A Resource Allocation Framework for Global Service-Oriented Networks

Justin Cappos, John Hartman

University of Arizona
Gould-Simpson, Rm. 749
Tucson, AZ 85719
(520) 621-2738
Fax: (520) 621-4246
{justin,jhh}@cs.arizona.edu

Submitted to SRMPDS. This paper will be presented by Justin Cappos.

Abstract

We study the problem of allocating resources on global networks where there is no central administrative control. We describe a framework that abstractly describes a number of components that are necessary in an auction system to provide users with a secure trading environment. We propose solutions for specific issues relating to auction granularity, cost/value effective bidding, bids on large resource sets, currency control, and computationally effective auction resolution. We then describe the application of our framework to PlanetLab and how the components would be implemented on this system.

Keywords: Resource Allocation, Conditional Bids, Escrow, Resource Reservation

1. Introduction

Effectively managing the allocation of resources on a heterogeneous, distributed network is a daunting task that faces many worldwide networks today. We abstractly define a framework for resource allocation that uses virtual currency and bidding. We then discuss how this framework would be applied to PlanetLab[14] to support the sort of programs and services that the PlanetLab user community is interested in running.

We broadly propose to solve the problem of deciding which users are granted resources on specific nodes in the system. Although a number of other groups have proposed solutions to this problem [1, 3, 5, 9], we solve it in a novel manner that does not have the problems that plague other systems.

In most systems, users wishing to gain resources generally either must wait for bidding intervals to obtain resources or may only gain resources in a short term / best effort manner. We accommodate the different needs of those users who want to run a new job immediately, those users who are happy to take any "spare" resources available on systems, and those users who are running large scale, long running programs. We believe that many users will want to run large scale programs from time to time and that they should not require mechanisms outside of the auction system to do so.

In addition, resource allocation systems must ensure that a stable base for service programs is available in the system. Our framework supports long-term trading of resources which helps to provide a static resource set to support these programs. In addition, many services and programs require resource guarantees so we consider Lottery Scheduling [20] or proportional sharing schemes [9] unacceptable to many users. Our system also facilitates the secure exchange of provable receipts by services in exchange for resources. We believe the negative social effects of refusing to honor a provable receipt are sufficient to keep services honest.

A key concern to users of any resource allocation system is the mechanisms involved in the bidding process. If the system is difficult to use or lacks flexibility and power then users will not use it. From an informal poll we learned that PlanetLab users are interested in being able to place bids that are cost/value effective, descriptive, take into account metrics not handled by the system, are conditional on the outcome of other bids, and are easy to phrase. Another concern is that users want to be able to make bids depending on whether or not other bids are resolved. Many systems use XOR based bidding languages to build complex bid types[1]. One problem with pure XOR based bids is that the size can grow tremendously when attempting to do complex queries. For example, an XOR based bid able to request between fifty to one hundred nodes would result in one bid item for each number between fifty and one hundred (and that is assuming that the system can find a set of any n nodes without introducing more XOR constructs!). We use a different mechanism to provide users with additional resources as long as it is cost effective.

We provide an explicit mechanism for converting virtual funds for real funds in the system. We believe that when a user is spending something equivalent to real funds they will make a more careful evaluation of decisions.

We use the term *service* to mean a set of distributed and cooperating programs delivering some higher-level functionality to end-users, other services, or the infrastructure itself.

Throughout this document the term *user* will be used to refer to what is likely a human driven interaction. *Buyer* and *seller* are also implicitly human agents. There is no reason why computer controlled agents may not perform these actions, but the examples will assume human interaction.

2. Challenges

There are many challenges for providing efficient, predictable, fair, and secure resource allocation in a global network.

- A global network will necessarily span multiple administrative domains. Users and services from each domain generally care more about getting the resources they need than the overall fairness and stability of the system. Furthermore, there isn't a central authority that can step in and resolve resource request disputes.
- The resources in such a system are heterogeneous, if for no other reason than different resources are located at different geographic locations within the network. The quantity of resources available at different locations within the network may vary greatly. The location of resources with respect to other devices in the network also is a key concern for many applications.
- Resource allocation should be efficient. Resources should not go unused when there is sufficient demand. The allocation mechanism itself should not require excessive amounts of resources.
- Users who bid on large resource sets should not have to pay significantly more than the sum of the individual resource costs.
- Resource allocation should be predictable and understandable by users in the system. Users should feel currency has true value and be able to estimate reasonable bid values.
- The language for expressing resource requests should be expressive enough to capture a service's complex needs, while remaining easy to use for simple requests.
- Resource requests must incorporate marginal costs, allowing users to express their desire for additional resources if they are cost-effective.

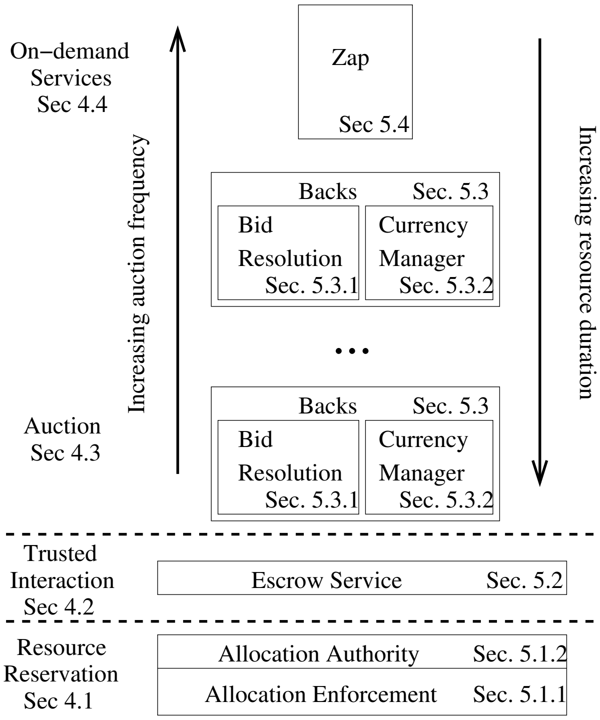

Figure 1: *This is an overview of the layered architecture we build a resource management system upon.*

- Resource discovery and allocation must be integrated so as to produce good allocations without requiring undue amounts of resources to do so.

3. Overview

We are developing a resource allocation system to address these challenges (Figure 1). There are three main components of the system. The lowest level is *resource reservation*, which is responsible for enforcing the current allocation of resources for programs. In the middle is *trusted interaction*, which is an escrow service that allows mutually untrusting parties to safely exchange resources and currency. At the highest level is *resource allocation*, which is responsible for deciding how to allocate resources to the various programs that require them. Each of these is addressed in the following sections.

3.1. Resource Reservation

At a low level the system must be responsible for acting on the resource allocation decisions. This layer provides real guarantees of resources and gives hard limits to programs to ensure they stay within their allowed resources. This layer also tracks which users own resources and which programs are being funded with those resources. Users may directly change the funding or ownership by directly contacting the resource reservation system. We use this layer to give us a secure interface for changing resource allocation on a node and to enforce resource allocation decisions.

3.2. Trusted Interaction

Any system that promotes the open exchange of valuable items requires a method to securely exchange items between mutually untrusting parties. We provide a facility that handles such transactions in a secure way. This layer is also used to provide receipts for services in exchange for currency or resources.

3.3. Resource Allocation

Resource allocation is handled by Backs, an auction system that allows users to buy and sell resources. We run copies of Backs with different resource granularities and bidding intervals to allow users to get different time increments of resources for different prices. This prevents users from being forced into a set bidding schedule (such as needing to bid every hour for resources on a program you wish to run for months or being forced to bid one week in advance for a one hour chunk of resources).

3.3.1. Bid Resolution

Backs has a bid resolution component that allows users to submit complex bids and have them resolved in the system. We use approximation algorithms to handle complex bids and allow the bidder to pay for processing resources to resolve their bid. XOR based bidding schemes that use decoupled resource discovery [11] and bid resolution [1] have efficiency issues. For example, a bid for any ten nodes so that no two nodes have internode RTT $>$ 100ms results in a tremendous number of answers from the resource discovery system. Using XOR based bidding schemes to evaluate these results is not efficient. We integrate resource discovery and bid resolution to allow us to quickly and efficiently approximate such queries. Users should be able to place bids that are cost/value effective, descriptive, take into account metrics not handled by the system, are conditional on the outcome of other bids, and are easy to phrase. Our bid resolution mechanism provides all of the above.

3.3.2. Currency Manager

The *currency manager* regulates resource prices, avoiding sharp swings in prices. A problem in many systems is that the value of currency fluctuates so wildly it is difficult for users to use the system or determine what is an acceptable bid for resources. We solve this problem by regulating the amount and value of currency in the system through the currency manager buying and selling resources. We will experiment with mild inflation and taxation as other methods to regulate currency and provide incentive for users to spend funds in the short term (as funds either grow less valuable through inflation or recede through taxation over time).

Another problem in auction systems is user indifference when spending currency. A user can build up currency over a long period of disuse of the system and then when they need something, bid a tremendous amount of currency. We make resource and currency decisions of more consequence to users by allowing users to exchange virtual and real currency. We assume that when a user is spending something equivalent to real funds they will make a more careful evaluation of decisions.

3.4. On-demand services

Services may discover they need additional resources immediately, perhaps to cope with unexpected demand. To handle this situation, at the top of the auction hierarchy is a “buy it now” service called Zap. Zap is the convenience store of the system. There users can buy a small limited set of resources at any time but for a proportionally high price. Zap purchases resources from Backs by attempting to determine future users’ needs. While waiting for a user to purchase its resources, Zap funds highly interruptible programs. This provides us not only with better utilization of the system, but also resources on-demand in the system.

The end result of our auction system is a forum that is fair in the handling of user requests and allows users an easy way to obtain resources to run desired programs in a reasonable amount of time. Our system also provides long-running programs the resources that they need to continue operation.

4. PlanetLab Prototype

This section describes a prototype implementation we are developing for PlanetLab.

4.1. Resource Reservation

Resource Reservation is described above as a single entity, but on PlanetLab it is divided into two subcomponents to leverage existing tools for resource allocation: *allocation enforcement* and *allocation authority*.

4.1.1. Allocation Enforcement

PlanetLab currently has a system called Sirius that handles resource reservations. Sirius has been given an allocation of resources that it can re-allocate to other programs. The mechanism is fairly crude at the moment, but the goal is to give those users that assign programs to ”wait in the Sirius queue for their turn” as much CPU capacity as they would have received during quieter times rather than thrash in an overloaded system. So conceptually, Sirius is a resource control mechanism on PlanetLab. We intend to work with David Lowenthal to modify

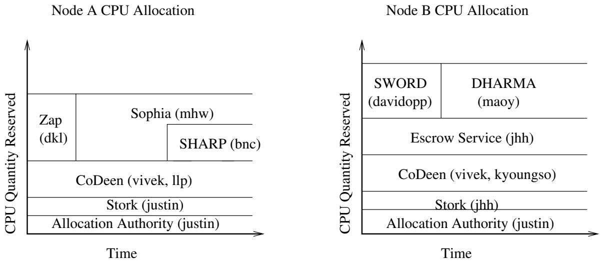

Figure 2: *Example CPU resource allocation information for two nodes can be viewed as a table with time on the x axis and resource amounts on the y axis. The user will have an area on the graph corresponding to a set of resources they control for a period of time. Each node may have a different quantity of each resource and thus may have a different total area of resources. The program names are succeeded by the funding users (in parenthesis). Note that multiple users are able to pool resources to fund the same program (as is the case for CoDeen).*

Sirius to meet the resource allocation demands for our framework and separate out the ”queuing”, etc. portions of Sirius into a separate service.

4.1.2. Allocation Authority

The allocation authority controls who owns resources on each node. This service maintains a local table (see Fig. 2) that is used by the allocation enforcement software to limit the resources used by specific programs. The individual node software is responsible for enforcing resource usage boundaries. The purpose of the allocation authority is to provide a secure interface for modification of which users have ownership of specific resources and to allow those users to fund different programs with those resources.

The allocation authorities schedule resource intervals in small time blocks (perhaps 6 hours) on PlanetLab. However, users may freely reassign those resources to fund different programs at any point. The allocation authority keeps information about what resources are reserved for different users from the current time period into the future.

Sites may want to give resources to specific services to attract them to deploy on a node. For example, the Stork service [19] provides package management and disk space saving functionality so it is desirable for both users and nodes to have it deployed, thus many nodes may freely give resources to it over long periods of time.

4.2. Escrow

The *escrow service* facilitates the exchange of resources and currency within our system. It allows mutually untrusting users to perform transactions through a trusted third party.

The escrow service runs on a few nodes and acts as an intermediary for transactions. The basic model is that a set of users will come to an agreement upon a set of transactions. For example, User A wants to trade resources with User B. User A and User B will get a unique transaction ID marked with a time from the escrow service and they will each sign a copy of the proposed transaction. Each transaction consists of the parties involved, a Transaction ID, and a timeout (if the transaction cannot be completed within this time it is canceled and ownership is returned to the original owners). After this document is submitted to the escrow service, each user must transfer ownership of the resources specified to the escrow service. Assuming this all completes within the timeout period, resources are then transferred by the escrow service to the applicable users.

The escrow service uses the allocation authority’s interface to perform transactions. This means that other services that use this interface to handle transactions may use the escrow service to interact. For example, in Stork we will use escrow transactions to gain the disk space necessary to install a new package for a program. Stork and the user funding the program will sign a transaction where Stork gives a “receipt” for the package in exchange for the disk space the user provides. Notice that this will not prevent the user from being cheated in such a case (by Stork maliciously refusing to install the package), but the user is provided with proof that they were cheated.

4.3. Backs

Backs represents a central tradinghouse where users buy and sell resources. Instead of requiring users to barter directly for resources [5], Backs uses a virtual currency to match buyers and sellers automatically. Bids for resources and the sale of resources are provided to Backs via the escrow service (described in Section 4.2). Backs charges a small fee to bidders to discourage frivolous bidding and also to recoup the resource costs of running the Backs program (see Section 4.3.2).

Backs provides users with a “receipt” that either their bid will be met for at most the agreed price or the user will be refunded their currency. In the same way users who sell resources will be refunded their resources or given at least that amount of currency at the conclusion of the auction.

Some systems use a marginal cost ranking [3] to order bids and then handle them individually. One problem with this is that it becomes very difficult for users to get large sets of resources. Consider an example where we have 3 users who are bidding on resources on 3 equivalent nodes. Suppose each of these users wants the resources on one node. If a new user wants to get resources on any one of these nodes, they need only bid more than the lowest bidder. If this new user instead wanted resources on all 3 nodes, they would need to bid more than the highest bidder *per node*. In other words, you must bid enough currency so that your marginal cost is higher than the users who are competing for the resources you need.

In addition, items like inflation or taxation (see Section 4.3.2) are more problematic for users requesting large jobs because loss of a percentage of value implies that they lose more value.

To offset the disadvantages of large bids we use a modified marginal cost system. We rank bids according to a formula $\frac{a^x}{r^y}$ where a is the amount of the bid and r is the quantity of resources the user is bidding on. In such a formula, a pure marginal cost ranking is represented by $x = y$, while we prefer to tilt the ranking towards higher bid amounts thus in our system $x > y$. We intend through experimentation to discover sane parameters for the x and y values which set the ranking in our system.

Backs first orders bid requests according to our modified marginal cost formula and then resolves each bid individually. When attempting to resolve individual bids in an auction, Backs resolves the bid in an attempt to maximize total bid surplus in the system (similar to the approach used in [12]).

4.3.1. Resource Queries

Users should have the ability to issue requests in an understandable bidding language that is flexible and powerful. While the speed of bid resolution is important, queries must also be:

- cost/value effective: Users want to be able to place bids that will provide them with a certain subset of resources and then additional resources as long as it makes sense economically.
- descriptive: Users should be able to put forth requests that obtain nodes on separate sites, that provide for the best network coverage, that correspond to a low latency group and many other factors. The requirements of the user shouldn't change to meet the constraints of the query language.
- flexible: Users may have program specific metrics in their requests (for example nodes in separate BGP ASes, nodes connected by DSL links, etc.). Thus users should have the ability to request nodes based upon metrics that we cannot predict without requiring us to modify the query system.
- conditional: A user may wish to only submit additional bids if previous bids meet a specific outcome. For example, users may want to bid on resources for a completely different program if their original requirements aren't met.
- simple to use: Users want to be able to easily put forth complex requests without learning a strange query language. Requests should "make sense" and yet be flexible enough to state any reasonable request.

Backs provides a resolution mechanism to address these concerns and issues.

Requests for complicated resource descriptions are processed using time purchased by the bidder. Backs allows complicated resource query types and use approximation algorithms to determine the feasibility of the bid.

One approach is to separate resource discovery from the bid resolution mechanism [1]. The resource discovery phase finds all resources that meet the user's requirements; the resource allocation phase finds the cheapest subset. A problem is that it is not efficient to have the resource discovery mechanism return a huge number of answers that all must be sifted through to find a feasible bid. For example, if we request a set of resources on any five PL nodes that have internode RTT < 100ms, there will be a tremendous number of matches. Giving this output as input to the bid matching system results in a tremendous amount of slowdown because it considers all cases.

We believe that the resource discovery mechanism *must* be tightly integrated with the bid resolution mechanism for efficiency. As a result, we combine the resource discovery and bid resolution steps to form an approximation algorithm that attempts to minimize cost while maximizing the accuracy of the match. For example, a user may spend a small amount of currency that pays for their bid computation time and then request a set of resources on a set of nodes that provide the best network coverage of the Internet. The system spends a small number of seconds of processing time in order to compute the best approximation with a low cost for their request. The algorithm discards matches and removes nodes that are too expensive early on in the algorithm in an attempt to speed up processing time.

Backs allows users to use built in approximation algorithms to certain problems that allow expressive bid types. We consider many simple problems that can be solved optimally such as bids based upon sets of nodes (a request for any 10 nodes that have the applicable resources). We also consider hard problems such as the K-clique problem, graph centers problem, dense subgraph problem, etc. The user may choose to tradeoff the accuracy of the solution for the cost of the resources.

For example, users can request resources on nodes so that all the links obey a property (such as < 100 ms RTT time or have at least 5 IP hops) which is the K-clique problem. In addition, the user may value the total internode latency of the selected nodes as a 1/1 proportional factor of the cost. In other words, if we can reduce the total internode latency by 1/2 we are willing to pay twice as much. One way of gaining an acceptable answer would be to use a simple greedy algorithm to choose a cheapest node and then choose the cheapest nodes with low latency and continue this process until we have a match (backing out and trying other nodes if this attempt doesn't work). Now that we have a solution to the problem, we know we can discard any other solutions that have greater cost. So we may aggressively discard high cost nodes and quickly backtrack out of fruitless paths to more quickly refine our search. We are currently developing other techniques to give approximate solutions to problems of interest.

Another problem is that in order to consider metrics in the bidding process they must be part of the resource discovery mechanism. This means that users must lobby for modifications to the existing system to be able to bid based upon new metrics. We believe that the diversity of our user group may be such that they will be interested in metrics the system has not measured. Backs provides a system where users may place their own metrics for nodes in the system that will be evaluated when performing bid resolution.

Users may place their own metrics for bids by providing a valuation for nodes or links along with their bid. The purpose of such a list of values is to allow Backs to compute queries based upon metrics without needing to measure or understand them. For example, a query might be submitted that references the node's BGP Autonomous System Number. The corresponding bid may request a set of 25 nodes that have distinct BGP ASNs. In this way we allow users the flexibility to specify their own metric types that are submitted along with their bids. To state another example, suppose that a user only wants to deploy their program on nodes where a certain service is running, for example Stork. Stork may post a list of values on their website that lists the nodes on which it is running and then users can upload this to allow their bids to take this factor into account without requiring Backs to understand anything about Stork.

To provide a similar functionality to XOR bids while solving the problem of having the quantities of bids grow tremendously, Backs allows users to submit conditional bids that will only be submitted if the original bid reached a specified outcome. The user may request the consideration of a bid given the acceptance or rejection of a previous bid.

Conditional bids are useful in many cases including gaining additional resources as long as it is economically feasible or attempting to gain resources of a different type in the case that the first set is not available. For example, to request resources on any 50 to 100 nodes in our language a user would place a bid for 50 nodes and then attach a success conditional bid that adds a node until the bid funds have been exhausted or there are 100 nodes. In addition, we support the notion of adding resources as long as it is cost/value effective through conditional bids. The user submits a bid for the minimal set of resources that they need to be able to run their program. As a success conditional bid, they request additional resources a piece at a time until there are no longer beneficial resources available for the amount they are willing to spend per unit.

4.3.2. Valuing Virtual Currency

The universe in which Backs works (the set of resources total in the system along with the set of currency) provides a model which we attempt to regulate through several mechanisms to increase and decrease the value of currency.

Backs allows users to exchange real currency and virtual currency. This allows virtual currency to have true value to users and provides users with a real incentive not to waste currency. It also causes resources to be valued by the users in the system as resource use is indirectly linked to currency.

Backs also regulates the amount of currency in the system by purchasing long-term resources from Resource Managers and selling those resources. If Backs wishes to remove currency from the system, it will charge more for those resources it puts up for auction and purchase long-term resources for low amounts. Conversely, to add currency to the system it will purchase long-term resources for higher amounts and sell resources in the short term for low values.

Similarly, to discourage hoarding of currency, we will intentionally introduce mild inflation into the system to grease the wheels of commerce [8]. The gradual inflation means that the currency that is in a user's account will only decrease in value the longer they maintain it. Periodically, when the currency is inflated to the point where a single unit of currency has little value we will perform a system wide "currency exchange" which effectively halves the amount of currency in the system to prevent the number of currency units from becoming unwieldy. We intend for currency exchanges to be rare (perhaps once a year). In this case, all users of Backs will be notified in advance of the exchange. As input in our system, we have a "currency type" that will be incremented with every exchange and requests in terms of old currency will be automatically converted.

We automatically regulate the currency in the system by the use of a Currency Manager which has a notion of the value of resources in terms of real currency (based upon hardware costs, resources in the system, etc.) and attempts to keep a rough balance in the system. As the number of resources in the system changes, it will in a similar manner alter the amount of currency in the system to compensate. It also decides by looking at the level of inflation upon the timing of the currency exchanges.

The bid fee in Backs is used to allow Backs to be able to "pay for itself" and recoup the cost of computing bids. Backs can recover enough currency through charging a fee for bids to purchase enough resources to run itself. In other words, Backs acquires enough resource value from incoming bids to support the cost of resources to operate it. We intend to fluctuate the bid fee slightly in Backs also to regulate currency in the system.

4.4. On-demand resources

Services may suddenly require additional resources. Submitting a bid and waiting for the next auction to complete may not be acceptable. The solution to this problem is *Zap*, a service that provides on-demand resources to users at a fixed price. Conceptually, Zap is a version of Backs that runs with infinite auction interval and has short resource durations.

Zap is a service that optimistically purchases resources it thinks on-demand users may wish to use for their program. It is not privileged in that it is considered to be a Backs user, the same as any other user requesting resources. It makes a determination based upon what it was able to sell in the past and attempts to predict future requests and usage. It then charges a premium on those resources which are being transferred to users as they realize the need. It is conceptually a convenience store for PlanetLab resources, with Backs representing ordering groceries for delivery from a supermarket. Zap has limited selection and higher prices but items are available at all times.

The resources of Zap go to waste until they are purchased by a user (if ever). There are many programs that can use resources in a piecemeal fashion to compute part of a larger problem (such as SETI at home [17], The Great Internet Mersenne Prime Search [6], etc.). Zap distributes and funds such programs on nodes (for a very low price) until those resources are purchased by another user. This aspect of Zap is similar to the DAWGS system [10] proposed by David Lowenthal.

5. Future Work

As this is a design document, we have a significant portion of the system left to implement.

We intend to experiment with different bid surplus distribution schemes based upon resource valuation, using the second-trade model as a baseline.

The amount and type of currency value regulation is also an area where we intend to experiment and analyze different approaches to see what works well in our system.

Many aspects of the design of the bid query language have been left underspecified.

6. Acknowledgements

We would like to thank Larry Peterson, David Lowenthal, Michael Stepp, and Prasad Boddupalli for their helpful comments and suggestions. We would also like to thank Patrick Homer for subconsciously planting the name Zap.

7. References

[1] AuYoung, A., Chun, B., Snoeren, A., Vahdat, A., "Resource Allocation in Federated Distributed Computing Infrastructures", Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure, 2004.

[2] Burrows, A., Abadi, M., Needham, R., "A logic of authentication", ACM Transactions on Computer Systems, 1990.

[3] Chun, B., Ng, C., Albrecht, J., Parkes, D., Vahdat, A., "Computational Resource Exchanges for Distributed Resource Allocation", http://berkeley.intel-research.net/bnc/papers/share.pdf

[4] Foster, I., Kesselman, C., "Globus: A Metacomputing Infrastructure Toolkit", International Journal of Supercomputer Applications, 11(4):115-128, 1997.

[5] Fu, Y., Chase, J., Chun, B., Schwab, S., Vahdat, A., "SHARP: an architecture for secure resource peering", Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, 2003

[6] Mersenne Prime Search. http://www.mersenne.org/prime.htm

[7] Grid. http://www.globus.org.

[8] Inflation - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Inflation

[9] Lai, K., Rasmusson, L., Adar, E., Sorkin, S., Zhang, L., Huberman, B., "Tycoon: an Implementation of a Distributed Market-based Resource Allocation System", http://arxiv.org/pdf/cs.DC/0412038

[10] Lowenthal, D., Peterson, L., Hartman, J., Cappos, J., E-mail discussion about DAWGS and Backs, September, 2004

[11] Oppenheimer, D., Albrecht, J., Patterson, D., Vahdat, A., "Distributed resource discovery on PlanetLab with SWORD", First Workshop on Real, Large Distributed Systems (WORLDS '04), December 2004

[12] Parkes, D., Kalagnanam, J., Eso, M., "Achieving budget-balance with vickrey-based payment schemes in exchanges", In Proceedings of International Joint Conference on Artifical Intelligence, pages 1161-1186, 2001.

[13] Peterson, L., "Dynamic Slice Creation", PDN-02-005 Draft, 2002.

[14] Peterson, L., Anderson, T., Culler, D., Roscoe, T., "A Blueprint for Introducing Disruptive Technology into the Internet", PDN-02-001, 2002.

[15] Peterson, L., Roscoe, T., "PlanetLab Phase 1: Transition to an Isolation Kernel", PDN-02-003, 2002.

[16] Sandholm, T., Subhash, S., Gilpin, A., Levine, D., "Winner Determination in Combinatorial Auction Generalizations", ACM AAMAS, 2002

[17] SETI@home: Search for Extraterrestrial Intelligence at home. http://setiathome.ssl.berkeley.edu/

[18] SSH. http://www.ssh.com/.

[19] Stork. http://www.cs.arizona.edu/stork/.

[20] Waldspurger, C. A., Weihl, W. E., "Lottery Scheduling: Flexible Proportional-Share Resource Management", Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI '94), pages 1-11, Monterey, California, November 1994