, with multiple buyers and sellers, we choose to structure the mechanism around sequences of auctions. Incoming requests are divided across multiple pools, and within each pool there is a sequence of auctions, with each auction associated with a single service provider and particular subset of that provider's resources. Unsuccessful bids are retained in subsequent auctions. Such a sequence of auctions breaks the strategyproofness of well known auctions such as Vickrey (second-price) auctions, because an agent can influence the price that it pays by submitting a bid that is successful in the auction with the lowest second-price. We propose a novel "virtual world" construction to address this problem, and make truthful revelation of a user's requirements, e.g. value, storage capacity, and duration, a dominant strategy.

The underlying auction mechanism, used in each round, is the strategyproof and approximately-efficient auction due to Lehmann et al. [LOS99]. Individual auctions are for a total storage capacity, starting now, and for a particular duration. Any bids in the auction pool that would be satisfied with the auction duration are included as bids in an auction, and allocated using a greedy knapsack approximation algorithm. A virtual world, created for a winning agent, is used to track the minimal price that the agent could have achieved in subsequent auctions. This provides a simple generalization of their analysis to a sequential setting, and recovers strategyproofness.

The basic runtime complexity is $O(PM \log M)$, where there are P pools, and $M = N/P$ is an upper bound on the number of requests at any time that can be active in any one pool, given an upper bound of N on the total number of active requests at any one time. In practice, we would expect multiple pools to run on separate servers, with additional pools spawned as necessary to provide scalability. Currently, we utilize B-tree and secondary indexing facilities provided by an embedded database to speed up data access.

In Section 2 we describe the data staging problem, introduce the salient features of the corresponding mechanism design problem, and provide a high-level overview of our solution. Section 3 describes the virtual world construction, and proves the buyer-side strategyproofness properties. Section 4 describes the *pooling and consignment* method, which we use to bring buyers and sellers together into a sequence of auctions, and proves the seller-side strategyproofness properties. Section 5 introduces our experimental methodology. This leads to Section 6, in which we present efficiency and strategyproofness results. Section 7 finishes with a critical discussion of the current system, and directions suggested for future work.

1.1 Related Work

Nisan & Ronen [NR00, NR01] initiated the study of algorithmic mechanism design (AMD) within the theoretical CS (TCS) community. The approach focuses on strategyproofness, and on the resource-bounded nature of the mechanism *infrastructure*, for example on a centralized auctioneer [LOS99, BGN02]. Recently, Feigenbaum and colleagues [FPS01, FPSS02] have initiated a research agenda in *distributed* algorithmic mechanism design. The goal is to distribute the *entire* computation of a mechanism across a network of agents.[1]

[1] A key challenge is that the *commitment* provided by a mechanism to implement a particular outcome rule is lost, and agents must also receive incentives to perform appropriate *computation*, and implement the intended mechanism.

Traditional approaches to dynamic revenue and inventory management in operations research are non game-theoretic, and assume exogenous bid distributions (e.g. [Bel87]). There is some previous work in mechanism design for dynamic systems (e.g. [VvRM02, LN00]), but previously the problems studied have been one-sided, and for fixed capacity auction problems, quite different from the two-sided infinite horizon setting that we consider in this paper.[2] There is a large literature on the game-theoretic analysis of dynamic queuing and resource sharing problems in networking (e.g. [FNY89, She94, KLO95]), but the approaches are predominantly Nash equilibrium based, and have deemphasized strategyproof implementations.

There has been much work that attempts to advocate the use of market methods within traditional computer systems problems. Spawn [WHH$^+$92] was a classic paper that uses markets to support distributed allocation. In the domain of distributed database, Mariposa [SAL$^+$96] enables bidding for running query optimization on self-interest servers. Nemesis [NM01] demonstrates energy management within mobile systems using market methods. Popcorn [NLRC98] provides a Java-based, centralized market infrastructure for distributed computing on the Internet, and supports sequential Vickrey auctions, and a double auction with clearing across asks and bids.

1.2 Relation to Service Discovery

We are providing an alternative to traditional *service discovery* protocols. Service discovery enables client lookup of services in dynamic systems, and service advertisements of what are available. Services can range from basic ones such as printing to advanced business web services. Discovery is typically conducted via discovery servers, which accept and maintain service advertisements and query client requests for services. Some popular protocols include JINI [Mic99] and UPnP [MSR]. A service that wants to be discovered needs to submit its advertisement to the discovery server. The advertisement is typically key/pair attributes that describe the service in general. For a printer, its advertisement may contain: "Service:

[2]Blum et al. study the online winner-determination problem in a double auction, but provide no incentive analysis.

Print," "Type: Laser," "Color: Yes," etc. Similarly, a client sends a query describing its desired service attributes.

Most protocols will return all services that can, in principle, perform the service. The client must then select the service with the best fit. A particular problem in this selection task is that most protocols do not provide good support for updated service state information [FDC01], and hence clients typically need to contact each service individually to query its state. This gets particularly unreasonable in dynamic environments, for example in our Times Square scenario.

2 Example: Dynamic Data Staging

In this paper we focus on a dynamic data staging problem with the following characteristics:

request agents There are multiple request agents (RAs), indexed $i = 1, \ldots$. Each RA, i, arrives into the system at time t_i, and has value v_i for s_i MB of storage for a period of l_i secs. Requests also have a *timeout*, Δ_i, to indicate how long the request remains valid.

service agents There are multiple service agents (SAs), indexed $j = 1, \ldots$. Each SA, j, arrives into the system at time t_j, and has capacity C_j MB of storage, that is available for L_j secs. All SAs are assumed to have zero marginal cost.

One of the assumptions that we make here is that the SAs have no switching costs, for swapping RAs in and out of their capacity.

We assume self-interested agents with quasi-linear utility functions. We also assume that there is a method to implement payments within the mechanism. With this, the utility, $u_i(x_i, p)$, to RA, i, for allocation, x_i at price p, is defined as $u_i(x_i, p) = v_i - p$, if $t(x_i) \leq t_i + \Delta_i$, for $s(x_i) \geq s_i$, and $l(x_i) \geq l_i$, and $u_i(x_i, p) = -p$ otherwise. Notation, $t(x_i)$, denotes the start time, $s(x_i)$ denotes the size, and $l(x_i)$ denotes the length. The utility to SA, j, is equal to the total payments received in the mechanism. The SA's are assumed to have zero marginal cost for providing services.

We measure performance in terms of the total value to the RAs for the matches, as a fraction of the total possible value if all RAs had been matched. This provides a challenging comparison, in particular as the system becomes loaded. Alternative metrics would normalize by the value computed with the optimal offline algorithm, or by the value computed with the optimal online algorithm. We get additional insight into the performance by comparing with the performance of a simple greedy, and non-strategyproof, matching algorithm. This is described in Section 7.

Another useful metric is the *utilization* of the system, which is the total amount of capacity allocated to RA's (in MB secs) divided by the total capacity over the SA's that are *admitted* into the auction.[3] We also measure the total capacity that is made available for sale in an some auction, and refer to the *competition* of the system as the total amount of capacity allocated to RA's divided by the total capacity that is up for sale.

2.1 The Mechanism Design Problem

As a mechanism design problem, dynamic data staging is interesting for a number of reasons:

impossibility There is a general impossibility result [MS83], that arises purely as a result of incentive-compatibility requirements, for implementing perfectly efficient outcomes in two-sided markets (with buyers and sellers) without injecting money into the system.

combinatorial The winner-determination problem, even in just one auction with capacity, C, and with multiple indivisible bids with weight w_i and price p_i, is the classical 0/1 knapsack problem, and among the best known of NP-complete problems.

dynamic RAs arrive dynamically, and can strategize about the timing of their bid and *which* auction to join. Similarly, SAs arrive and depart dynamically.

Of course, we can add to this list the computational requirement that the mechanism be fast and scalable.

[3]The number of SA's in the system is constrained at any particular time to maintain a minimal auction frequency for each SA.

2.2 Overall Architecture

The exchange is designed to accept SA and RA messages anytime and to conduct auctions continuously. Figure 1 depicts an overall architecture of the exchange. All new RAs to arrive are queued up in the system. The RAs can state a size, a duration, and a value. Before starting a new round of auctions, each queued RA is assigned to join one of the auction pools, in round-robin fashion. An RA can only remain in the pool until its timeout.

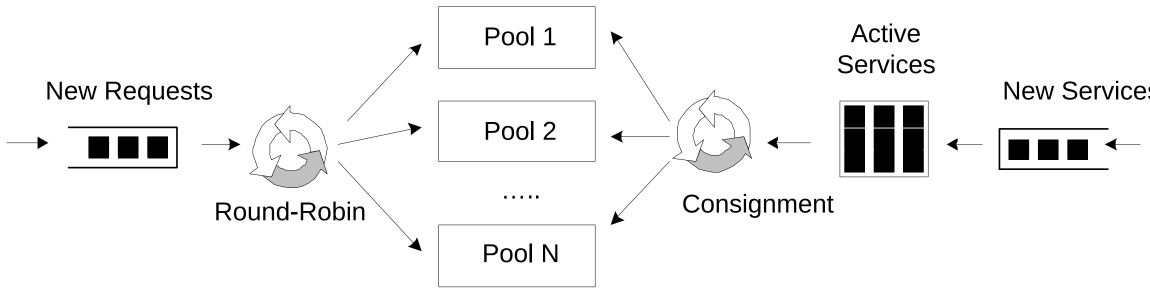

Figure 1: Exchange architecture

Similarly, the new SAs to arrive are put in a new services queue. New SAs are imported into the current set of active SAs every τ seconds, and the *consignment* is revised for each SA. The goal of consignment is to divide up the capacity of each SA into chunks to be auctioned in individual auctions. The decision is based on the number of SAs in the systems, the number of auction pools, the frequency of auctions, and also the demographics of the RAs, in terms of the durations they request. Once consignment is calculated the SAs in the active set are assigned to the pools in round-robin fashion. There is a limit to how many SAs can be assigned in a single consignment. This is explained in Section 4.

All pools perform sequential auctions over time. All pools will be running at most one auction at any time. The pools operate independently of each other and are monitored by the exchange. The basic information a pool needs to start a new auction are the current RAs assigned to the pool, and the SA assigned to the next auction, along with the chunk of capacity that it will make available (as determined in consignment).

The pool maintains multiple *allocation states.* One state defines the actual "real world" state– the matches that the pool has implemented between RAs and SAs. The pool also maintains an additional “virtual world” state for each winner, w, that is still active (has not timed out). This state defines the matches that the pool would have implemented without that one winner, w. The virtual world state also defines the tightest current upper-bound on the winner’s actual payment. Figure 2 shows auctions within a pool and the real and virtual worlds associated with it.

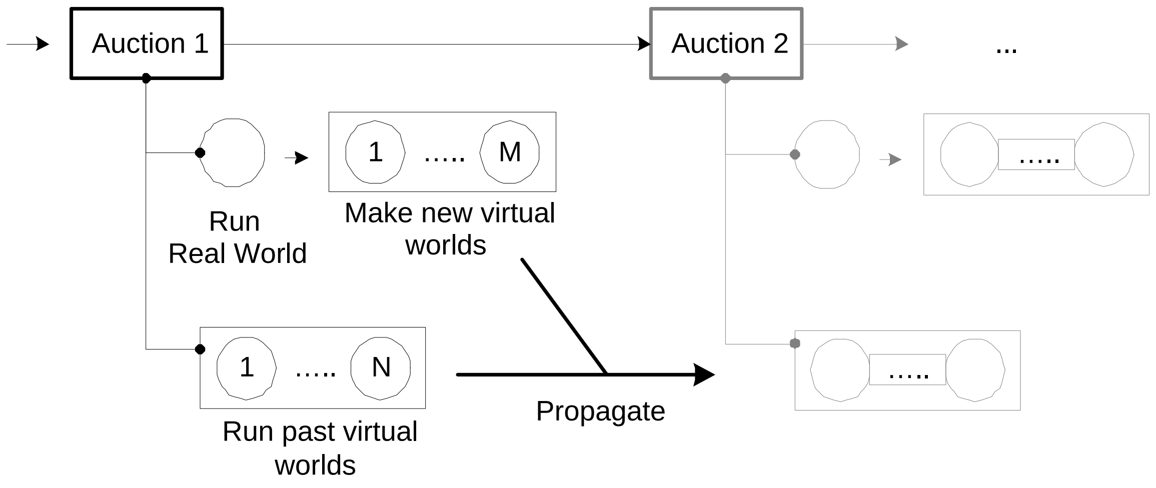

Figure 2: Virtual Worlds

At the start of a new auction, the pool introduces the new RAs into the real world state and calculates the new winners in the real world and an initial value for the payment of each winner. Next, a new virtual world is created, for each new winner, to represent the state if the auction had been cleared without that particular winner. The payment is initialized to the payment computed for the winner in the real world period in which it wins. Then, each virtual world associated with a winner from a previous system that is still active is updated, based on the new RAs to arrive in the current period. The payment for the winner, as maintained in the virtual world, is revised downwards if the winner would have been a winner in this period of its virtual world, and for a lower price than its current payment. Think of this new payment as the price the RA would have paid if it had timed its arrival and bid in this auction.

3 Component 1: Strategyproof Sequential Auctions

First, in this section we propose a fast and strategyproof design for a sequential multi-unit auction. We consider a simplified version of the data staging problem in which bids have two attributes– size and price –and in which there is already a fixed schedule of capacity to sell in a sequence of auctions.

Lehmann et al. [LOS99] previously proposed a fast and strategyproof auction for a one-time "single-minded" combinatorial auction. In the single-minded combinatorial auction, each bidder has value for a particular bundle of items. The idea is that a winner pays, per good in its bid, the average bid price of the first bid that is displaced from the outcome because of her bid.

As a multi-unit auction, with C units for sale, the auction collects bids (s_i, v_i) from bidders, $i = (1, \ldots, N)$, for s_i units at total price v_i. Let $p(i) = v_i/s_i$ be the average bid price per unit. Let L be the sorted list of bids, sorted in decreasing order, with $p(1) \geq p(2) \geq \ldots$. Let $x = (x_1, \ldots, x_N)$ denote an allocation. An allocation is feasible if $\sum_i x_i s_i \leq C$. Consider a greedy allocation algorithm, that takes the sorted list, L, and allocates the first bid, and then examines each bid of L in order and allocates it if the new allocation is still feasible. Let x^* denote the allocation computed by the greedy allocation with all bids, and x^{-i} denote the allocation computed without bid i. Now, for any winner, i, let $n(i)$ denote the first bid (or ϕ if no such bid exists) following i in order L that was denied in x^* but is accepted in x^{-i}. The auction implements allocation x^*, and agent i pays zero if her bid is denied or if there is no bid $n(i)$, and $s_i \cdot p(n(i))$ otherwise.

Lehmann et al. [LOS99] establish the strategyproofness of their auction by appeal to the following sufficient properties:

Exactness: bids are either accepted in full or denied.

Monotonicity: if a bid (s_i, v_i) loses, then a bid (s'_i, v'_i) with $s'_i \geq s_i$ and $v'_i \leq v_i$ also loses.

Critical: the payment is exactly equal to the minimal price at which a bid, for the true size, would have still been accepted.

Participation: only winners make payments.

Note that the *Critical* property implies that the payment is independent of the bid price, because *Exactness* and *Monotonicity* taken together imply that the critical value is defined by the bids of the other agents.

We state an immediate corollary of Theorem 4 [LOS99] for multi-unit auctions.

Theorem 1 *[LOS99] In a multi-unit auction with single all-or-nothing bids, if the auction satisfies Exactness, Monotonicity, Participation, and Critical, then it is strategyproof.*

Let us now turn to the sequential multi-unit auction problem. Let $k = 1, 2, \dots$ denote the sequential auctions, and suppose that bidders $i \in A_k$ arrive in auction period k. Bidder i has value v_i for s_i units, and is indifferent between receiving the units now, or in any period $k' < k + \Delta_i$, where Δ_i is the *patience* of the bidder. Each auction is capacity C_k.

Consider the following auction, VirtualWorlds, which is a multi-period extension of Lehmann's auction. VirtualWorlds maintains a set of *active bidders*, N_k, at the start of each period, k, which are the bidders that are within their patience *and*, either losing, or winning at some price $p_{\text{virt}}(i) > 0$. This price is the *virtual world price*, $p_{\text{virt}}(i)$, of bidder i, and is maintained by the auction for every winning bidder as long as it is still active. Let $W_k \subseteq N_k$ denote the set of active bidders that are already winners, at the start of period k. Every winner has an associated *virtual world* while it is active. This virtual world maintains the set of active bidders that would be winners, $W_k(i)$, at the start of period, k, if winner i had never submitted its bid, along with the virtual world price.

With this, the auction proceeds with the following steps, in each round k:

1. Remove all agents, i, from the active set, whenever they exceed their patience, or they are winning in the real world and have a zero price, $p_{\text{virt}}(i) = 0$.

   –*whenever a winning agent is removed, then collect its final payment.*

2. Add new arrivals, A_k, to the active set, N_k.

3. Run [LOS99] on the set $N_k \setminus W_k$, and add all new winners to W_k. Initialize a virtual world for every new winner, i, with $p_{\text{virt}}(i)$ initialized to the price just computed in the

real world auction.

– *implement the matching for any new winner.*

4. Run [LOS99] on agents $N_k \setminus W_k(i)$ in each virtual world. (Note this auction includes agent i, now bidding as though this was the first time it entered the system.) If bidder i is a winner, then update the price, $p_{\text{virt}}(i)$, to the minimal across the current price and the price just computed for i in this virtual world. Either way, propagate the solution to the greedy allocation algorithm run on agent $N_k \setminus (W_k(i) \cup i)$, i.e. without bidder i, to the next period.

Define the *critical value*, p_c, for agent i in VIRTUALWORLDS, as the minimal price that the agent can bid in the auction and still be a winner, given a set of bids from other agents, and given a particular size s_i. Also, define the critical value, p_{ck}, as the minimal price that agent i can bid in period k, and be a winner in period k, assuming that the agent can wait and announce its arrival directly into period k.

Theorem 2 *Auction* VIRTUALWORLDS *satisfies Exactness, Monotonicity, Participation, and Critical, and therefore truth-revelation of v_i and s_i is a dominant strategy.*

Proof:
Exactness is trivial, and Monotonicity is immediately inherited from [LOS99] because if a bid loses it loses in each of a sequence of [LOS99] auctions. Critical follows from Lemma 1. Consider a bid, $v' > p_c$, and show that the payment computed in VIRTUALWORLDS is exactly p_c. First, we know that the bid is successful, in some period k, by the definition of p_c. Assume, without loss of generality, that the bid arrives in period 1, so that the last period within patience is period Δ_i. We know that $p_c = p_{ck'}$ for some $k' \geq k$, and $k' \leq \Delta_i$, because otherwise we would need $p_c = p_{ck'}$ for some $k' < k$, by Lemma 1, and the bid would have already been accepted in an earlier period. Now, we explicitly compute payment $\min\{p_{ck}, p_{c\Delta_i}\}$ in VIRTUALWORLDS, and implement the critical value. Participation trivially holds. ■

Let $\underline{k}_i$ denote the period in which bidder i arrives into the auction. Let $\overline{k}_i = \underline{k}_i + \Delta_i - 1$ denote the final period in which bidder i is happy to receive the items.

Lemma 1 *The critical value, p_c, in* VirtualWorlds*, equals* $\min\{p_{c\underline{k}_i}, \ldots, p_{c\overline{k}_i}\}$.

Proof:
By contradiction. First, assume that there is some k', between $\underline{k}_i$ and $\overline{k}_i$, for which $p_{ck'} < p_c$. This is a contradiction because an bidder in VirtualWorlds can payment $p' \leq p_{ck'}$ by submitting a bid with price $v_i' = p_{ck'}$. If this bid is successful in an auction before k', then the payment can only be less than the bid price. Otherwise, this bid will be, by definition, just competitive in auction k', and is a critical value for RealWorlds. Second, assume that $p_c < p_{ck'}$, for all $\underline{k}_i \leq k' \leq \overline{k}_i$. But, this implies that a bid, p_c, will be successful in some period, k'', in the interval $\underline{k}_i \leq k'' \leq \overline{k}_i$, and therefore $p_{ck''} \leq p_c$. ■

Note carefully that we state a strategyproofness result *only* with respect to the (s_i, v_i) part of a bidder's type. The strategyproofness does not extend to bidders that are able to misrepresent their arrival time, or misrepresent their patience.

First, consider the effect of announcing a larger patience. As an example, suppose that an agent that arrives in period 1 with patience 5 is matched in period 1, and that its payment is calculated based on the state of its virtual world in period 4. This agent can extend its patience beyond 5 *without risk*, because it will always be matched in period 1 and stating a larger patience increases the number of periods in which its virtual world payment can be refined downwards.[4] Similarly, consider the effect of delaying arrival time. As an example, suppose that an agent that arrives in period 1 with patience 5 is matched in period 4 and makes the payment computed in period 4. This agent can hope to lower its payment by announcing its arrival in period 4, and thus extending the horizon of its patience and the potential for a lower virtual world payment. Thus, even if the mechanism simply assumes a

[4]On the other hand, if the same agent was not matched with patience 5, but was matched in some period, 7, by announcing a patience of 10, then the agent would lose utility because it would make a payment for an allocation for which it has no value.

fixed patience for all bidders, the mechanism is still not strategyproof with respect to bidders that can delay their arrival. This holds true even if the patience is exactly one.

4 Component 2: Pooling and Consignment

Second, we describe a method to construct consignments from SA capacity, and a method to divide arriving RAs into pools to keep the architecture scalable.

The key differences between the data staging scenario and the problem described in the previous section are:

1. multiple sellers, dynamically arriving, and with different numbers of units for sale and for different lengths of time.

2. self-interested, rational, sellers that would be expected to misrepresent their available capacity if that can improve their expected utility.

3. an additional dimension in bid space, the *duration*, of time that the RA requests.

Given that the incentive properties of auctions such as [LOS99] are not well understood when there are multiple sellers in the same auction (this provides a flavor of a combinatorial exchange [PKE01]), we elected to assign a single SA to each auction.

One can imagine a number of schemes to assign SAs to auctions. It is helpful to consider the following two alternatives: Scheme (A) in which each SA receives access to the auctions within a pool in proportion to the total reported capacity of an SA; and Scheme (B) in which each SA receives access to the same number of auctions within a pool, irrespective of its reported capacity. In Scheme A an SA can take revenue away from other SA's by overstating its capacity, and getting access to more auctions. This is probably especially useful when supply exceeds demand in the system, and this is also precisely the case in which an SA would be able to follow such a strategy without much risk to over exposure to RAs, with the consequent potential for detection. We follow Scheme B, and allocate SAs in a round robin style to the auctions, as they are scheduled.

The next task is to decide how to divide the reported capacity of an SA across the auctions in which it is scheduled. The data staging model allows RAs to request jobs of size s_i (MB) for a period l_i (s). Let us refer to the period l_i, as the *type* of the good. So, 100 MB of 10s capacity is the same type of good as 5 MB of 10s capacity. In the consignment step, we divide the capacity of an RA first into different types of goods, and then into consignments of a single type of good for individual auctions. The consignment algorithm is free to choose the appropriate types of goods to sell in auctions, and the total division of capacity across types (e.g. capacity available for 5s data staging vs. 1hr data staging).

Let A_j define the auctions/sec assigned to SA j. Then, given classes, $\mathcal{L}$, of goods, and a top-level decision to divide total capacity, into fraction f_l for each class, $l \in \mathcal{L}$, and t_l (s) duration for class l, we allocate $f_l A_i$ auctions/sec to type l. Setting the capacity in each auction to meet the total capacity goals, this reduces to $C_j/(A_j t_l)$ MB of goods in each class l auction. This is valid while the auction periods are fast enough, while $1/(f_s A_j) < t_l$. Otherwise, we need to cap the capacity by another term, to prevent an overflow in capacity between successive auctions. We obtain the following overall consignment rule:

$$\min\left(f_s^2 C_j A_j t_l, \frac{C_j}{A_j t_l}\right), \tag{1}$$

to determine the total capacity that SA, j, to sells in each auction of good class l with duration t_l.

We are careful to allocate an RA, i, that reports type, (s_i, l_i) to any auction with duration, $t_l \geq l_i$. Otherwise, if we strictly matched RAs to SA auctions based on the duration, l_i, then there could be an incentive for an RA to overstate its required duration in order to get into larger capacity auction with a lower price. Auctions of different types of goods are just scheduled in a random sequence within each pool, and the virtual worlds methods work seamlessly across multiple types of auctions.

There is one remaining opportunity in the Exchange for strategic behavior. An individual SA can understate its capacity in an attempt to reduce supply in a particular auction and drive up the second prices. However, as a pleasant side effect of the virtual worlds method, this opportunity is mitigated in practice because the prices are determined over a sequence of auctions, all operated by different SAs.

In summary, we can state some theoretical properties for the combined virtual worlds and pooling and consignment architecture, which we call here the *Exchange*.

Theorem 3 *The Exchange is strategyproof for RAs, with respect to value, size, and duration (but not for delayed arrivals or extended timeouts). An SA can never benefit from stating a larger capacity, but can sometimes benefit from understating its capacity.*

5 Implementation Overview

To test the exchange concept in a realistic distributed systems settings with latency and failures, we choose to implement the exchange, SAs, and RAs as distributed program entities that can communicate efficiently using standard messaging formats.

We adopt Sun Microsystem's Project JXTA platform [Gon] as our implementation architecture. JXTA is a peer-to-peer architecture that supports formation of groups of peers, discovery of peers and services peers provide, sending/receiving XML messages, and peer-level message routing/multicasting mechanisms. JXTA is not an API, but a protocol that gives us all the necessary core services yet provides flexibility. In fact, JXTA software can be created and deployed in different programming languages and OSes and will support heterogeneous distributed environments.

The exchange is written in Java (SDK 1.4.1) and currently runs on a stand-alone Windows XP Professional machine with a 1.8GHz Pentium IV processor and 512MB of RAM. Berkeley DB [Sof]is used as an embedded database that provides efficient management and operations of the auction pool and virtual world data structures. The exchange registers itself in a JXTA group and listens continuously for SAs and RAs messages sent within the group. SAs and RAs are distinct Java programs on different machines that belong to the same LAN as the exchange. SAs and RAs discover and join the same group that the exchange resides. Agents then send messages to submit requests and service announcements to the exchange.

6 Experimental Methodology

6.1 Goals

We want to prove that our scheme, despite its overhead compared to lightweight discovery protocols that do not provide handshaking (i.e. *matching*), provides high allocative efficiency for RAs. The main variable in our tests is RA/SA load. This is defined as the number of RAs arriving in the exchange per second divided by the number of SAs arriving per second. We would like to see much load the current implementation can support, and the effect of load on allocative efficiency.

We ask the following questions in designing our experiments:

1) What is the allocative efficiency created in the system? Namely, what is the amount of value created among all RAs due to matching? Because we normalize with the total possible value, if all RAs are served (see Section 2), we expect the overall efficiency to fall as the load increases.

2) How effective are the auctions at achieving a high utilization of resources? Specifically, when the system allocative efficiency falls, does it fall because the server capacity is reaching full utilization or because the exchange does an inefficient job of matching capacity and jobs in auctions?

3) Does an individual RA achieve greater value in our mechanism than with a generic discovery protocols, that simply returns a list of SAs that can in principle match its request and leave the matching to the RA?

4) Is the system strategyproof in practice, for both RAs and SAs? Specifically, can an agent gain, on average by misstating its private information, such as the duration of time that it requires for data staging?

6.2 Experimental Distributions

We setup our experiments with the following distributions for different parameters. Everything is fixed, except for the load which we adjust by changing the arrival frequency of RAs while fixing the arrival frequency of SAs.

Exchange parameters:

We have two pools in the exchange. There will be one auction per second for the pools. Consignments are updated every five seconds. The resource durations sold are either five seconds or ten seconds, and in equal total capacity (in MB seconds).

SA parameters:

The SA arrival rate is one per second. The SA capacities are uniformly distributed in [50, 100] (MB), and offer data staging for a total duration uniformly distributed in [25, 50] (sec).

RA parameters:

The RA arrival rate varies depending on the load. The capacity requirements and the values of the RAs are uniformly distributed in [1, 10] (MB) and [1, 10] (dollar), respectively. The duration requirements are at random from 5, 10 (sec) with equal probabilities.

The RAs have timeouts uniformly distributed in [5, 15] seconds. For these experiments we assume that the RAs announce these timeouts truthfully.[5]

6.3 Naive Discovery Algorithm

We want to compare our implementation to a naive, non-strategyproof discovery protocol. As described in Section 1.2, many current protocols do not store live states of services and only match with the static attributes in advertisements. We assume that an RA that uses such a protocol will look at a fixed subset of the SAs returned in the list (20% in our naive implementation), asking for the currently available capacity and duration, until a match is found or failing if no match is found.

We process the RAs in the queue periodically. For each RA, the RA looks at most 20% of the SAs returned by service discovery, and either finds a match based on current capacity and duration or fails. After a match we still factor in the non-strategyproofness of the protocol, and assume that the SA's actual available capacity is between 0 and 50% less than its stated (uniformly distributed in this range). Similarly, we assume that the SA's actual available

[5]We propose to actually fix the timeout within the system to some constant value for all RAs to prevent time-based manipulation using this parameter.

duration is between 0 and 20 (secs). If both the true size and the true duration still matches the RA's needs, then we have a match. After a match the SA's capacity will be deducted accordingly for that round. If an RA ever fails to find a match it leaves the system.

7 Results

Figure 3 depicts the first set of results, in terms of the metrics, *efficiency* (normalized by total possible value), *utilization* (fraction of admitted capacity that is sold), and *competition* (fraction of auctioned capacity that is sold), that were introduced in Section 2. Each point on the chart represents values obtained from running the exchange for 500 seconds (or 100 consignment iterations). For each curve, we ran with ten different RA/SA ratios, between 2 to 40.

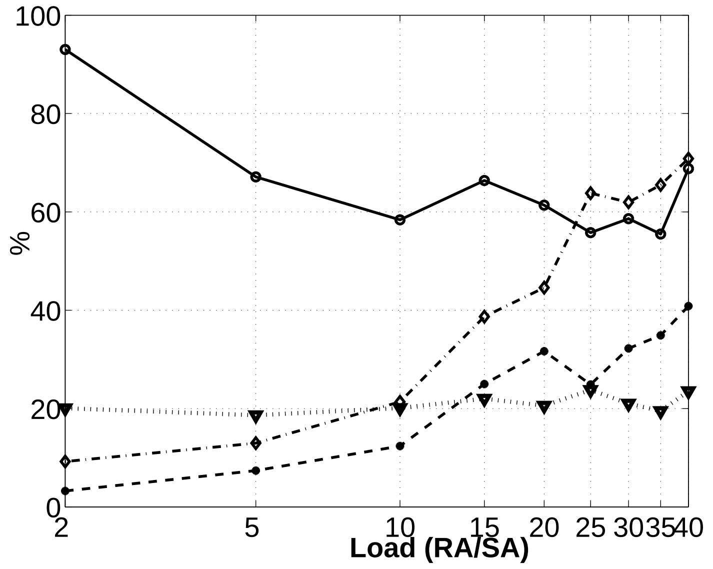

Figure 3: Efficiency, Competition and Utilization vs. System Load. **Key:** -o- exchange efficiency; -.- competition; – utilization; ... naive efficiency

The efficiency is close to 100% for low loads, and remains quite high, at around 60% even as the competition in the system, for resources that are brought to auction, increases to around 70%, which occurs at high RA/SA ratios. The utilization measurement shows low utilization at low loads and gradually improving utilization as load increases. Ideally, we would like utilization to be higher than this as load increases, to counteract the increased competition within auctions. We suspect that utilization can improve if the pooling technique is smarter. An example technique will spawn new pools when certain thresholds of SAs are exceeded.

In comparison with the exchange, the naive discovery protocol averages around 20% efficiency, even for low loads. Increasing the search space of the agents in the naive from 20% to 30% was found to improve efficiency to about 25 to 27%. Notice, though, that in dynamic environments it might just take too long to contact a large number of SAs and query their state.

We also ran experiments to verify strategyproofness. First, we hold the value of an RA fixed (at 100), and test its overall utility when it overstates its storage requirement (from 100MB). Figure 4 shows what happens when the RA overstates by 0, 25, 50, 75, and 100 percent. Utility for the RA drops as much as 40 dollars when it reports twice the storage requirement.

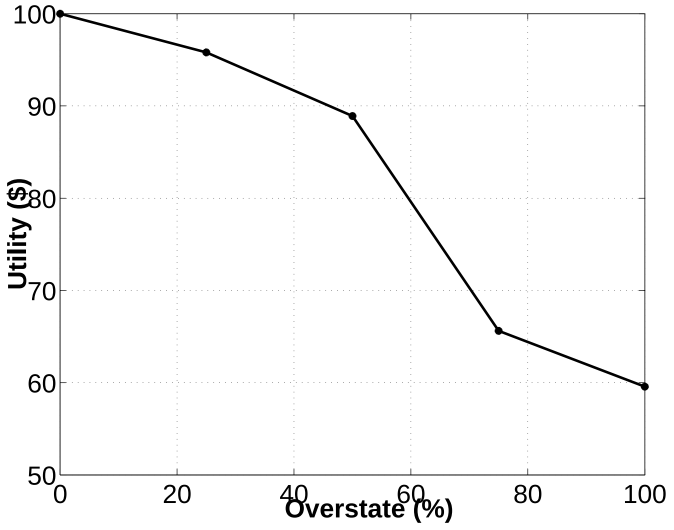

Figure 4: Strategyproofness: RAs and Size

Second, we hold the storage fixed, and let the RA understates it value. The motivation for the RA is to try to have a receive a lower final price. Figure 5 shows that RA suffers similarly to when overstating size.

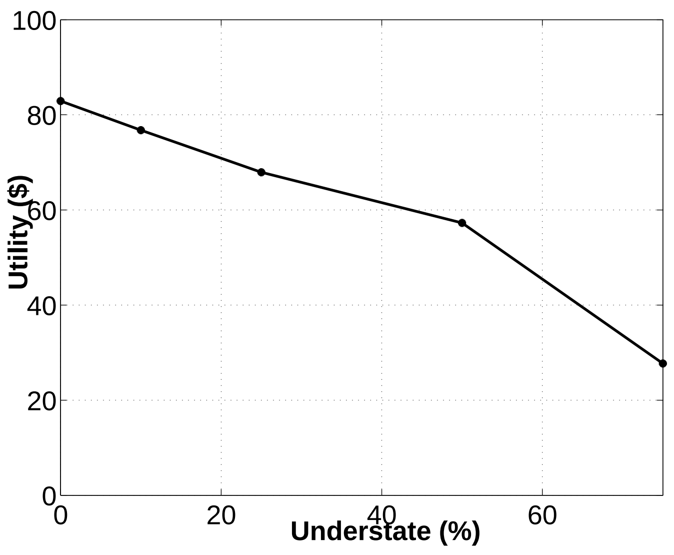

Figure 5: Strategyproofness: RAs and Value

We are currently running additional experiments to benchmark the virtual world overhead, and also for SAs with lower capacities and/or RAs with more individual demand to test performance in a high-utilization regime. Currently, with two pools and one auction per second in each pool, and fresh consignments every five seconds, the auctions begin to take longer to clear than allowed by the auction schedule. This prevented us from running experiments to higher loads in order to demonstrate the behavior for higher still utilization levels. As the number of RAs increases, and the number of winning RAs increases, the overhead of running virtual worlds increases.

The system scales reasonably well for the current example. Currently, the system supports about 40 jobs per second, for two pools, one auction each per second. That is over 2,000 jobs per minute and is acceptable for many applications. Nonetheless, our example is still simplistic in terms of number of attributes. Ultimately, we believe that the right way to address future large-scale implementations is through the use of multiple pools in parallel, and multiple distributed exchanges, as well as through open systems in which exchanges can compete to provide services to RAs and SAs.

8 Conclusions and Open Problems

We have proposed an architecture and mechanism for a strategyproof exchange for dynamic resource allocation between rational and self-interested agents, and considered its application to a dynamic data staging scenario. We have proposed a method to allow a sequence of auctions to draw from the same pool of bids, without compromising strategyproofness for bidders. We have also proposed a consignment protocol to partition incoming server capacity for schedule to individual auctions, without allowing an SA to gain access to additional auctions by overstating its capacity.

There is still much left to be done. In immediate work, we intend to perform many additional experiments to fully characterize the performance and strategyproofness of the system, across a rich variety of RA and SA distributions. In addition, we intend to carefully benchmark the virtual world experiments, and to investigate the power that the ability to dynamically introduce new pools can bring in constraining the maximal number of winners in any one pool and keeping a control on the virtual world.

For the future, we would like to explore methods to allow limited aggregations, for example with multiple SAs in a single auction, and to move a little further away from single-minded bidders, for example with one more bid dimension, such as bandwidth, in addition to capacity and duration.

References

[Bel87] Peter P Belobaba. Airline yield management: An overview of seat inventory control. *Transportation Science*, 21(2):63–73, 1987.

[BGN02] Y Bartal, R Gonen, and N Nisan. Incentive compatible multi unit combinatorial auctions. Technical report, The Hebrew University of Jerusalem, 2002.

[MSR] Microsoft Corporation 1999. Universal Plug and Play: Background Whitepaper.

[FDC01] Adrian Friday, Nigel Davies, and Elaine Catterall. Supporting service discovery, querying and interaction in ubiquitous computing environments. In *Second ACM international workshop on Data engineering for wireless and mobile access*, pages 7–13. ACM Press, 2001.

[FNY89] D Ferguson, C Nikolaou, and Y Yemini. An economy for flow control in computer networks. In *Proc. 8th Infocom*, pages 100–118, 1989.

[FPS01] Joan Feigenbaum, Christos H Papadimitriou, and Scott Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63:21–41, 2001.

[FPSS02] Joan Feigenbaum, Christos Papadimitriou, Rahul Sami, and Scott Shenker. A BGP-based mechanism for lowest-cost routing. In *Proceedings of the 2002 ACM Symposium on Principles of Distributed Computing*, 2002. to appear.

[FS02] Joan Feigenbaum and Scott Shenker. Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, 2002.

[Gon] Li Gong. Project jxta: A technology overview. Whitepaper.

[KLO95] Y Korilis, A Lazar, and A Orda. Architecting noncooperative networks. *Journal on Selected Areas in Communications*, 13:1241–1251, 1995.

[LN00] R Lavi and N Nisan. Competitive analysis of online auctions. In *Proceedings ACM Conference on Electronic Commerce*, 2000.

[LOS99] Daniel Lehmann, Liadan O'Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *Proc. 1st ACM Conf. on Electronic Commerce (EC-99)*, pages 96–102, 1999.

[Mic99] Sun Microsystems. Jini technology architectural overview. Whitepaper, 1999.

[MS83] Robert B Myerson and Mark A Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 28:265–281, 1983.

[NLRC98] N Nisan, S London, O Regev, and N Camiel. Globally distributed computation over the Internet– the POPCORN project. In *Proc. 18th Int. Conf. on Distr. Computing Systems*, 1998.

[NM01] Rolf Neugebauer and Derek McAuley. Energy is just another resource: Energy accounting and energy pricing in the nemesis os. In *8th Workshop on Hot Topics in Operating Systems (HotOS VIII)*, May 2001.

[NR00] Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. In *Proc. 2nd ACM Conf. on Electronic Commerce (EC-00)*, pages 242–252, 2000.

[NR01] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

[PKE01] David C Parkes, Jayant R Kalagnanam, and Marta Eso. Vickrey-based surplus distribution in combinatorial exchanges. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.

[SAL$^+$96] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. *VLDB Journal: Very Large Data Bases*, 5(1):48–63, 1996.

[She94] S Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 47–57, 1994.

[Sof] Sleepycat Software. Website.

[VvRM02] Gustavo Vulcano, Garret van Ryzin, and Costis Maglaras. Optimal dynamic auctions for revenue management. *Management Science*, 2002. to appear.

[WHH$^+$92] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *Software Engineering*, 18(2):103–117, 1992.