Computational Resource Exchanges for Distributed Resource Allocation

Brent N. Chun[†], Chaki Ng[*], Jeannie Albrecht[‡], David C. Parkes[*], and Amin Vahdat[‡]

[*]Harvard University [†]Intel Research Berkeley [‡]University of California, San Diego

Abstract

Appropriate abstractions, mechanisms, and policies for resource allocation is quickly emerging as the fundamental problem facing emerging computation and communication environments such as PlanetLab and the Grid. This paper explores the utility of one simple abstraction for global resource allocation with a number of appealing properties: a centralized auction that collects user descriptions of resource configurations and the values placed on these configurations. The task of the clearinghouse is to determine a set of winning bids and to assign appropriate subsets of global resources to individual users. One challenge with this model is the computational complexity associated with determining winners. To make the problem tractable, we propose appropriate bidding languages that constrain the type of bids that users can make, while maintaining required expressiveness. Computing optimal solutions to such auctions for scales of current interest (e.g., 1000 nodes) is intractable on current hardware, even given aggressive optimizations. Thus, we introduce a number of heuristics that appear to perform well in practice. Another challenge with auctions is the lag in clearing the auction and the uncertainty in whether resources will actually be acquired. We introduce a formulation for "Buy it Now" pricing to address some of these limitations.

1. Introduction

Emerging computation and communication environments possess a number of characteristics that set them apart from traditional environments, requiring a fundamental rethinking of the key mechanisms and policies underlying their implementation, access, and use. These systems, including PlanetLab [32] and the Grid [12, 13], are large-scale, distributed, and federated. These federated architectures are emerging because they enable access to resources, both in terms of absolute quantity and their relative global placement, that would otherwise be impossible or prohibitively expensive to acquire for any single site. However, an interesting dilemma arises. Participating sites federate with the express goal of gaining access to global resources in quantities larger than what is available locally. At the same time, the system will undergo "the tragedy of the commons" if all participants attempt to simultaneously consume as much global resources as possible. PlanetLab once again witnessed such a scenario in the days leading up the OSDI 2004 paper submission deadline, as depicted in Figure 1. The graph shows 5th, average, and 95th percentile load on 220 PlanetLab nodes as reported by Ganglia. Each node reports its 5 minute load average once per hour.

Thus, a fundamental abstraction required by any such global and federated computation and communication environment is appropriate resource allocation mechanisms and policies. Historically, resource allocation has been one of the primary problems addressed by the systems research community. However, many of the novel research techniques have gone largely unused because characteristics of dominant computing environments allowed users and organizations to adopt simple, often non-technical, solutions to the resource allocation problem. For example, in the time-shared mainframe days, resource allocation was employed by some combination of FIFO scheduling, organizational fiat, and social interactions with the mainframe operator. As the computing landscape evolved to personal computing, resource allocation was accomplished by over provisioning (to a degree) the amount of available resources. Finally, as large shared clusters of workstations have emerged, users have once again moved toward social policing and organizational fiat for resource allocation. Unfortunately, these techniques are not applicable to federated computing environments because there can be no single organizational fiat to effect global policy and system scale is typically too large to make social, "out of band" resource negotiation effective.

The key requirements for an effective resource allocation mechanism for such large-scale federated environments include: i) high aggregate resource utilization, ii) "fair" sharing of resources according to some agreed upon set of policies, iii) high aggregate "utility" for global participants, indicating that end users are typically able to access desired resources based on current levels of demand, and iv) resistance to to gamesmanship (where each site has appropriate incentives to contribute to global good). In this paper, we explore one point in SHARE, a centralized resource exchange/auction. In the exchange model, sites sell access to their resources to other global participants in exchange for centrally backed virtual currency that enables them to in turn purchase access to remote resources. We present techniques for currency distribution such that no single site is able to accumulate currency such that it becomes easy to monopolize global resources for any duration and such that sites with little of their own resources are still able to utilize global resources. In the auction based model, all resources are pooled through a central arbiter that auctions the resources off to bidders. During times of contention, per-unit resource pricing is low enabling users to gain access to a sig-

nificant share of global resources for relatively low cost. As resource contention increases, per-unit cost increases at the same time, decreasing the amount of resources any single site can obtain.

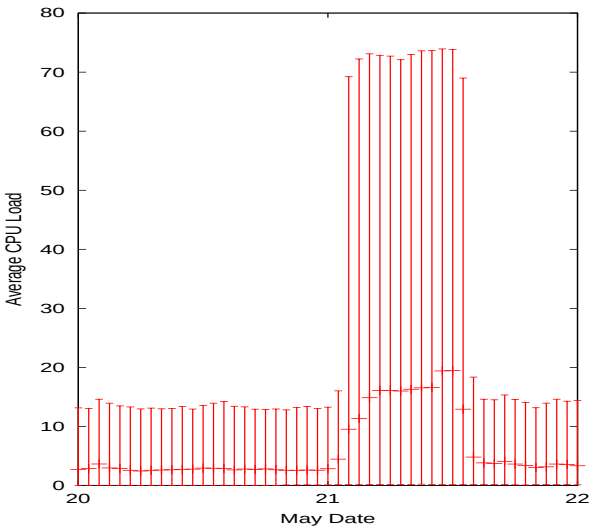

Figure 1: 5th, average, and 95th percentile load average on 220 PlanetLab nodes leading up to the OSDI 2004 submission deadline.

One important question in the designing any such exchange is how to express both buy and sell requests on the exchange or the auction and the computational complexity of determining a currency maximizing matching. For example, the ability to bid for arbitrary resources over arbitrary time granularities will quickly make the problem of finding appropriate matchings intractable. In SHARE, we propose a flexible resource description language that gives users fine-grained control over the quantity, placement, and type (e.g., varying levels of local CPU, network, memory, or storage availability) of their resources. However, we take the approach of collecting bids over a relatively long time period (e.g., 1 day) and then attempting to clear the auction, determining the best possible matchings within a fixed time window (e.g., matchings will be returned in no more than 1 hour). Beyond a certain problem size, either in terms of target resources or time window, calculating optimal solutions becomes infeasible. Thus, we propose a number of heuristics that appear to perform well in practice. We also address some of the known weaknesses with auctions—namely uncertainty in whether resources will actually be obtained and the delay in obtaining the resources even in the case of success—through "Buy it Now" pricing strategies. Here resources may be acquired immediately, though potentially at a higher cost than what might be available through the auction.

2. Computational Economy Design Space

In this section, we present a taxonomy of the computational economy design space in terms of a handful of key parameters and describe where existing approaches lie within this space. A number of previous research efforts have advocated using market-based mechanisms for distributed resource allocation. Using our taxonomy, we highlight the key design decisions made in each of the major approaches and motivate why we elected to position SHARE in its point in the computational economy design space.

2.1 System Model

The systems we target with SHARE are large-scale, distributed, and federated. Examples of such systems include wide-area testbeds such as PlanetLab and RON [1] and computational Grids. In such systems, multiple sites around the world federate by contributing and sharing computational resources with other participating sites in the system. In SHARE, we assume that these sites are, to a limited extent, cooperative. More specifically, we assume that site operators do not maliciously modify system infrastructure software. In PlanetLab, for instance, this is tantamount to assuming that sites do not try to subvert the PlanetLabOS [3] operating system or trusted infrastructure services for their own advantage.

On the other hand, we do not assume that users at the various sites are cooperative. In contrast, we assume that users will act in their own *self-interest* when trying to acquire resources for their applications and experiments. While this assumption may not hold in all cases (e.g., a pair of users may be friends and may attempt an out-of-band solution), we believe that this approach scales better as the number and diversity of users and sites grows. At PlanetLab's current size, for instance, there are already over 800 users at 161 sites in over 30 countries. This diversity in the user population already presents a significant challenge (one which has not been met) for resource allocation using social solutions.

We assume that users in the system compete for resources and attempt to maximize their own utility, where utility is a function of space and time. For example, a user might wish to acquire resources on 128 nodes, each in a distinct autonomous system (AS), for a period of two days anytime in the next week and value such an allocation at $200. Given such utilities, a resource allocation system might optimize for a number of criteria (e.g., maximize total utility, maximize minimum utility to any user, etc.).

2.2 Design Parameters

The design of a computational economy involves tradeoffs between often conflicting design goals. For example, on one hand, we might wish to allow users to express very complex descriptions of resources they wish to acquire in the economy. On the other, computing efficient resource allocations over such resource descriptions could algorithmically be very expensive and hence untenable. In building an economy, trade-offs thus need to be made for the target system of interest. In particular, we need to strike a balance along a number of key design axes which include:

- **Decentralization**: the extent to which the architecture is fundamentally centralized or decentralized.
- **Scalability**: scaling behavior both in time and space as a function of number of resources and users in the system (e.g., computational complexity to compute allocations).
- **Resource Guarantees**: whether the system provides hard (e.g., 3 Mbit/s of outgoing bandwidth) and/or soft (e.g., proportional share [43] allocations with a variable number of total tickets) resource allocations.
- **Adaptability**: timescales at which the system permits adaptive resource allocation.
- **Uncertainty and Usability**: uncertainty (e.g., waiting for an auction, whose outcome is uncertain, to clear) and usability issues the end-user faces.

- **Expressivity**: how expressive the resource descriptions submitted by the end-user are allowed to be.
- **Economic Efficiency**: how efficient (in terms of utility) the resulting resource allocations in the system are.
- **Gameability and Fairness**: susceptibility of the system to being gamed and whether it is perceived as fair by end-users.

2.3 Examples

The four main approaches taken by previous work in market-based resource allocation are: fixed pricing, congestion pricing [18, 9, 37, 26], bartering [14, 10, 5], and auctions [23, 34, 42, 25, 39, 40]. Fixed pricing is common in environments such as high-performance computing (HPC) centers and computational Grids where a collection of institutions need to mediate access to HPC resources. With fixed pricing, each site is allocated some number of service units (essentially a virtual currency) based on a research grant. Users at such sites then acquire and consume resources through a queueing system [19] and are charged service units based on the total amount of resources consumed by the application. In some cases, there might be 2-3 queues to choose from, each with a different priority which in turn leads to scaled service unit charges.

Fixed pricing in HPC environments are centralized, scale well since they employ relatively simply scheduling algorithms, and provide hard resource guarantees since resources are space shared. Their primary limitations lie in their economic efficiency and gameability. Because charges are generally fixed, users get charged for consuming resources independent of current demand. When the system is idle, users are still charged [1]. Similarly, when demand is high, the system provides no incentives for users to back off, and instead users see significant queueing delays. Finally, fixed pricing schemes used in conjunction with a batch scheduler are frequently gameable. For example, it is not uncommon (if the application permits it) to break a larger job up into multiple smaller jobs to obtain higher queue priority, thereby allowing for promotion of jobs in terms of when resources are delivered in time.

Congestion pricing is a technique that is employed in a variety of scenarios ranging from pricing of toll charges (e.g., based on freeway congestion) to pricing of network bandwidth. The primary advantages of this approach are decentralization, scalability, adaptability, and, in some simple problem domains, economic efficiency. In the context of distributed resource allocation, the primary limitations of congestion pricing are lack of hard resource guarantees, uncertainty, and economic efficiency. With congestion pricing, resources are generally priced based on instantaneous supply and demand (perhaps with a lag in time) and hence the amount resources delivered to an application varies as a function of demand. Because prices also fluctuate, so does the total amount of currency required to obtain a block of resources over time which leads to uncertainty when planning. While admission control can help mitigate these issues to some extent, this then leads to potential economic inefficiencies. In summary, congestion pricing is challenged when it comes to producing economically efficient and predictable resource allocations.

Bartering as a technique for distributed resource allocation and trading of resources in peer-to-peer systems has recently been explored by a number of research groups. Bartering itself, of course, is an old and commonly used mechanism for performing resource allocation when bootstrapping real-world economies. Its primary advantages are its simplicity and, compared to systems based on a currency system, inherent immunity to hyperinflation by definition since it does not rely on currency. In terms of our design parameters, the primary advantage of bartering in computer systems is that is naturally decentralized and simple. Bartering, as a baseline mechanism to fall back on, is hence an attractive proposition for building more efficient mechanisms on top. By itself, however, it does have drawbacks. In particular, bartering is generally economically inefficient and also suffers from lack of expressivity when complex exchanges are desired (e.g., a user wants 4 machines at Berkeley and 8 machines at MIT or nothing at all).

Lastly, auctions where users bid for resources and a matching algorithm is executed to maximize overall utility have also been proposed. In the context of resource allocation in distributed systems, the majority of such auction-based approaches have examined relatively simplistic scenarios in terms of the types of resources that can be bid on and the constraints that users might have. For example, it is often assumed that resources for all machines in the pool being auctioned off are identical and that entire machines are allocated. To address the limitations in expressivity in single-good auctions, research in the AI, theory, and operations research communities have examined combinational auctions [36, 46, 11, 35] where users bid for bundles of resources (e.g., a user might want a bundle X of nodes 13, 15, 18 or none at all) and bids may contain additional constraints (e.g., a user wants bundle X or bundle Y but not both). Such auctions enable support for a much richer class of resource requests compared to single-good auctions.

In SHARE, we leverage work from the combinatorial auction literature by formulating the distributed resource allocation problem using a combinatorial resource exchange. The key design parameters we focus on are resource guarantees, economic efficiency, uncertainty and usability, and gameability and fairness, and expressivity. Resource guarantees and predictable performance are a key requirement since best-effort allocations on a highly loaded system are difficult to perform meaningful experiments with. Such resource guarantees should be allocated to maximize economic efficiency such that users who value the resources the most get what they need. The system should be usable and allow for reducing uncertainty if desired and possible (e.g., paying a premium on a price to get resources now as opposed for waiting for an auction to clear). The system should not be easily gameable and should be perceived as fair to users. While it is generally difficult to be entire strategy-proof, it is possible to design systems such that is it is very difficult, and hence generally not worth the effort, to game the system. Finally, the system should support expressive resource requests when bidding in the system. Such expressivity is key requirement for resource allocation in large-scale federated systems given that users often select nodes to run on based on a multitude of factors and in complex ways.

3. SHARE Architecture

[1]Some supercomputing centers implement time-of-day charging, but this is only a rough approximation of being flexible to varying demand. For instance, near a conference deadline, systems are often heavily contended for during all hours of the day.

To address the distributed resource allocation problem, we have designed SHARE, a distributed resource allocation system based on a computational resource exchange. By computational resource exchange, we mean a system in which multiple users of the system buy and sell resources in a competitive market. The primary goal of SHARE is to match the resource requests from the users with available resources in an economically and computationally efficient manner. Towards that end, the SHARE system is comprised of several key pieces. First, SHARE provides a language that allows users to express sets of desired distributed resources in a succinct manner. Second, SHARE uses a virtual currency system that allows users to express the value of the resources they wish to acquire. Third, SHARE implements mechanisms and algorithms to efficiently match resource requests against available resources. Finally, SHARE supports policies for managing the distribution of virtual currency in the computational economy.

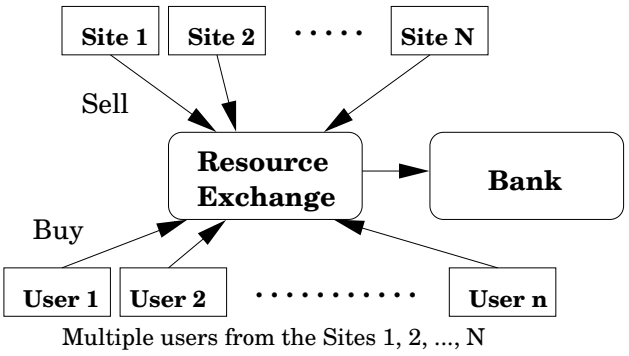

Figure 2: SHARE Architecture.

The architecture of SHARE in shown in Figure 2. The system is composed of multiple sites, each of which shares some number of machines by selling futures on machine resources in a competitive market implemented by the resource exchange. Users of each of these sites can then buy resources at remote sites through the exchange and subsequently use those resources to run their distributed applications and experiments. Three key questions that arise in this setup are (i) how do users express bids on distributed resources which span multiple nodes, are multi-dimensional, and are often exchangeable (e.g., perhaps a user wants any 10 nodes, as long as each node is in a distinct autonomous system (AS)), (ii) how do we match buys and sells on the exchange in economically and computationally efficient manner, and (iii) how does the system support policies on currency distribution that work well in practice. In the rest of this section, we describe each of key components of SHARE and discuss how they collectively address these problems.

3.1 Bidding Language

The first component of SHARE is a bidding language for expressing sets of distributed resources in a succinct manner. As mentioned, the usage model in SHARE is that users buy and sell computational resources through a competitive market by presenting bids to a resource exchange. Each bid is composed of a resource set that specifies the resources of interest and a value that indicates either the maximum amount the user is willing to pay (a buy) or the minimum amount the user is willing to accept (a sell). The complexity of the resource set in a distributed system can vary based on the needs of the user's application. In some cases, the user might wish to select nodes based purely on a set of per-node constraints (e.g., nodes with low CPU node and high availability). In others, the user may wish to select nodes based on more complex criteria (e.g., 16 nodes, each in a distinct autonomous system).

The primary design goals for the bidding language are to make it expressive enough to handle the majority of users' needs and to make such bids as simple and succinct as possible. Previous work in both distributed resource discovery [33] and combinatorial auction bidding languages [4, 28, 11] have examined this problem in some depth. In practice, we have two problems, and draw on both of these areas of work. First, users must be able to specify distributed resources of interest based on a set of constraints on per-node attributes (e.g., CPU load), inter-node attributes (e.g., inter-node latency), and logical or physical attributes (e.g., geographic region). We use resource discovery for this. The second phase is to specify additional constraints on acceptable levels of resources on each node from within this set.

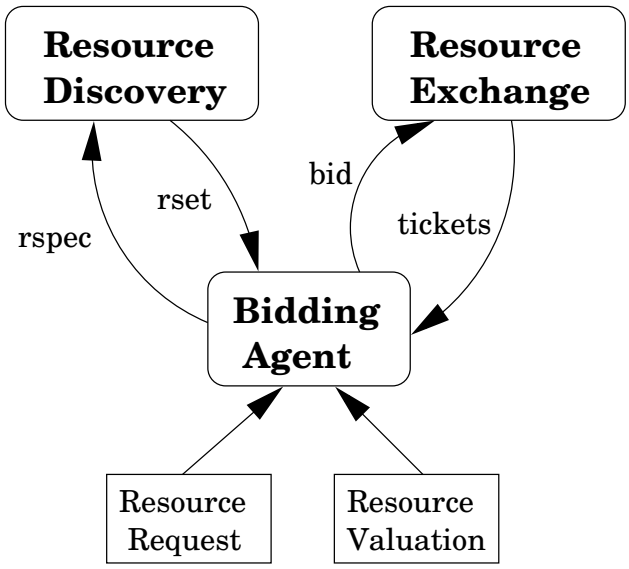

Figure 3: Bidding and Acquiring Resources.

On the buy side, the first phase involves discovering resources of interest (`rspec`) using a resource discovery system and translating those resources into a resource set (`rset`) as part of a bid on the exchange is shown in Figure 3. We assume a resource discovery system that takes abstract resource specifications as input and outputs candidates sets of nodes that meet those specifications. In contrast, on the sell side, the seller brings a concrete set of resources to the table. These resources can be immediately represented as a `rset` and thus no resource discovery phase is needed in that case.

Given this set of "acceptable" nodes, the expressiveness requirements for the second phase are dictated in terms of five key considerations:

- **Multi-unit Allocations**: In time-shared systems, users will bid for partial allocations on nodes of interest. Another way to view this is that partial node allocations are

just multi-unit allocations of the same item type (e.g., 30 CPU units on node5).

- **Complementaries**: In many distributed applications and experiments, nodes need to be co-allocated both in time and space in order to obtain meaningful results, e.g., a user may want node5, node8, and node13 from 8am to 12pm or nothing at all. Thus, the system needs to support expressing complementary resources of interest.
- **Substitutes**: Often, a user wishes to acquire resources on a set of nodes but does not care which of set of nodes are returned as long as the set meets desired constraints (e.g., all nodes have CPU load less than 2.0). Thus, rather than specify a concrete set of interest, the bidding language should allow substitutes to be expressed to maximize the probability of a match.
- **Range Constraints**: In a time-shared system, users will rarely (if at all) be able to acquire all the resources on a particular node. At the same time, users will often have minimal requirements on resource allocations in order to make effective use of a node. Thus, users might wish to specify a range of acceptable allocations with the amount paid on a match scaled accordingly.
- **Set of Sets Allocations**: In a wide-area system, users will often wish to combine complementaries and substitutes on set of nodes where node subsets are taken from specific (usually disjoint) subsets of other nodes. For example, a user might want 32 nodes or nothing (complementaries) and only if each of the 32 nodes is in a distinct autonomous system. Here, the user would be asking for a set of 32 sets, each which is a singleton set.

Formally, we define a set of nodes $\mathbb{N} = \{1, \ldots, M\}$, a set of users $\mathbb{A} = \{1, \ldots, N\}$, a set of time periods, $\mathbb{T} = \{1, \ldots, \}$, and a set of feasible durations $\mathbb{D}$ (to define the number of consecutive periods for which a bid can be stated). For instance, the durations might allow $\mathbb{D} = \{1, 2, 4, 8, 16\}$, with $\mathbb{T}$ measured in 4 hour intervals to allow scheduling for up to 64 hours. We consider two basic models for bids on the buy-side of the exchange. Model A allows a bid to specify unequal desired resource shares, in terms of CPU, memory, network and disk resources. Model B assumes that the exchange will only allocate "shares" of resources on each node, where a share (e.g. 10%) allocates the same fraction across all resource types.

Model A: Fine-grained Resource Allocation A bid $b_i = (s_1, s_2, d, Q, f_c, f_m, f_n, f_d, C, v)$ from user $i \in \mathbb{A}$ defines: a range of acceptable start times $[s_1, s_2] \subseteq \mathbb{T}$, with s_1 no earlier than the current time; a duration $d \in \mathbb{D}$; a desired (minimal) quantity $Q \geq 1$ of nodes; (f_c, f_m, f_n, f_d) to represent the fraction of CPU, memory, network, and disk resources (respectively) required on each allocated node; a set of acceptable nodes $C \subseteq \mathbb{N}$; and a maximal price that the user is willing to pay, v (measured in the virtual currency).

Model B. Uniform Share A bid $b_i = (s_1, s_2, d, Q, f_s, C, v)$ from user $i \in \mathbb{A}$ is as in Model A, except that shares on nodes are defined in terms of uniform shares of CPU, memory, network, and disk resources, and f_s represents the number of shares that is required on each allocated node.

As an example, Model A allows a bidder to state "I want 40 nodes for 10 hours, starting over the next 6 hours, with 40% CPU, 80% memory, 20% network, and 10% disk resources, across this set of nodes that I consider interesting, and it is worth at most \$0.50 gold (the virtual currency)." In comparison, Model B allows a bidder to state "I want 40 nodes for 10 hours, starting over the next 6 hours, with a 40% share of all node resources (CPU, memory, network, disk), across this set of nodes that I consider interesting, and it is worth at most \$0.50 gold." Multiple bids can be submitted by a user in both cases, and each can be independently accepted.

In both cases, we can be smart about how a user can express her set of acceptable nodes, C, within a bid. As described above, a user actually states constraints in terms of meaningful features such as the CPU load or inter-node latencies, from which the set C is automatically populated during resource discovery.

Both models can be augmented to allow the for "set of sets" constraint on acceptable allocations. Specifically, we allow:

Model B1. Uniform Share with Set-of-Sets. A bid is extended, and defined as $b_i = (s_1, s_2, d, Q, Q_{\text{set}}, f_s, \{C^1, C^2, \ldots, C^l\}, v)$ from user $i \in \mathbb{A}$. Sets $\{C^1, C^2, \ldots, C^L\}$ are a partition of the set of nodes acceptable to the user, such that each $C^l \subseteq \mathbb{N}$ and $C^l \cap C^{l'} = \emptyset$ for all sets l, l'. Q, as before, is the total number of nodes required. Q_{set}, which must be a *divisor* of Q, specifies the number of different sets that must be active for the bid to be satisfied, with Q/Q_{set} nodes allocated from within each activated set.

We can also make the exact same extension to Model A, to allow for Model A1. Model B1 allows a bidder to state "I want 40 nodes for 10 hours, starting over the next 6 hours, with a 40% share of all node resources (CPU, memory, network, disk), and with the nodes allocated across 4 of the following 8 sets of nodes, where each set contains nodes in a distinct autonomous system, and it is worth at most \$0.50 gold." In this case, any allocation of 10 nodes from 4 different sets, that also satisfy the other requirements, would be acceptable.

Sellers, unlike buyers, already have specific shares of resources on nodes. This asymmetry suggests that the bidding language on the sell-side should be different from that on the buy-side. We choose to describe a variation on Model B for the sell-side, but can also easily support a variation on Model A (with fine-grained constraints on node resources) within our framework.

Seller Model. Uniform Share A bid $b_i = (s, e, Q_l, Q_h, f_{\text{low}}, f_{\text{high}}, C)$ from user $i \in \mathbb{A}$ (now acting as a seller) defines: a start time $s \in \mathbb{T}$, an end time $e \in \mathbb{T}$, with s no earlier than the current time and $e \geq s$; a range $[Q_l, Q_h]$ on the number of nodes on which the seller is prepared to sell (the seller must hold appropriate resources on at least Q_l nodes for this bid to be valid); a range $[f_{\text{low}}, f_{\text{high}}]$ to define the share of resources that the seller is prepared to sell on each of between Q_l and Q_h nodes; and $C \subseteq \mathbb{N}$, the set of acceptable nodes to consider within the bid.

Notice that (for the moment) we choose *not* to allow a seller to define a value for a set of nodes. Instead, the seller is limited to making a subset of resources, on a subset of nodes, available within the exchange. The seller will receive

payment for the nodes, as determined by the bids made by buyers. We view this as a useful simplification, to keep the decision problem facing sellers (of how to structure bids) as simple as possible. Also, note that we do allow a seller to submit multiple bids for disjoint sets of node resources. As for buyers, these are handled independently within the exchange.

3.2 Virtual Currency

Bids in SHARE express valuations in units of a virtual currency which is used throughout the system. In SHARE, each site has a bank account that stores virtual currency for that particular site. A SHARE bank maintains account balances for all sites and handles transactions to clear matches on the exchange. Each site's account is debited as authorized users (e.g., users at that site) buy resources on the exchange and is credited as site resources (i.e., futures on the site's machines) are sold on the exchange. We assume the existence of an authentication infrastructure that allows users interacting with the exchange (and the bank) to authenticate themselves and, additionally, for the system to identify which bank accounts a user is authorized to use (e.g., by default, perhaps all users at a site are allowed to share a site's funds). In PlanetLab [32], for example, PlanetLab central [2] maintains a central database containing a variety of information, including user authentication information for over 800 users at 155 participating sites around the world. Simple replication and use of this database would be one way to implement the authentication component of SHARE.

3.3 Mechanisms

We propose a combinatorial exchange to clear resources within SHARE. The bidding language has already been described. The goods in the market are described in terms of node identities, $\mathbb{N}$, time periods, $\mathbb{T}$, and fractions of each resource type (CPU, memory, network, disk space). In Model B— with bids limited to equal shares of each resource type —the good space can be collapsed to focus exclusively on the fraction of resources on each node allocated to a user. Thus, goods are defined for all time periods into the future, and bids can be submitted for periods through some future time window (e.g. 2 months forward). Periodically new bids (both on the buy-side and the sell-side) are collated, and the exchange is cleared. The outcome of this is: a) a new allocation of property rights; and b) payments, made immediately between users. A user winning resources in one period for some future time can make an offer to sell those resources back in a later period.

Concretely, we model the SHARE resource exchange as follows. Periodically,

1. Receive new bids (from buyers and sellers). Perform resource discovery to construct feasible node-sets, and validate the bids from sellers correspond to resources actually owned by a seller.

2. Clear the exchange to maximize total reported value (i.e. adopting allocative-efficiency as the goal).

3. Use the *Threshold rule* [31] to determine payments.

4. Report (and enforce) any changes in allocation and payments.

[2] `http://www.planet-lab.org`

The Threshold rule computes payments that are a simple variation on the payments in the Vickrey-Clarke-Groves (VCG) [41, 7, 15] mechanism. The VCG mechanism is interesting because it is non-manipulable (users can do no better than bidding truthfully). However, the VCG mechanism is not useful in our setting because it is not budget-balanced. Rather, it requires an infusion of funds by the market maker. The Threshold rule circumvents this problem by allocating payments to *minimize the worst-case deviation, across all users, from the VCG payments.* Recent analysis due to Krych [21] suggests that the Threshold rule is effective in mitigating opportunities for manipulation and providing good efficiency in two-sided and combinatorial markets.

Specifically, let V_N denote the total (reported value) from the efficient trade given all bids, and let V_{N-i} denote the total (reported value) from the efficient trade with all bids from user i removed. In the VCG mechanism, user i's final payment is determined as its reported value for the trade (i.e. v in the bid from a buyer, or 0 from a seller), *minus* her "VCG discount", $\Delta_{\text{vcg},i} = V_N - V_{N-i}$. Thus, a buyer pays less than her bid price and a seller receives payment equal to $\Delta_{\text{vcg},i}$ from the exchange. The Threshold rule modifies this, determining discounts $\Delta = (\Delta_1, \ldots, \Delta_N)$, to minimize $\max_i \Delta_{\text{vcg},i} - \Delta_i$, and subject to the (budget-balance) constraint $\sum_i \Delta_i \leq V_N$. Once VCG discounts are computed, Threshold discounts can be computed in linear time [31].

We also augment the periodic clearing of an exchange with *buy-it-now* prices. These prices are designed to provide users with the option of paying market price for resources (when such prices can estimated) to avoid the uncertainty of awaiting the outcome of the auction. Buy-it-now pricing works as follows (again, this is described here only for Model B with uniform resource shares):

1. Maintain historical information on the exponential-weighted average clearing price for each node $j \in \mathbb{N}$, each period $t \in \mathbb{T}$, per 1% share of the resources on a node. The average clearing price for a given *node-share-period* is computed by determining all bids that were allocated shares in that specific node-period, computing the average per-share price for each bid, and computing a weighted average over these per-share prices.

2. Sellers place "standing offers" for node resources that they are willing to sell at these buy-it-now prices. These standing offers can be revoked by a seller at any time. The standing offers are expressed in *exactly* the same language as the bidding-language for sellers in the exchange.

3. In between the periodic clearing of the exchange, bids can be placed to buy resources at the buy-it-now prices. Bids are accepted in FIFO order, and cleared to find the best combination of resources to buy (given buy-it-now prices and seller constraints), given the requirements in the bid. Bids are expressed in *exactly* the same language as the bidding language for buyers in the exchange. Bids are accepted if the lowest-price, feasible, combination of resources is priced below the value stated within a bid. At this point the resources change hands and the exchange collects payment on behalf of the seller(s) involved.

3.4 Policies

Currency distribution policies in SHARE serve an analogous purpose as macroeconomic policy in government. In systems that operate as closed economies, these policies are especially important for a practical SHARE deployment. For example, consider a research testbed such as PlanetLab. In PlanetLab, sites typically contribute comparable amounts of resources (usually 2-3 nodes). However, a recent measurement study [6] observes that per-site resource consumption varies by several orders of magnitude and that a small number of sites are responsible for the majority of global resource consumption. What should a resource allocation system do about this? Clearly, we would probably not want to disallow these sites from consuming resources and doing good research when global resources are not heavily contended for. Related to this, we probably also do not want to have idle sites accumulating massive amounts of currency which could later lead to monopolization of resources or severe underutilization of the system.

A SHARE currency distribution policy consists of mechanisms and rules which are intended to produce some desired global behavior. Such mechanisms and rules are directly analogous to those employed by governments to implement macroeconomic policy, although the situations we aim to remedy here are quite a bit different in some cases. For example, taxation is a mechanism to raise capital. Such capital might then be redistributed according to rules that aim to implement a certain policy. In PlanetLab, for instance, taxing revenue earned on the exchange and redistributing this revenue is one way to increase utilization and also to create certain types of incentives. For example, a simple rule might be that all sites are taxed 50% of their earnings and that revenue is redistributed in proportion to the number of nodes and services each site contributes to the system. Another might be that unpopular sites (e.g., perhaps they have long hostnames) that nevertheless contribute to the community (e.g., suppose they provide a global monitoring service) get subsidies.

The space of mechanisms and rules to implement policy will essentially revolve around fees and taxes. Taxation on revenue as a result of trade was one possibility. Fixed or variable fees on each trade for both buyers and sellers is another as is periodic membership fees for being part of the system. There are a number of mechanisms one could employ and infinitely many rules for redistributing currency amongst the various participants in the system. Practically speaking, however, a currency policy in SHARE should do no more than is needed to produce the desired behavior in the system.

4. PlanetLab Resource Exchange

In this Section, we present the design of a SHARE system targeted at distributed resource allocation on PlanetLab. PlanetLab is a wide-area network testbed that currently comprises 384 nodes at 161 sites spread across over 30 countries around the world. It has over 800 active users engaged in a wide variety of research projects in areas such as distributed storage, routing overlays, network measurement, distributed query processing, and global intrusion detection. As PlanetLab aims to enable widely distributed applications with wide geographic coverage, users frequently seek access to large amounts of global resources in order to perform experiments at scale. As illustrated in Figure 1, such resource access often results in contention for scarce resources which, in turn, leads to thrashing of system resources and difficultly in obtaining meaningful performance measurements. The goal of this SHARE instance would be to mitigate such contention by scheduling resource requests more intelligently in space and time based on the value that users place on the resources. Such resource valuations would result in payments using a virtual currency with currency being debited against the corresponding site's bank account and currency being credited based on the payment rules of the exchange in combination with a currency distribution policy. In this Section, we present algorithms for clearing a combinatorial resource exchange, currency distribution policies that determine how currency is distributed amongst sites in the federation, and finally how resources allocated by the exchange would be claimed and ultimately utilized by an application.

4.1 Algorithms

In Section 3.3, we proposed using a combinatorial exchange to match resource requests with available resources. Such an exchange requires an algorithm to perform this matching in both a computationally efficient and economically efficient manner. That is, the algorithm should clear the exchange in a timely manner relative to the time granularity of the resources being allocated. Further, it should also compute resource allocations that lead to high economic efficiency (e.g., consider the computationally efficient algorithm that simply performs no matches and returns.). Here, we present two approaches to clearing the exchange in SHARE: mathematical formulations based on mixed-integer programming (MIP) and greedy algorithms.

4.1.1 MIP-based Formulations

The MIP is described in more detail in the Appendix. For now we describe the decision variables and explain the structure of the formulation. The main feature of the formulation is that we do not explicitly formulate a decision variable $x_i(j,k)$ for whether node j is allocated to user i in period k. Rather, we work at the level of *blocks* of nodes. For instance, if a bid is for Q nodes for 4 consecutive periods, we use decision variables to capture whether Q nodes are allocated *for 4 consecutive periods*, starting from a particular time. The implication of this allocation (i.e. that resources are used in the first period and also the next 3 periods) is captured through careful "resource balance" feasibility constraints.

The main decision variables are: $x_i \in \{0,1\}$, to denote whether or not bid from user i is successful; $4b(i,k) \in \{0,1\}$ to denote whether or not a bid from user i (in this case for duration of 4 time periods) is allocated to start in period k, and similarly for $8b(i,k), 2b(i,k)$, etc., for all $d \in \mathbb{D}$; and $4n(i,j,k) \in \{0,1\}$ to denote whether or not bid from user i (in this case for duration of 4 time periods) is allocated to start in period k on node j, and similarly for $8b(i,j,k), 2b(i,j,k)$, etc., for all $d \in \mathbb{D}$. Feasibility requires that $x_i = 1 \Rightarrow 4b(i,k) = 1$ for one period k in the range of acceptable start periods in user i's bid (assuming the bid is for duration 4). Feasibility also requires that $4b(i,k) = 1 \Rightarrow 4n(i,j,k) = 1$ on Q nodes, where the bid was for a total of Q nodes. Finally, $4n(i,j,k) = 1$ on node j starting in period k (for 4 periods) implies that we must check that there are resources available (CPU, memory, disk, network) to allocate to this user on this node during these periods. Taking each resource type in turn, e.g. CPU, this is achieved by writing

a single feasibility constraint for every node j in every period k, that ensures that the total amount of CPU resources implied by the new allocation is no greater than the total CPU resources available on the node. Bids from sellers are introduced at this level, with seller bids also able to contribute the supply of resources to help with this resource-balance feasibility constraint.

The beauty of a MIP approach is that there are well-established commercial solvers (that have anytime performance), and also that the approach is highly flexible. With little effort we were able to capture all the variations that we found interesting, such as buyers *and* sellers, and such as the "set of sets" representation.

4.1.2 Greedy Algorithms

Clearing a combinatorial resource exchange is a computationally difficult problem that is at least as hard as weighted set packing, a problem known to be NP-complete. While commercial solvers such as CPLEX routinely solve instances of similar combinatorial problems with either optimal or high-quality solutions, it is nonetheless an interesting question to see how well heuristics perform in practice given realistic resource allocation workloads. Thus, as a point of comparison, we have developed two simple and fast greedy algorithms for clearing the exchange in SHARE (either Model A or Model B). While these algorithms are not guaranteed to provide optimal solutions, they do offer the advantage of running in a computationally efficient manner. This, in turn, allows the exchange to be cleared very quickly which subsequently opens up other possibilities (e.g., clearing exchange at a finer time granularity).

The *Greedy* algorithm performs as follows. First, it calculates a bid value bid_i for each bid, $bid_i = v_i/Q_i$. This bid value gives us a *per-unit* valuation for each agent. Second, we sort all bids in descending order according to bid_i. That is, we are interested in how valuable each unit of resource is to each agent with a preference for bids with higher per-unit valuations. Notice that if we do not create bid_i but instead use v_i directly, we would consider an agent that has, for example, $v_i = 10, Q_i = 100$ to have a more valuable bid that another agent that has $v_i = 9, 1$ (i.e,. a \$10 bid for 100 nodes versus a \$9 bid for a single node). Third, we go through the sorted bid list and evaluate each single bid. If the resource request for b_i can be fulfilled with the remaining node resources, we will allocate the resource to i. Specifically, we start by searching into the time slots s_1, s_2 that b_i specifies. We seek a specific time duration d_i such that we could find at least Q_i nodes that have resources available for what b_i asks for. The specific criteria to check such availability depends on whether we use Model A or B. The greedy algorithm stops after scanning the entire bid list and trying to match each bid.

Notice that unlike the MIP-based approach, the greedy algorithm is fast simply because it evaluates a very narrow set of possible outcomes. The order of the bids determine the scope of this set, and each time a bid is allocated the set of remaining possible outcomes become even smaller.

The second greedy algorithm we consider is *Greedy Refined*. It works the same as the above, except for the computation of the bid value. Here, we scale each bid's value inversely based on the aggregate number of resource shares requested by the bid (i.e., nodes x duration x shares) as follows. For model B, which allocates entire shares of nodes, we have $b_i = v_i/(Q_i * d_i * f_s)$. The "unit" in this case is a single time slot, for a single share, on a single node. Similarly, for model A, which allocates shares of individual resources (CPU, memory, network, and disk), we have $b_i = v_i/Q_i * d_i * f')$, where $f' = f_c + f_m + f_n + f_d$. The "unit" is same as above, except that we aggregate shares for individual resources in our calculation. Further details of both the *Greedy* and *Greedy Refined* algorithms are provided in Appendix C.

4.2 Currency Distribution Policy

Using a combinatorial exchange for resource allocation involves bids which specify value in units of virtual currency, and such virtual currency requires a currency distribution policy both for bootstrapping the system and for achieving specific global behavior. Based on a 3-month measurement study of resource usage on the PlanetLab infrastructure, we propose a currency distribution policy that targets two key issues that are common on PlanetLab: (i) widely varying per-site resource consumption and (ii) sites that are idle for long periods of time. To address these issues, we propose a currency policy based on two ideas: caps on the currency that can be accumulated in each account and a community pool of virtual currency. Underlying both of these ideas is a baseline amount of currency that each PlanetLab site is assigned when it joins the system. (Differences in this baseline can be made after the fact.) This number would be a reflection of a site's contributions to the system and would be assigned by PlanetLab central based on a well-known formula based on the number of nodes the site contributes and the number of active services the site provides (e.g., monitoring, storage, routing, measurement, caching). Discussions of such a formula and associated incentives have already been topics of discussion in the PlanetLab community and thus are not unreasonable.

Given baseline currency for each site, each site has a cap imposed on its bank account. Sites can use credit at whatever rate they wish but they can accumulate credit only up to some maximum amount (e.g., perhaps a small multiple of its baseline). The intent of the cap is to reward sites that contribute more than they consume but to limit the impact of a site accumulating substantial amounts of credit and bursting. This idea is similar to the credit policy scheme used in Stoica's microeconomic parallel scheduler [39]. Credit in accounts would be accumulated as usual through revenue obtained by selling on the exchange. Once a site reaches its cap, all additional currency then would get automatically donated to a community pool. The goal of the community pool would be to fund sites that are consuming resources when resources are not scarce. One initial currency distribution rule might then be to distribute currency in a proportional-share fashion based on baseline values for all sites that need it (e.g., all sites that have current balances below their baseline).

More specifically, let K be the amount of virtual currency in the community pool to be distributed and let $C_1, C_2, \ldots, C_n$, $B_1, B_2, \ldots, B_n$, and $M_1, M_2, \ldots, M_n$ denote the current, baseline, and maximum virtual currency for each site, respectively. The new balance for each site that meets the requirements for distribution (i.e., $C_i < B_i$) from the community pool would then be computed as:

$$C_i = max(C_i + (B_i/(\sum_{\forall j | C_j \neq M_j} B_j))K, M_i) \quad (1)$$

To distribute the currency, the above would be iteratively applied to all site accounts until all funds in the community pool have been exhausted.

4.3 Trading and Claiming

Once resource requests have been matched with resources on the exchange, resources then need to be physically instantiated on specific machines in the federation and bound to the user's application. This is effectively the execution of the outcome as determined by the exchange. Given bids that specify resource requests and an exchange with available resources, the exchange determines the winning bids along with the specific nodes allocated, the specific times resources will be allocated on those nodes, and the specific numbers of resource shares that will be allocated. Once such a determination has been made, the outcome of this process needs to be physically instantiated, in this case by allocating computational resources on particular PlanetLab nodes.

In PlanetLab, a fundamental abstraction is a slice. A slice is network of virtual machines [44, 38, 2] spanning some set of physical machines with per-node resources bound to each virtual machine. Typically, a user creates a different slice for each distributed application the user wishes to run on the PlanetLab infrastructure. In the context of SHARE, we assume that creation and provisioning of slices is done separately from resource allocation via SHARE. A number of existing and emerging services on PlanetLab already provide such functionality and are currently in use today. Examples include Emulab [45], PlanetLab Central (PLC), and Stork.

We envision that each PlanetLab node will run a SHARE daemon that is responsible for binding and releasing of hard resource allocations associated with specific slices. Such daemons would run in a privileged slice that is able to directly request hard resource allocations using proportional-share CPU, memory, and network schedulers provided by the underlying PlanetLab operating system [3]. One possible approach for coordinating the actions of these daemons might then be to have the daemons have a trust relationship with the SHARE exchange such that cryptographically signed requests from the exchange are always honored (e.g., bind 30 shares of CPU to slice `ucsd_bullet`). Such an approach is simple to implement and, indeed, is a frequently employed mechanism in PlanetLab's current operational infrastructure.

Given the above, the model we thus envision is that users will create virtual machines on nodes in PlanetLab in one of two ways. One way is that a user creates a "best effort" virtual machine when idle resources are available as they currently do today. Given such virtual machines, a user who has a winning bid on the exchange would then obtain cryptographically tickets for hard resource guarantees from the exchange and bind those resources to existing virtual machines by performing multiple secure RPCs in parallel to relevant nodes. This mode of operation would essentially be analogous to that used in the SHARP [14] and `dslice` systems, both of which also were deployed on PlanetLab at various times. A second mode of operation might be that the system requires that the user present resource tickets when creating a new virtual machine. In this case, hard resources are bound to virtual machines as they are created and having such hard allocations is a requirement for slice creation. Such a scenario might be used when "best effort" resources are scarce and the system is performing admission control for example.

5. Microbenchmarks

In this section, we present microbenchmarks that quantify the performance of SHARE using different algorithms for clearing the auction. We simulate a system consisting of 1024 nodes and focus on resource allocation over a 16-day period with resource requests being either 1-day, 2-day, 4-day, 8-day, or 16-day requests for shares on subsets of machines satisfying different constraints. For each algorithm, we examine its performance as a function of resource demand presented to the system. We choose 1024 nodes because it is approximately of the same order of magnitude of current large-scale computation and communication infrastructures and because it stresses the ability to clear auctions using currently available hardware.

5.1 Experimental Setup

Our experimental setup is shown in Figure 4. Each experiment is driven by a synthetic workload which was designed to allow us to stress each algorithm as we scaled up the resource demand being placed on the system (i.e., the number of bids). Resource requests in the workload specify distributed resources in space and time and are passed to a meta-bidding agent, which simulates all users in the system. Each resource request is constructed as follows:

- Number of nodes requested taken from the following distribution: 128 nodes (20%), 256 nodes (30%), 512 nodes (30%), 1024 nodes (20%).
- Request duration (number of days) taken from the following distribution: 1 day (10%), 2 days (20%), 4 days (50%), 8 days (20%), 16 days (10%).
- Minimum start time was chosen uniformly at random from $[0, 16 - duration]$. Maximum end time chosen either uniformly at random from $[minstart + duration, 16]$ or set to 16.
- Minimum resources requested (in shares) chosen from a Zipf distribution.
- Node constraints chosen uniformly at random from single attribute constraints on CPU speed, CPU load (1, 5, and 15 minute load averages), free disk space, total disk space, free memory, and total memory. These constraints determined the set of feasible nodes for the given request (e.g., "64 nodes with each node having at least 1.0 GB of physical memory").
- Value for resources is based on aggregate amount of resources being requested. More specifically, the value (expressed in units of virtual currency) of a resource request is the product of number of nodes, duration, and number of shares. The final product is normalized by a factor of 1/10000 for convenience.

For each request, the meta-bidding agent constructs a bid by mapping an abstract resource specification (`rspec`) contained in the request down to a concrete set of resources (`rset`) that the user is interested in. This is accomplished by automatically formulating a resource discovery query based on the `rspec` and performing a resource discovery query using a resource discovery system based on PlanetLab data. The resource discovery system uses both static and dynamic

data collected from the PlanetLab testbed extrapolated to up to 1024 nodes. Static data is obtained from PlanetLab central [3] while dynamic data is obtained from an instance of Ganglia [4] which monitors all PlanetLab nodes.

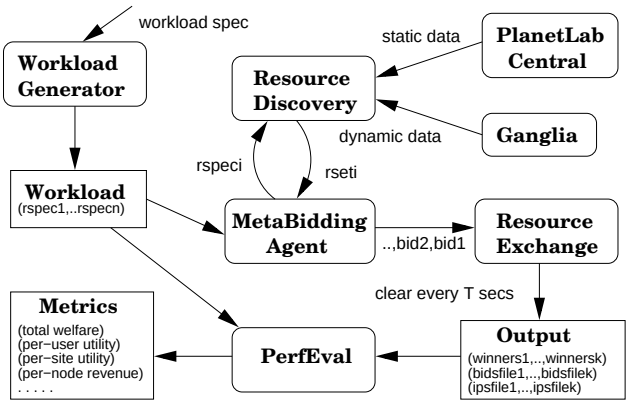

Figure 4: Experimental Setup.

Given concrete resources (`rset`) from the resource discovery system, the meta-bidding agent then constructs a bid (in the bidding language from Section 3.1) and passes that bid to the SHARE auction. In a live implementation, such bids would then get matched during periodic clearing of the auction. For purposes of this evaluation, we simply collect a desired number of bids (based on how much demand we want imposed on the system) and immediately clear the auction and examine the results. The results from clearing the auction include (i) a set of winning bids and (ii) the concrete nodes, times, and shares allocated to those bids.

5.2 Microbenchmarks

We evaluated SHARE using three different algorithms: (i) a MIP-based formulation using CPLEX, a commercial, linear-programming based, branch-and-cut solver for mixed integer programs (Section 4.1.1), (ii) a greedy algorithm (Section 4.1.2), and (iii) a refined greedy algorithm (Section 4.1.2). We refer to these algorithms as CPLEX MIP, Greedy 1, and Greedy 2. Each algorithm was evaluated given the workload just described and performance was measured as total revenue collected by the auction. Assuming each user's declared bid value reflects the user's true utility for associated resources, then revenue collected by the auction represents aggregate utility. All algorithms were evaluated on an 8-way multiprocessor PC with 2 GHz CPUs, 3.5 GB of physical memory, running Linux 2.4.20. For CPLEX experiments, we used CPLEX version 8.110 [5] and imposed an upper-bound of three hours for CPLEX to clear the auction. We also added performance optimizations to CPLEX by specifying an algorithmic emphasis on feasibility (as opposed to optimality) and by having CPLEX generate Gomory cuts more aggressively (cut aggressiveness was set to 2) when executing the branch-and-cut algorithm.

At this point we are mainly concerned with the complexity of clearing the exchange, which is the key to validate our design. We defer Vickrey/Threshold pricing and buy-it-now

[3] `http://www.planet-lab.org`
[4] `http://ganglia.sourceforge.net`
[5] Note that all CPLEX experiments utilized only a single CPU since we did not have a license for CPLEX's parallelization capabilities.

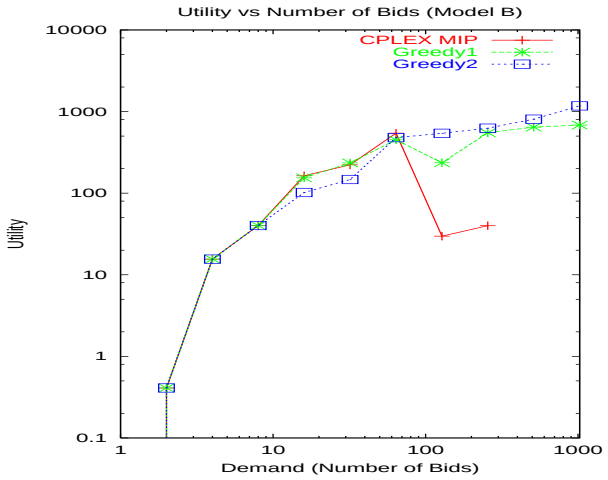

Figure 5: Utility as a Function of Demand.

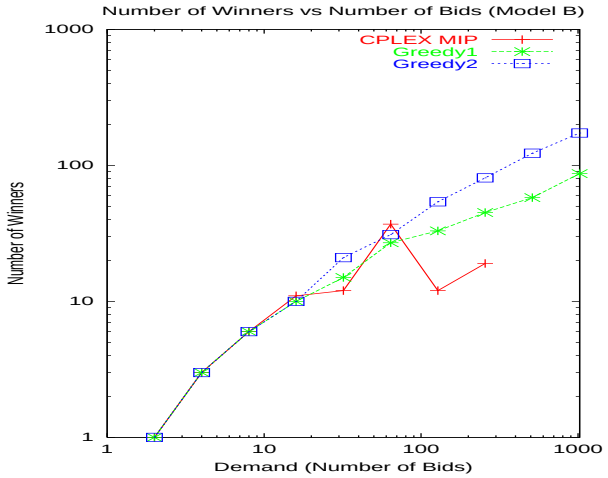

Figure 6: Accepted Requests as a Function of Demand.

prices for future studies. The results for Model B are shown in Figures 5 and 6. Our raw results from solving the CPLEX MIP indicated that an *optimal* solution was obtained for resource demands consisting of 1 bid up to 64 bids (recall, though, that each bid can include up to 1024 nodes and up to 16 days). From the graph, we see that the Greedy 1 algorithm also performed optimally over this range of demands, while Greedy 2 performed only slightly worse up to 64 bids. At 64 bids, we observe an inflection point for the CPLEX MIP approach. This drop is due to CPLEX's inability to find a high-quality solution given the time constraints that are currently imposed. The greedy algorithms, on the other hand, always clear the auction in less than a few minutes.

We expect to be able to achieve an order-of-magnitude improvement in the performance of CPLEX through a tighter reformulation of the MIP in which continuous variables, such as $4nCPU(i,j,k)$ (see the Appendix) are dropped completely. Rather, the idea that an allocated node is useful to satisfy a bid only when the associated amount of node resources are also provided can be directly handled by encoding, for instance, $f_c \cdot 4n(i,j,k)$ within the feasibility con-

straints. We note that the current formulation requires a lot of effort in presolve to reduce the MIP tableaux before entering search, and that the integrality gap is also high (as high as 500%) during search. On the other hand, it is interesting to observe that both greedy algorithms perform surprisingly well. This is particularly interesting given the flexibility on the resource allocation constraints in space and time afforded by the bidding language for Model B. Given an improved MIP-based solution, it will be interesting to compare the performance of these greedy algorithms for more demanding workloads against an optimal solution by pushing CPLEX's scalability out further.

6. Related Work

6.1 Combinatorial auctions and heuristics

Recently researchers have explored the computational tractability of combinatorial auctions. In general the problem is known to be NP-hard, although fast algorithms (such as CPLEX) have proved useful in practice. For provable polynomial-time performance, Lehmann et. al. [22] proposed a greedy heuristic for the special-case of single-minded bidders that are interested in a specific bundle of resources. The mechanism is strategyproof (with a dominant strategy is to bid truthfully), but does not extend to the complex minded bidders that we need to model within SHARE. It is also for a one-sided auction, while our interest is in an exchange.

Zurel and Nisan [47] provide a novel linear-programming (LP) based heuristic algorithm (ALPH) to approximately solve very large combinatorial winner-determination (WD) problems. The first phase of the algorithm computes an approximate solution to an LP relaxation of the WD problem. The second phase then runs a sequence of greedy hill-climbing algorithms to improve the initial solution. The goal of the heuristic is to find allocations that are close to optimal. Although fast and close to optimal, it is not clear how to extend their approach to the expressive and compact bidding language that we adopt in SHARE. The language in ALPH is a flat XOR representation, which is not practical for SHARE.

Two-sided combinatorial auctions, or combinatorial exchanges [30], present new difficulties for both economic and computational reasons. First, simple VCG-based methods that provide truthfulness for one-sided auctions do not extend. Second, the winner-determination problem is more difficult because feasibility is harder to achieve (notice that a feasible solution is easy to find with a greedy heuristic in a one-sided auction because a set of bids can be greedily adopted, but that this does not provide a feasible set of trades in a two-sided market) [36].

6.2 Mechanism Design

Algorithmic Mechanism design [29] aims to design systems to implement outcomes with desirable system-wide properties, such as efficient resource allocation or revenue maximization. The theory of mechanism design assumes individual rationality and self-interested agents, and a central concern is to provide incentives so that agents will reveal truthful information about their values for different outcomes (via bids), in equilibrium. In the special case in which truthful bidding is a dominant strategy equilibrium this serves to simplify the strategic problem facing users, and frees them from performing costly counterspeculation.

Several new types of mechanisms could potentially be used by SHARE in the future. Online mechanisms [17] assume dynamic agents arrivals and departures (just like slice requests in PlanetLab). Such a mechanism does not "open and close" like a traditional auction does. Instead, it makes allocation decisions for each agent as it arrives and before its departure, until all goods are sold. The payment rules of the mechanism are such that they are independent of an agent's own valuation, and hence an agent's dominant strategy is to submit bids as soon as the need arises.

Another direction to apply mechanism design for SHARE is to decentralize the allocation process and let the individual nodes (or sites of nodes) to run their own mechanisms. In this case, we have to be concerned with the guarantees that users are expecting (i.e. getting nodes from one site but not another). There have been proposals to building such decentralized mechanism-based open systems [27], which provides the attractive notion that more than one mechanism can be deployed in the system. Nodes can select mechanisms with the right properties for their own goals (e.g. revenue, fairness). The new challenge is in verifying the properties of mechanisms, and keeping only those mechanisms that are strategyproof.

6.3 Pricing

Economists have long proposed pricing as a way to decentralize resource allocation. If done correctly, pricing provides a number of benefits [16]: decision making is decentralized; pricing can change user behavior (for example shifting demand); and pricing can make systems more transparent. Varian and Mackie-Mason [24] proposed one of the first usage-based pricing model for the Internet.

Kelly et. al. [20] consider the problem of allocating bandwidth within a network amongst competing traffic. They present mathematical models to analyze the stability and fairness of a class of rate-control algorithms. Although the allocation problem is much simpler than that considered in SHARE it is conceivable that some of the concepts from congestion pricing can be adopted within the buy-it-now pricing component of SHARE. Mungi [18] is another project that uses price functions to provide congestion pricing. The main challenge in adopting a pricing-approach for SHARE is that it is combinations of resources that matter, but it would not be practical to price all combinations. By comparison, auctions allow for expressive bids and use optimization methods to determine allocations of goods.

6.4 Market-based systems

Much work has advocated the use of market methods for computational systems [8], but we are aware of few fielded market-based systems.

Spawn [42] was one of the pioneer papers that used markets to support distributed allocation. A number of workstations each sell CPU time and each runs sealed-bid second price auctions. Applications are funded by respective users' funding rates. Spawn only address CPU and does not provide the guarantees that SHARE does. In Popcorn [34], applications are decomposed into small Java programs that can then be run on servers in a distributed system. Buyers submit requests along with their values to be matched with sellers in a centralized market. The bidding language in SHARE is much more expressive than that in Popcorn, and SHARE is focused on the allocation of bundles of networked resources while Popcorn can split up a problem and allocate a sequence of compute-server resources across time.

7. Conclusion

Resource allocation has long been fundamental to the design of computer systems. Over the years a number of efforts have explored applying economic models to allocating system resources, given the seemingly good match between the problem domain and well-studied solutions in economics. However, these efforts have met with mostly limited success for a variety of reasons. We contend that the most important of these reasons is that target resources were typically under the direct control of a single user or organization, making relatively simple solutions feasible. However, emerging federated global-scale computation environments are of sufficient scale and shared by sufficient users and organizations with disparate and conflicting policies that existing solutions are proving intractable. In this paper, we explore the feasibility of a centralized auction for allocating global resources to competing users based on some policy for distributing virtual "currency" to competing organizations. We envision this work as being a first step toward a more flexible computational exchange where organizations sell their available resources in exchange for currency that can later be used to purchase desired global resources in a stable and efficient macroeconomic computing environment.

8. REFERENCES

[1] Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles* (October 2001).

[2] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (October 2003).

[3] Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., and Wawrzoniak, M. Operating systems support for planetary-scale network services. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation* (March 2004).

[4] Boutilier, C., and Hoos, H. H. Bidding languages for combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)* (2001), pp. 1211–1217.

[5] Chun, B., Fu, Y., and Vahdat, A. Bootstrapping a distributed computational economy with peer-to-peer bartering. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems* (June 2003).

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

[7] Clarke, E. H. Multipart pricing of public goods. *Public Choice 11* (1971), 17–33.

[8] Clearwater, S. H., Ed. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[9] Cocchi, R., Shenker, S., Estrin, D., and Zhang, L. Pricing in computer networks: Motivation, formulation, and example. *IEEE Transactions on Networking* (1993).

[10] Cooper, B., and Garcia-Molina, H. Peer-to-peer resource trading in a reliable distributed system. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems* (March 2002).

[11] de Vries, S., and Vohra, R. V. Combinatorial auctions: A survey, 2000.

[12] Foster, I., and Kesselman, C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications 11*, 2 (1997), 115–128.

[13] Foster, I., Kesselman, C., and Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications 15*, 3 (2001).

[14] Fu, Y., Chase, J., Chun, B., Schwab, S., and Vahdat, A. SHARP: An architecture for secure resource peering. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles* (October 2003).

[15] Groves, T. Incentives in teams. *Econometrica 41* (July 1973), 4.

[16] Gupta, A., Stahl, D. O., and Whinston, A. B. The economics of network management. *Communications of the ACM 42*, 9 (1999), 57–63.

[17] Hajiaghayi, M. T., Kleinberg, R., and Parkes, D. C. Adaptive limited-supply online auctions. In *Proceedings of the 5th ACM conference on Electronic commerce* (2004), ACM Press, pp. 71–80.

[18] Heiser, G., Lam, F., and Russell, S. Resource management in the mungi single-address-space operating system. In *Proceedings of the 21st Australian Computer Science Conference* (February 1998).

[19] Jackson, D., Snell, Q., and Clemen, M. Core algorithms of the maui scheduler. In *Proceedings of the 7th Workshop on Job Scheduling Strategies for Parallel Processing* (June 2001).

[20] Kelly, F., Maulloo, A., and Tan, D. Rate control in communication networks: shadow prices, proportional fairness and stability. In *Journal of the Operational Research Society* (1998), vol. 49.

[21] Krych, D. Calculation and analysis of nash equilibria of vickrey-based payment rules for combinatorial exchanges. Undergraduate thesis, Harvard University, April 2003.

[22] Lehmann, D., O'Callaghan, L. I., and Shoham, Y. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM 49*, 5 (September 2002), 577–602.

[23] Levy, L., Blumrosen, L., and Nisan, N. On line markets for distributed object services: the majic system. In *Proc. of USITS '01* (Mar. 2001).

[24] MacKie-Mason, J. K., and Varian, H. R. Pricing congestible network resources. *IEEE Journal on Selected Areas in Communications 13*, 7 (1995), 1141–1149.

[25] Miller, M. S., Krieger, D., Hardy, N., Hibbert, C., and Tribble, E. D. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific Press, 1996, ch. 5 : An Automated Auction in ATM Network Bandwidth.

[26] Neugebauer, R., and McAuley, D. Congestion prices as feedback signals: An approach to qos management. In *Proceedings of the 9th ACM SIGOPS European Workshop* (September 2000).

[27] Ng, C., Parkes, D. C., and Seltzer, M. Strategyproof computing: Systems infrastructures for self-interested parties. In *Workshop on Economics of Peer-to-Peer Systems* (Berkeley, CA, 2003).

[28] Nisan, N. Bidding and allocation in combinatorial auctions. In *Proceedings of the 2nd ACM Conference on Electronic Commerce* (2001), pp. 1–12.

[29] Nisan, N., and Ronen, A. Algorithmic mechanism design. In *Proceedings of the 31st ACM Symposium on Theory of Computing* (May 1999).

[30] Parkes, D., Kalagnanam, J., and Eso, M. Achieving budget-balance with vickrey-based payment schemes in exchanges. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (2001), pp. 1161–1168.

[31] Parkes, D. C., Kalagnanam, J., and Eso, M. Achieving budget-balance with vickrey-based payment schemes in exchanges. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (August 2001).

[32] Peterson, L., Culler, D., Anderson, T., and Roscoe, T. A blueprint for introducing disruptive technology into the internet. In *Proceedings of the 1st Workshop on Hot Topics in Networks* (October 2002).

[33] Raman, R., Livny, M., and Solomon, M. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing* (July 1998).

[34] Regev, O., and Nisan, N. The popcorn market – an online market for computational resources. In *Proc. of ICE '98* (Oct. 1998).

[35] Sandholm, T., Suri, S., Gilpin, A., and Levine, D. Cabob: A fast optimal algorithm for combinatorial auctions. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (August 2001).

[36] Sandholm, T., Suri, S., Gilpin, A., and Levine, D. Winner determination in combinatorial auction generalizations. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems* (July 2001).

[37] Shenker, S., Clark, D., Estrin, D., and Herzog, S. Pricing in computer networks: Reshaping the research agenda. *ACM Computer Communications Review 26*, 2 (1996), 19–43.

[38] Solucorp. Whitepaper: Virtual private servers and security contexts. Available from: `http://www.solucorp.qc.ca/miscprj/s_context.hc`, September 2002.

[39] Stoica, I., Abdel-Wahab, H., and Pothen, A. *Lecture Notes in Computer Science, Vol. 949*. Springer-Verlag, 1995, ch. A Microeconomic Scheduler for Parallel Computers, pp. 200–218.

[40] Sutherland, I. E. A futures market in computer time. *Communications of the ACM 11*, 6 (1968).

[41] Vickrey, W. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance 16* (1961), 8–37.

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

[43] Waldspurger, C. A., and Weihl, W. E. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation* (November 1994).

[44] Whitaker, A., Shaw, M., and Gribble, S. D. Scale and performance in the denali isolation kernel. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation* (December 2002).

[45] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. An integrated experimental environment for distributed systems and networks. In *Proc. of OSDI '02* (Dec. 2002).

[46] Zurel, E., and Nisan, N. An efficient approximate allocation algorithm for combinatorial auctions. In *Proceedings of the 3rd ACM Conference on Electronic Commerce* (October 2001).

[47] Zurel, E., and Nisan, N. An efficient approximate allocation algorithm for combinatorial auctions. In *Proceedings of the 3rd ACM conference on Electronic Commerce* (2001), ACM Press, pp. 125–136.

A SHARE MIP Formulations

Model A

A bid $b_i = (s_1, s_2, d, Q, f_c, f_m, f_n, f_d, C, v)$ from user $i \in \mathbb{A}$ defines: a range of acceptable start times $[s_1, s_2] \subseteq \mathbb{T}$, with s_1 no earlier than the current time; a duration $d \in \mathbb{D}$; a desired (minimal) quantity $Q \geq 1$ of nodes; (f_c, f_m, f_n, f_d) to represent the fraction of CPU, memory, network, and disk resources (respectively) required on each allocated node; a set of acceptable nodes $C \subseteq \mathbb{N}$; and a maximal price that the user is willing to pay, v (measured in the virtual currency).

The MIP has the following classes of decision variables: $x_i \in \{0,1\}$ to denote whether or not bid i is accepted; $4b(i,k) \in \{0,1\}$ for all bids i with duration $d = 4$ and for all periods $s_1 \leq k \leq s_2$ to denote whether or not bid i is allocated Q nodes for a duration of 4 periods starting in period k (and similarly with variables $8b(i,k) \in \{0,1\}$ for all bids with duration $d = 8$, etc.); $4n(i,j,k) \in \{0,1\}$ to denote whether or not bid i (with duration $d = 4$) is allocated node j for 4 periods starting in period k (and similarly with variables $8n(i,j,k) \in \{0,1\}$, etc.). Finally, variables $0 \leq 4nCPU(i,j,k) \leq 100$ denote the share of CPU on node j allocated to bidder i, and similarly for $0 \leq 4nMEM(i,j,k) \leq 100$, etc, and for other durations $d \in \mathbb{D}$. The objective is

$$\max \sum_i x_i v_i \tag{2}$$

To illustrate the constraints, consider a bid for four consecutive periods. First,

$$x_i \leq \sum_{k \in \{s_1,\ldots,s_2\}} 4b(i,k) \tag{3}$$

so that $x_i = 1$ only when one of $4b(i,k) = 1$. Second,

$$\sum_{k \in \{s_1,\ldots,s_2\}} 4b(i,k) \leq 1 \tag{4}$$

so that at most one starting period is considered for each bid. Third,

$$Q \cdot 4b(i,k) \leq \sum_{j \in C} 4n(i,j,k), \forall k \in \{s_1,\ldots,s_2\} \tag{5}$$

so that $4b(i,k) = 1$ only when at least Q nodes are allocated for 4 periods, starting from period k. Fourth,

$$f_c \cdot 4n(i,j,k) \leq 4nCPU(i,j,k), \quad \forall j, \forall k \in \{s_1,\ldots,s_2\} \tag{6}$$

$$\cdots\cdots$$

$$f_d \cdot 4n(i,j,k) \leq 4nDISK(i,j,k), \quad \forall j, \forall k \in \{s_1,\ldots,s_2\} \tag{7}$$

so that $4n(i,j,k) = 1$ only when at least f_c CPU resources, f_d disk resources, etc. are allocated on node n, for 4 periods, starting from period k. Finally, we need feasibility constraints. For simplicity, we assume that the possible durations $\mathbb{D} = \{1,2,4\}$. Also, let $CPU(j,k)$ denote the CPU available at node j in period k. Then, for CPU resources, we have constraints

$$\sum_{i|d=4} \sum_{k' \in \{k-3,\ldots,k\}} 4nCPU(i,j,k') + \sum_{i|d=2} \sum_{k' \in \{k-1,k\}} 2nCPU(i,j,k') + \sum_{i|d=1} 1nCPU(i,j,k) \leq CPU(j,k) \tag{8}$$

for all nodes j and all periods k, where the only bids that are enumerated are those with time periods that span period k. Similarly for the other node resources. This constraint accounts for all activated blocks of bids that include period k.

Model B

A bid $b_i = (s_1, s_2, d, Q, f_s, C, v)$ from user $i \in \mathbb{A}$ is as in Model A, except that shares on nodes are defined in terms of uniform shares of CPU, memory, network, and disk resources, and f_s represents the number of shares that is required on each allocated node. The formulation is simpler than for Model A because the variables $4nCPU(i,j,k)$, $4nMEM(i,j,k)$, etc. are simply replaced with variables $4nSHARE(i,j,k)$, to indicate the shares on node j that are allocated to bid i for 4 periods starting in period k. Thus, constraints (6) through (7) are simplified to:

$$f_s \cdot 4n(i,j,k) \leq 4nSHARE(i,j,k), \quad \forall j, \forall k \in \{s_1,\ldots,s_2\} \tag{9}$$

and constraints (8) are simplified to:

$$\sum_{i|d=4}\sum_{k'} 4nSHARE(i,j,k') + \sum_{i|d=2}\sum_{k'} 2nSHARE(i,j,k') + \sum_{i|d=1} 1nSHARE(i,j,k) \leq SHARE(j,k) \quad (10)$$

for all nodes j and all periods k, where $SHARE(j,k)$ is the number of shares on node j in period k currently unallocated.

Model B1

A bid $b_i = (s_1, s_2, d, Q, Q_{\text{set}}, f_s, \{C^1, C^2, \ldots, C^l\}, v)$ now defines sets $\{C^1, C^2, \ldots, C^L\}$, which partition the set of nodes acceptable to the user, such that each $C^l \subseteq \mathbb{N}$ and $C^l \cap C^{l'} = \emptyset$ for all sets l, l'. Q, as before, is the total number of nodes required. Q_{set}, which must be a *divisor* of Q, specifies the number of different sets that must be active for the bid to be satisfied, with Q/Q_{set} nodes allocated from within each activated set.

Varying Model B, this reformulation introduces variables $4n(i,l,k) \in \{0,1\}$ to indicate whether or not Q/Q_{set} nodes from set C_l are selected, and replaces constraints (5) with

$$Q_{\text{set}} \cdot 4b(i,k) \leq \sum_{l \leq L} 4n(i,l,k), \forall k \in \{s_1, \ldots, s_2\} \quad (11)$$

so that $4b(i,k) = 1$ only when Q_{set} sets of nodes are allocated. Then, we add constraints:

$$(Q/Q_{\text{set}}) \cdot 4n(i,l,k) \leq \sum_{j \in C_l} 4n(i,j,k), \forall l \leq L, \forall k \in \{s_1, \ldots, s_2\}, \quad (12)$$

so that $4n(i,l,k) = 1$ only when Q/Q_{set} nodes in each set are allocated to the bid. The rest of the formulation is unchanged.

Introducing Sellers

The beauty of the MIP approach is that it extends elegantly to handle the case of a combinatorial exchange, with multiple buyers and sellers. We omit the full details of this formulation in the interest of space. Recall that we allow a seller to submit bid $b_i = (s, e, Q_l, Q_h, f_{\text{low}}, f_{\text{high}}, C)$, which defines: a start time $s \in \mathbb{T}$, an end time $e \in \mathbb{T}$, with s no earlier than the current time and $e \geq s$; a range $[Q_l, Q_h]$ on the number of nodes on which the seller is prepared to sell (the seller must hold appropriate resources on at least Q_l nodes for this bid to be valid); a range $[f_{\text{low}}, f_{\text{high}}]$ to define the share of resources that the seller is prepared to sell on each of between Q_l and Q_h nodes; and $C \subseteq \mathbb{N}$, the set of acceptable nodes to consider within the bid. There are two key ideas to modify the previous formulations to handle this case:

1. Change the RHS of the feasibility constraints, for instance replacing $\leq SHARE(j,k)$ with $\sum_{i:j \in C_i, k \in \{s_i, e_i\}} \beta(i,j,k)$, where $0 \leq \beta(i,j,k) \leq 100$ denotes the number of shares provided by seller i (that has node k in its acceptable nodes C_i and k in its time period) in the solution.
2. The additional constraints expressed in a seller's bid are handled through additional indicator variables $x(i,j,k) \in \{0,1\}$, where $x(i,j,k) = 1$ only when the user offering resources on node j sells between f_{low} and f_{high} shares, with additional constraints to ensure that between Q_l and Q_h nodes meet this requirement.

C Greedy Algorithms

C.1 Greedy (Model A)

```
 1: resources = [cpu, mem, net, disk]
 2: for all r ∈ resources do
 3:    availshares[r] ← numnodes * numslots * numshares[r]
 4: end for
 5: totalrevenue ← 0
 6: for all i ∈ bids do
 7:    i.bid ← i.value/i.numnodes
 8: end for
 9: Sort all bids by i.bids, in descending order
10: for all i ∈ bids do
11:    if ∀r ∈ resources, i.shares[r] ≤ availshares[r] then
12:       for s = [i.d, i.s] do
13:          for t = [t, t + i.h] do
14:             feasiblenodes ← ∅
15:             for n ∈ i.ok do
16:                if ∀t' ∈ t, ∀r ∈ resources, n[t'].shares[r] ≥
                   i.shares[r] then
17:                   Add n to feasiblenodes
18:                   if size(feasiblenodes) = i.numnodes then
19:                      success ← 1
20:                      exit for
21:                   end if
22:                end if
23:             end for
24:             if success = 1 then
25:                for all n ∈ feasiblenodes and r ∈ resources
                   do
26:                   n[t].shares[r] ← n[t].shares[r]−i.shares[r]
27:                   availshares[r] -= i.shares[r]
28:                   totalrevenue += i.value
29:                end for
30:                exit for
31:             end if
32:          end for
33:       end for
34:    end if
35: end for
```

C.2 Greedy (Model B)

Psuedocode for Model B is structurally the same as in Model A. The key is to replace all resource arrays (e.g. $availshares[r]$) by a single variable. This occurs in line 1 to 4, 11, 16, 26, and 27.

Example, replace 1 to 4 with:

availshares ← numnodes * numslots * numshares

C.3 Greedy Refined

We refine the above Greedy algorithms by making the bid value more concise. Specifically, we are interested in using the effective *per share* value that each agent submits. The algorithms for each model above would stay the same, except we replace line 7 of Greedy Model A with:

1: $i.bid \leftarrow i.value/(i.numnodes * i.duration * \sum_r i.shares[r]/numshares[$

And similarly for Greedy Model B with:

1: $i.bid \leftarrow i.value/(i.numnodes * i.duration * i.shares)$