Markets are Dead, Long Live Markets

Kevin Lai
Information Dynamics Laboratory, HP Labs

Researchers have long proposed using economic approaches to resource allocation in computer systems. However, few of these proposals became operational, let alone commercial. Questions persist about the economic approach regarding its assumptions, value, applicability, and relevance to system design. The goal of this paper is to answer these questions. We find that market-based resource allocation *is* useful, and more importantly, that mechanism design and system design should be integrated to produce systems that are both economically and computationally efficient.

Categories and Subject Descriptors: K.6.4 [**Management of Computing and Information Systems**]: Systems Management; D.4.7 [**Operating Systems**]: Organization and Design

General Terms: Design, Economics, Management, Performance

Additional Key Words and Phrases: Tycoon, resource allocation, markets

1. INTRODUCTION

A key advantage of the Internet, peer-to-peer file sharing networks, and systems like PlanetLab [Peterson et al. 2002] is the sharing of computational resources. This provides a variety of benefits, including higher utilization, increased throughput, lower delay (due to dispersion of resources in the network), and higher reliability. However, resource allocation remains an issue. The problem is how to allocate a shared resource fairly, with economic efficiency (where efficiency is the ratio of the actual total benefit for all users to the optimal total benefit), and at low cost.

Scheduling algorithms like Proportional Share are a partial solution. The problem with PS is determining how to set the weights. Assuming that the values of tasks varies over time, no single set of weights will suffice. Setting of weights cannot be left to users because they have a strong incentive to always ask for the highest possible weight. Having a non-omniscient system administrator set weights is time-consuming and likely to result in an economically inefficient allocation.

Economics and game theory offer an alternative. The area of *mechanism design* is concerned with algorithms where individuals optimizing their own utility results in high overall utility. A market (or auction) is an example. In the resource allocation context, as users optimize the benefit that they receive from their applications, the mechanism optimizes the overall efficiency of the resource allocation, without the

klai@hp.com 1501 Page Mill Road, MS-1139 Palo Alto, CA 94304

intervention of an administrator.

This is not a novel idea. Researchers proposed this approach as early as 1988 [Ferguson et al. 1988] [Malone et al. 1988], and there were likely earlier ones. Since then, several researchers [Waldspurger et al. 1992] [Regev and Nisan 1998] [Stratford and Mortier 1999] [Chun and Culler 2000] [Ng et al. 2003] have pursued it. Unfortunately, there have been few implementations [Waldspurger et al. 1992] [Chun and Culler 2000] [Coleman et al. 2004]. Given 17 years of research, it is surprising that there is not even one commercially or freely available market-based resource allocation system. Given the impact of poor resource allocation on systems like PlanetLab, this appealing approach and a large body of enthusiastic publications, why have so few market-based systems been built? We believe the lack of operational market-based systems is because questions persist in the minds of system designers about the value of the economic approach, its applicability, and its relevance to system design.

In this paper, we examine some of these questions using a combination of qualitative arguments and simulation. We do not claim to have conclusive answers. Instead, we hope to provide sufficient affirmative evidence that more real market-based systems should be built, deployed, and evaluated.

2. AN EXAMPLE

In this section, we examine Proportional Share (PS) scheduling as an example of the problem that market-based resource allocation seeks to solve. PS gives user i with weight w_i a $w_i / \sum w$ share of the resources. Using PS hierarchically, the user can assign these resources to his tasks. For example, if Alice has weight 2 and Bob has a weight of 1, then Alice has two-thirds of the total resources for her tasks and Bob has one third. Suppose that this is necessary because Alice must process twice as many queries as Bob. This works well if Alice is always doing something more important than Bob, but sometimes Bob will have an important task (e.g., serving a client query) while Alice has a less important task (e.g., non-time-critical background jobs like garbage collection). In this case, Alice's task will still get most of the resources. This is an economically inefficient situation because although Alice receives some small benefit, Bob receives less than he could, and the total benefit is less than if Bob received most of the current resources. This is a common situation because for most users and their applications, the arrival process of important work is highly bursty (e.g., a web/email/file server).

One solution is to rely on Alice to yield resources (e.g., using `nice` or other means to set lower weights on tasks) when doing less important tasks. Unfortunately, Alice has an incentive to deny Bob the resources and use them herself. This is an example "Tragedy of the Commons" [Hardin 1968] where optimizing for individual utility results in low overall utility. On the other hand, optimizing for overall utility depends on knowing the relative values of every task being run. Unfortunately, users cannot be relied on to accurately report these values without an honesty incentive. Some users may behave *obediently* by yielding resources or honestly reporting task values, but those that do not (*strategic* users) lower the efficiency of the system, and, worse, provide an incentive for obedient users to become strategic.

Another solution is to have a system administrator monitor the system and dy-

namically change the weights of the users to maintain high efficiency. However, this is expensive, time-consuming, and error-prone, and it does not scale to large numbers of users and resources.

Market-based resource allocation addresses this problem. For example, Alice and Bob are issued a currency with income rates in the ratio 2 to 1. They use these credits to bid for resources in a market where user i with bid b_i receives a $w_i / \sum w$ share of the resources. When a user has used her share, then her bid is deducted from her balance of credits. Since Alice is aware that garbage collection is a less critical task than serving client queries, she will spend fewer credits when doing the former and more doing the latter. The mechanism provides an incentive for users to truthfully reveal how much they value resources. This allows Bob to get more resources than Alice when he is processing queries and she is doing garbage collection. Over the long-term, Alice can still process twice as many queries as Bob (assuming similar workloads).

3. RESOURCE ALLOCATION MARKETS

In this section, we examine some questions about market-based resource allocation.

3.1 How strategically do users behave?

In particular, do users consume resources that others would benefit more from, thus leading to low overall economic inefficiency? If this is false, a market is unnecessary.

Hardin [Hardin 1968] named this effect the "Tragedy of the Commons". It is a basic assumption of economics, game theory, and businesses such as EBay, so this it is likely to have at least some general validity. The question is whether it applies specifically to computer systems. In fact, there are several systems examples. Adar and Huberman [Adar and Huberman 2000] showed that free-riding (a form of strategic behavior) on the peer-to-peer file sharing system Gnutella was widespread. Spam, denial-of-service attacks, and spyware are all examples where even a few highly strategic users can consume resources that would otherwise benefit other users and wreak havoc on the system. On PlanetLab, strategic behavior manifests as users who continuously consume resources (even during very high demand periods) even though they would receive as much benefit by running at low demand periods.

These are examples of intentional strategic behavior, but more unintentional strategic behavior is also harmful. Suppose that Alice has coded her program so that it meets her performance goals with the resources available to her. However, she could code it to use fewer resources. Meanwhile, Bob needs more resources than are available to him. If the cost for Alice to optimize her program is less than the benefit that Bob would receive from the extra resources, then this is an inefficient situation. Without an economic mechanism, Alice does not have an incentive to fix her program.

This is not to say that all users are highly strategic and attempt to cheat each other at every opportunity. Neither are they completely obedient. Instead, most are moderately strategic, where the desire for more resources is balanced by attention to the more traditional systems metrics of computational efficiency, reliability, security, complexity, and ease-of-use. Similarly, the attention of system designers, especially those designing scalable systems, should also be balanced with equal concern given

to strategic behavior.

3.2 What benefits do markets provide over long-term PS?

In long-term Proportional Share, user i's share of the resources is $w_i / \sum w$ over a longer time period (e.g., a week or year) than the 10 milliseconds of a typical CPU scheduler. This allows a user to shift her usage between different time periods and provides some of the flexibility of a market-based system.

However, long-term PS is not sufficient to reach economic efficiency. It does not encourage users to shift usage from high demand periods to low demand periods. It also does not encourage users to shift usage from high demand resources to low demand resources. For example, a system could have high demand for CPU cycles, but low demand for physical memory. The aggregate performance for applications on the system would increase if applications which could easily trade CPU cycles for memory pages (e.g., by caching) would do so. In a typical application of PS, the CPU and memory would be allocated separately and applications would have no incentive to use fewer CPU cycles and more memory [Sullivan and Seltzer 2000].

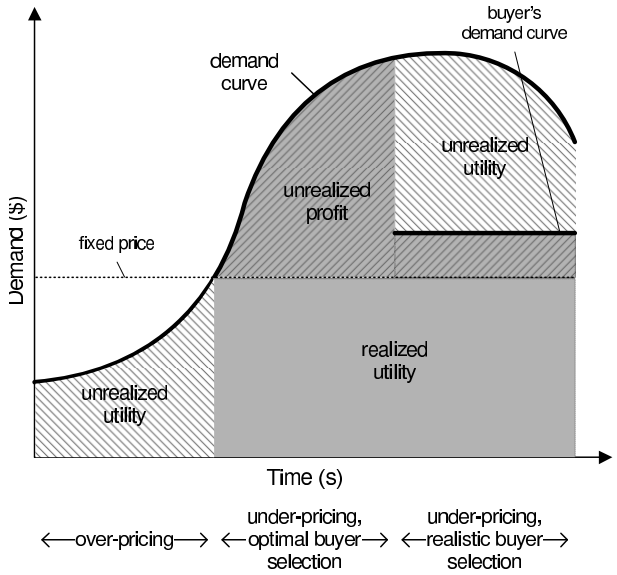

Fig. 1. **Fixed and Variable Pricing.** This figure shows a variable demand curve over time and how its efficiency compares to a fixed pricing curve.

One solution is to use fixed pricing. Suppose each CPU cycle costs \$10 and each memory page costs \$1 regardless of demand. In this case, an application does have an incentive to use more memory pages in order to save CPU cycles, so this begins to address the multi-resource problem. Pricing also provides a way for applications to express quality-of-service needs. For example, an application may not need many cycles, but instead needs them quickly after an interrupt has occurred [Duda and

Cheriton 1999]. Time slices could be priced differently based on how quickly they are scheduled.

The problem is how to set these prices. Figure 1 shows that no fixed price is as economically efficient as a variable price, assuming variable demand. In the left area, the demand is below the fixed price, so the buyer is unwilling to buy the resource and the utility that the buyer would have gained by using the resource is unrealized (indicated by the striped area under the demand curve). In the middle area, someone is willing to pay the fixed price, so the buyer is able to use the resource and gains some utility (indicated by the gray area under the demand curve). Assuming that the seller chooses the optimally efficient buyer (i.e., the one willing to pay the most), then the difference between the demand curve and the fixed price is unrealized profit for the seller (indicated by the striped gray area). Unrealized profit contributes to overall inefficiency in some cases (see § 3.4). However, without a covert channel, a fixed price seller cannot distinguish among the demand curves of potential buyers, so the actual buyer's demand curve will probably be lower than the optimal buyer's and the area between them is more unrealized utility. In general, the more variable the demand, the worse the efficiency loss of fixed prices.

One definition of a market is a way to set prices so that they follow the demand. As a result, most, if not all of the utility under the demand curve in Figure 1 would be realized using a market.

3.3 How fair are markets?

Markets allow users to save currency. Could this allow a user to save enough currency to starve out other users? Are markets unfair in some other way?

One definition of fair is the degree to which users receive resources in proportion to an exogenously determined weighting system. For example, if Alice has a weight of 2 and Bob has a weight of 1, Alice should get twice the resources of Bob over an arbitrarily long time interval. Assuming that Alice and Bob have demand curves that are equally correlated with the overall demand, the market described in § 2 is fair by this definition.

Another definition of fair is the degree to which users do not get starved. Some combination of Alice spending her credits and Bob saving his would allow Bob to starve Alice. By this definition, markets are potentially unfair.

This is not necessarily a bad thing. Alice chose to spend her credits to acquire resources immediately with the knowledge that she was reducing her future ability to acquire resources. Bob saved his credits so that he could acquire more resources at some future point. The situation may be unfair for Alice, but switching resources from Bob to Alice would be unfair for Bob. In addition, it removes some of the incentive for users to use resources carefully, save credits, and earn credits (in systems where users can earn credits). All of these result in lower economic efficiency.

On the other hand, the more that Bob hoards credits, the more unpredictable the system becomes because the other users cannot predict when Bob will make a massive purchase and starve them out of the system. This is less of a concern for larger economies because it is less likely that Bob can corner the market when it is large or the fraction of all credits in circulation that Bob controls is small.

Nonetheless, this is a concern for small systems or when Alice requires a specific

resource. One solution is to monitor user's saved credits and redistribute credits from the wealthy to the poor. As experience with taxation in the physical world indicates, a moderate level of taxation preserves the incentive to use resources carefully and to earn and save credits, while providing a disincentive to hoard them. The field of macro-economics contains several methods for determining what a moderate level of taxation is.

The conclusion is that markets are potentially unfair, but they can mitigate this by redistributing credits. System designers can tune market-based systems to make a tradeoff between efficiency and fairness that is appropriate for their users.

3.4 How useful are markets when real money is not involved?

Economic mechanisms provide an intuitive mapping between resource allocation and a business model for selling resources. However, in some cases, the resources are just being shared and not sold (e.g., employees sharing machines in their company's data center).

Markets are still useful in these situation. The simplest configuration is an open-loop economy, where the resource owner issues credits to users who can then spend them on resources. The main efficiency gain results from users having an incentive to truthfully reveal the value of their tasks (as shown in § 4).

Another alternative is a closed loop economy where users both consume and provide resources. PlanetLab could be run this way. A closed loop economy provides more incentives for efficiency than an open loop one: as prices rise, so does providers' profit, which increases the incentive to provide resources. This raises competition and eventually causes prices to fall. At no point in this cycle is real money necessarily involved.

3.5 How useful are markets for systems with low utilization?

Given that most systems, whether they are servers or network links, have low average utilization, this is an important question. A more general question is whether any resource allocation is useful for low utilization systems.

If a system has uniformly low utilization, then it is unlikely that resource allocation will improve performance. However, many systems have a bursty load. Examples are web servers, databases, and routers. There will be periods of contention when the performance of the system is dependent on resource allocation decisions. Although uncommon, these busy periods are important from a reliability perspective because they cause requests to time out, be dropped, etc.

In addition, many systems have low utilization because they are not being shared. Applications are frequently isolated on their own private platforms. This results in some systems being completely loaded and most others sitting idle. Experience with the Internet and PlanetLab suggests that when applications are moved from isolated systems to a shared-resource system, the resources become more utilized.

3.6 Are markets predictable?

Markets may allocate resources efficiently on average, but prices for resources fluctuate, so how can users predict the cost of the resources that they need?

In this context, we define predictability (i.e., performance isolation or quality-of-service) as the ability to provide a fixed amount of resources over a period of time

Parameter	Value
Users	10
Running Time	1000s
Task Interarrival	Gaussian, μ: [1s, 120s], σ: $\mu/2$
Task Size	Gaussian, μ: 10, σ: 5
Task Deadline	Gaussian, μ: 75, σ: 37.5
Task Value	Uniform, range: (0, 1]

Table I. Simulation parameters

with high probability, regardless of the demand put on the system. An example of an application needing predictability is a web server that needs to serve n requests per second with 99% of requests served within d seconds.

The market from § 2 can provide this capability by adding the ability to reserve shares, where a share is a fixed percentage of a resource (e.g., .1% of a 1 GHz CPU = 1 MHz). These shares have a fixed duration and are sold using an auction. The operator of the example web server calculates the resources necessary to meet his needs and bids for those resources. The cost of this approach is that resources may be under-utilized because some resources may be reserved, but go unused.

Although similar to techniques used in non-market systems [Stoica et al. 1997] [Sullivan and Seltzer 2000] [Duda and Cheriton 1999], the market allows the predictability mechanism to be used more efficiently. The problem with these systems is the difficulty in deciding how much of the resource should be devoted to best-effort service and how much to reserved service. The optimal split will likely vary significantly over time. Users would prefer reserved service if the cost to them is equal, but reserved resources are less efficient than best-effort because of the potential for under-utilization. The benefit of the market is in forcing users to consider whether they really require reserved resources and in helping the system determine reserved/best-effort split. High bids for reserved service will cause users who can tolerate best-effort to do so and indicate to the system to reserve more resources. Low bids will do the opposite.

4. SIMULATION RESULTS

In this section, we present preliminary simulation results quantifying the efficiency gains of a market. The basic idea is to simulate a single CPU server running the CPU-intensive tasks of several users. We examine different resource allocation algorithms and different user behaviors.

The simulation parameters are summarized in Table 4. A user may have more than one pending task, but users only run one task at a time. If a task completes by the deadline, then the user receives $value * size$ utility, otherwise nothing. There is one server providing resources for tasks. A task finishes when it accumulates resources equal to its size. A user can run one task, switch to a new task, and then switch back to the first task without cost.

The server uses one of two resource allocation schemes: Proportional Share or Market Proportional Share. With Proportional Share, the server allocates its resources to tasks according to the weight assigned by the task's owner. With Market

Proportional Share, users have an income of $1 credit per second. If Alice spends 1 credit and Bob spends 2, then Alice's task gets .66 resources, while Bob's task gets .33. The income can be saved.

We simulate three different user behaviors: obedient, strategic without a market, and strategic with a market. Obedient users assign a weight to their tasks equal to the task's value. Non-market strategic users assign the maximum possible weight to all of their tasks. Market strategic users budget their credits according to the task. The idea is to spend more credits on more valuable tasks and to apportion the credits over the lifetime of the task. Market strategic users assign the following credits per second to run their most valuable task: $(balance * value)/(deadline - now)$. *balance* is the user's current credit balance, *value* is the value of the user's most current valuable task, *deadline* is the deadline of that task, and *now* is the current time.

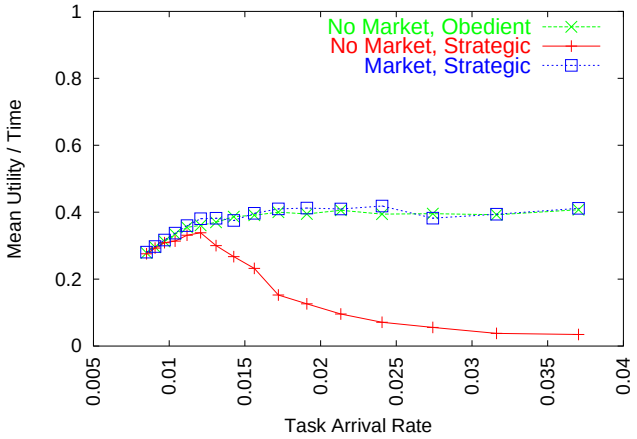

Fig. 2. The utility of different user behaviors and mechanism as a function of system load.

Figure 2 shows the simulation results. The y-axis is the mean utility per host per time unit. The x-axis shows the mean task arrival rate in the system and is a measure of overall system load. Each point in the graph is a run of the simulator.

As the load increases to the right, the obedient users without a market are able to maintain a high level of utility. However, users have no incentive to be unilaterally obedient. Instead, their incentive is to strategically give a high weight to all their tasks. The plot of the non-market strategic users shows that they are able to maintain a high level of utility when the system is lightly loaded (from 0.0 to 0.0125), but as the load saturates the system, utility drops to zero. At this point the system wastes resources running tasks that never meet their deadlines and therefore provide no utility. In a system without a mechanism or significant social pressure, some users inevitably become strategic. To counter this, we use the market mechanism. The strategic users are forced to truthfully reveal the value of their tasks and the system can maintain the same high level of utility as when all users were obedient.

5. INTEGRATED MECHANISM AND SYSTEM DESIGN

Mechanism design is traditionally part of economics while system design is part of computer science. Why should they be done in concert? How much benefit would be provided by adding an existing mechanism, such as bartering or EBay, to a separately designed system?

A long-standing principle [Levin et al. 1975] in system design is to separate policy and the computational mechanism used to implement the policy (not the economic mechanism in the mechanism design sense). However, as Clark, et al. [Clark et al. 2002] point out, policy and computational mechanism cannot truly be separated because the mechanism defines what policies are possible.

This would not be a problem if computational mechanism designers provided interfaces for efficient and scalable policies, but this has not been the case. For example, the market from § 2 requires statistics on resource usage (e.g., CPU cycles, memory pages, disk blocks) and dynamic control over allocation. However, many systems do not export detailed information on usage or allow dynamic control of allocations [Waldspurger 2002]. Another example is that several computational mechanisms assume a bartering policy. However, bartering economies have very little fluidity. It is difficult to find a mutually satisfying partner for each transaction and the complexity of determining the exchange rates of n resources is $O(n^2)$.

Even using an efficient economic mechanism from other contexts can result in poor efficiency in a computational environment. Unlike many other resources, the latency to access a computational resource is critical because changes in demand are unpredictable. For example, one possible economic mechanism for computational resources is to auction them on EBay. Auctions on EBay are tuned for human bidding so they typically take hours to close. The problem is that a web server's demand may be spiking right now. By the time the auction closes, the high load will have dissipated. The web server's operator could try to anticipate load and purchase capacity in advance, but this results in unused capacity.

In general, a pure mechanism designer is likely to design an economic mechanism with high economic efficiency, but with little concern for traditional systems metrics like computational efficiency, reliability, security, complexity, and ease-of-use. Pure systems designers have generally done the inverse. This is a direct consequence of a strict interpretation of the policy/mechanism separation principle. Instead, we advocate that systems designers embrace mechanism design as a first-order concern to eventually produce systems than can be both economically and computationally efficient.

6. ACKNOWLEDGMENTS

This paper benefited greatly from discussions with Leslie Fine, Bernardo Huberman, Adonis Houndroulis, Petros Maniatis, Lars Rasmusson, and Li Zhang. The editor, Mema Roussopoulos, was instrumental in clarifying the text.

REFERENCES

Adar, E. and Huberman, B. A. 2000. Free Riding on Gnutella. *First Monday 5,* 10 (October).

Chun, B. N. and Culler, D. E. 2000. Market-based Proportional Resource Sharing for Clusters. Technical Report CSD-1092, University of California at Berkeley, Computer Science Division. January.

Clark, D. D., Wroclawski, J., Sollins, K. R., and Braden, R. 2002. Tussle in Cyberspace: Defining Tomorrow's Internet. In *ACM SIGCOMM*.

Coleman, K., Norris, J., Candea, G., and Fox, A. 2004. OnCall: Defeating Spikes With a Free-Market Application Cluster. In *Proceedings of the IEEE Conference on Autonomic Computing*.

Duda, K. J. and Cheriton, D. R. 1999. Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose schedular. In *Symposium on Operating Systems Principles*. 261–276.

Ferguson, D., Yemimi, Y., and Nikolaou, C. 1988. Microeconomic Algorithms for Load Balancing in Distributed Computer Systems. In *International Conference on Distributed Computer Systems*. 491–499.

Hardin, G. 1968. The Tragedy of the Commons. *Science 162*, 1243–1248.

Levin, R., Cohen, E., Corwin, W., Pollack, F., and Wulf, W. 1975. Policy/Mechanism Separation in Hydra. In *ACM Symposium on Operating Systems Principles*.

Malone, T. W., Fikes, R. E., Grant, K. R., and Howard, M. T. 1988. Enterprise: A Market-like Task Scheduler for Distributed Computing Environments. In *The Ecology of Computation*, B. A. Huberman, Ed. Number 2 in Studies in Computer Science and Artificial Intelligence. Elsevier Science Publishers B.V., 177–205.

Ng, C., Parkes, D., and Seltzer, M. 2003. Strategyproof Computing: Systems Infrastructures for Self-Interested Parties. In *Workshop on Economics of Peer-to-Peer Systems*.

Peterson, L., Anderson, T., Culler, D., , and Roscoe, T. 2002. Blueprint for Introducing Disruptive Technology into the Internet. In *First Workshop on Hot Topics in Networking*.

Regev, O. and Nisan, N. 1998. The Popcorn Market: Online Markets for Computational Resources. In *Proceedings of 1st International Conference on Information and Computation Economies*. 148–157.

Stoica, I., Abdel-Wahab, H., and Jeffay, K. 1997. On the Duality between Resource Reservation and Proportional Share Resource Allocation. In *Multimedia Computing and Networking*. SPIE Proceedings Series, vol. 3020. 207–214.

Stratford, N. and Mortier, R. 1999. An Economic Approach to Adaptive Resource Management. In *Workshop on Hot Topics in Operating Systems*. 142–147.

Sullivan, D. G. and Seltzer, M. I. 2000. Isolation with Flexibility: a Resource Management Framework for Central Servers. In *Proceedings of the USENIX Annual Technical Conference*. 337–350.

Waldspurger, C. A. 2002. Memory Resource Management in VMware ESX Server. In *Proceedings of the Symposium on Operating Systems Design and Implementation*.

Waldspurger, C. A., Hogg, T., Huberman, B. A., Kephart, J. O., and Stornetta, W. S. 1992. Spawn: A Distributed Computational Economy. *Software Engineering 18,* 2, 103–117.