

Spawn: A Distributed Computation Economy

Waldspurger, Hogg, Huberman,
Kephart, and Stornetta

Presented by Jeff Pasternack

Motivation

- Want to run distributed applications over large, heterogeneous network, taking advantage of idle time.
- Many users and many distributed applications mean competition for resources.
- Need an efficient, effective and fair way to allocate resources based upon application priority.

Spawn Overview

- Spawn is essentially a microeconomic approach to scheduling distributed applications.
- Each application has a steady stream of funding (e.g. \$0.03/sec).
- A user divides funding given by system administrators (or earned by his machine selling time slices) among his applications.

Spawn Overview

- The amount of funding an application is given corresponds to its priority
- Ideally, the ratio of resources granted to an application matches the ratio of its funding to total funding.

Spawn Overview

- Applications are buyers, machines with idle time are sellers.
- Prices, determined by auctions, allow fair valuation among heterogeneous sellers.

Contributions

- “The first implementation of a distributed economy”
- Money as priority for distributed applications in a heterogeneous environment.
- “The first examination of the price dynamics of a computational economy”
 - i.e. they ran some experiments

System Overview

- Buyers want to run their distributed applications
- Sellers want to sell unused machine resources.
- Price setting via auctions.

Auction Mechanism

- Each selling machine employs an auction process to accept bids and announce results.
 - Not addressed in paper: the fairness of this auction process competing against the task the machine has been hired to run.

Auction Mechanism

- The auction is for the next available slice of time on the machine.
- Time slices are variable duration.
- Bids therefore consist of the length of time slice sought, the amount of money offered, and a task description.

Auction Mechanism

- Time slices are required by the seller to fall between a minimum and maximum duration.
- If a time slice were too short, overhead would be too high.
- If a time slice were too long, a machine would not be able to quickly return control if its primary user needed to use it.

Auction Mechanism

- An auction process may favor a longer or shorter time slice with a discount (or discourage it with a penalty).
- The authors do not explore this, but rather use a simple linear relationship between duration and cost.

Auction Mechanism

- Auctions are sealed-bid, second-price.
- Sealed-bid means that bidders do not know what others are bidding.
- Second-price means that the price the time slice is sold for is always the amount of the second-highest bid.

Auction Mechanism

- This is intended to provide an incentive for bidders to bid what the resource is really worth to them.
- “English” first price auctions yielded similar results, but higher overhead.
- This is not surprising.

Auction Mechanism

- Only just before the machine is about to become free does it choose which bid to accept.
- It then notifies the winner and all of the losers of the results.

Resource Managers

- Resource managers serve two purposes:
 - For sellers, starting and monitoring the task executing in the current time slice.
 - For buyers (applications), acting as the Spawn API, providing a high-level interface that allows the buyer to make funding and task-spawning decisions while hiding the underlying auction mechanism.

Right of First Refusal

- The resource manager normally aborts application processes that run over their allotted time.
- However, an application has the option to instead buy “extension” time slices at the going market price.

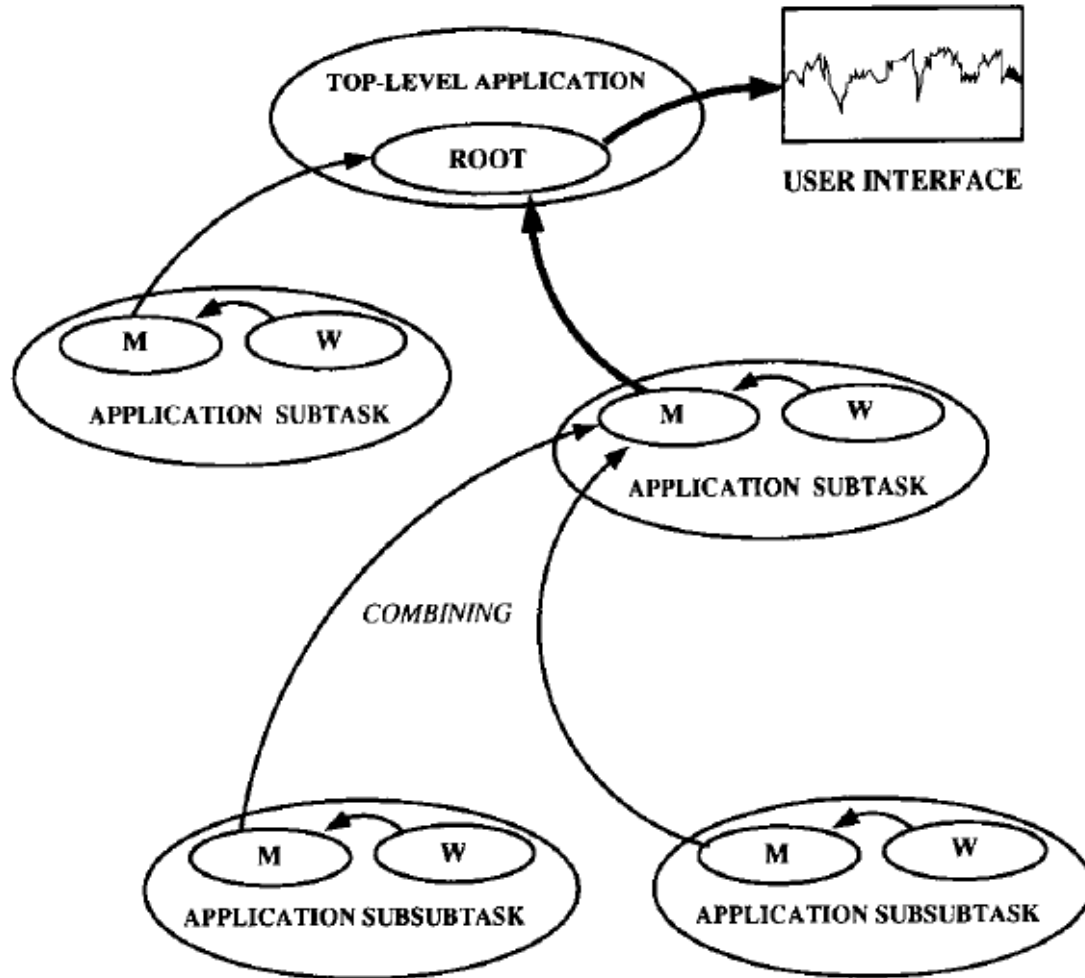
Right of First Refusal

- This extension capability is important because, in Spawn, processes cannot be persisted/serialized or restarted.

Application Processes

- Workers and managers
- Managers can employ submanagers.
- Managers collect subordinate information and report to superiors.
- Top level application has root manager (running on user's machine) that collects the results and presents them in the UI.

Application Processes



Spawning

- Managers can spawn new subordinates.
- When a manager asks the resource manager to spawn a new worker, it gives it a task name and a list of arguments.

Task Files

- The task name tells the resource manager which task file to use.
- This specifies, for every machine configuration on the network, which binary for that machine corresponds with the task
- It also gives the relative efficiency of that machine for the task.

Task Files

- Relative efficiency for a task (e.g. “multiplying matrices takes Y seconds on machine configuration X ”) determines how the resource manager will bid when trying to find a time slice.
- Notice that the need for a task file, which must accurately predict the time required for every machine configuration, is an important weakness of Spawn.

Spawning Managers

- Spawning managers to handle subtasks is like spawning workers, except additional information needed to govern the manager's behavior is passed along:
 - Tradeoff between time and money
 - Expected run-time for the task
 - User and group IDs which prevent managers belonging to the same user or application from competing in auctions.

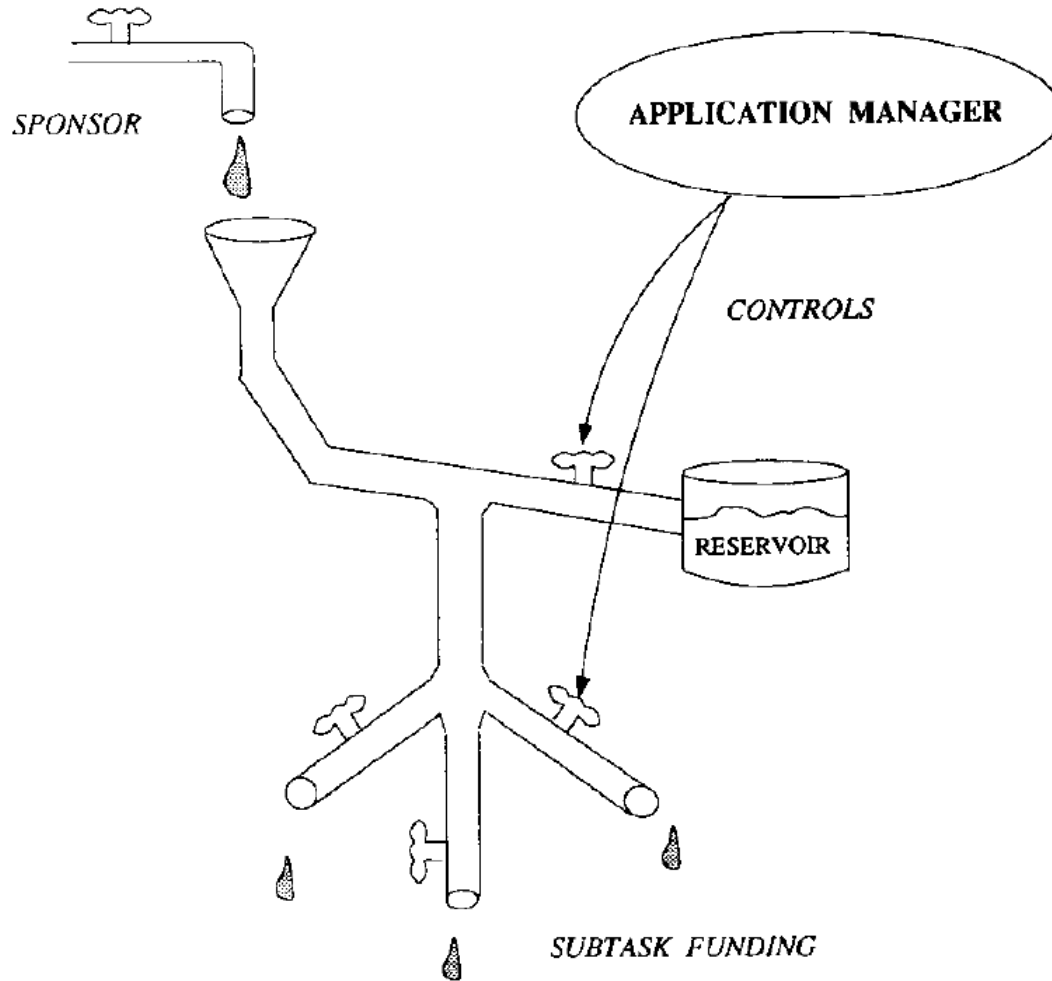
Spawning Managers

- When a manager's time slice expires, it can either:
 - Extend its time slice
 - Die and delegate its children to another manager, usually its superior.

Sponsors and Funding

- Managers get a stream of funds from their superior; the root manager gets funding from the user.
- Managers are free to decide how they allocate their funding among their subordinates; they can also store funding in a reservoir.
- Money can neither be created nor destroyed.

Sponsors and Funding



Sponsors and Funding

- A simple managerial strategy is to simply give every subordinate an equal share.
- Typically, one can do better by giving more funding to more cost-efficient or more “interesting” subordinates.

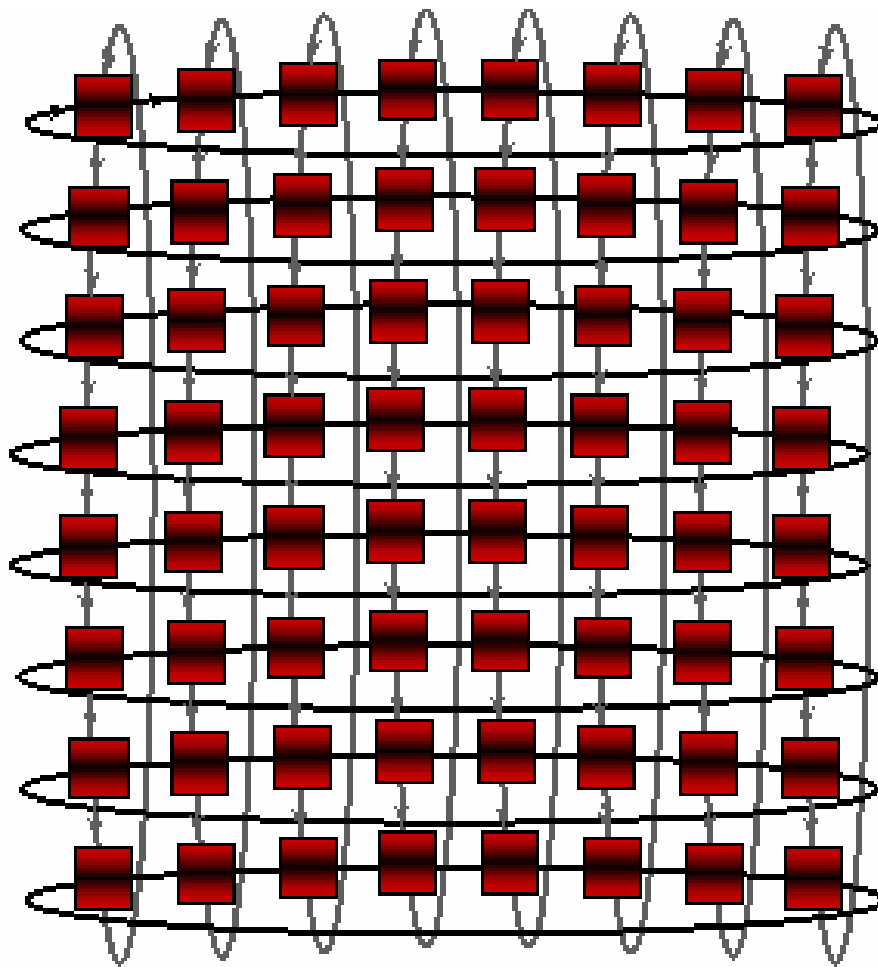
Spawn Implementation

- Written in C
- Run over a network of Sun UNIX machines
- Runs in user-mode; process creation expensive
- Created and tested in 1992, back when computers were still steam-powered.

Network Topologies

- Ideally, every buyer can participate in every auction.
- Infeasible in large network.
- Authors use torus topology in larger networks.

Torus Topology



Experiment: Basic Efficiency

- Basic efficiency: run a distributed application with Spawn in an otherwise completely idle network
- Compare with optimal execution, running constituent tasks serially on the machines with no Spawn code.
- 10.3% overhead for 60-second time slices; 7.6% for 120-second slices.

Experiment: Basic Efficiency

- Unfortunately, 60-second time slices are probably too long.
- A user returning to his computer would have to wait an expected thirty seconds before using it again.
- Time slices of ten seconds or so would be more palatable, but much less efficient.

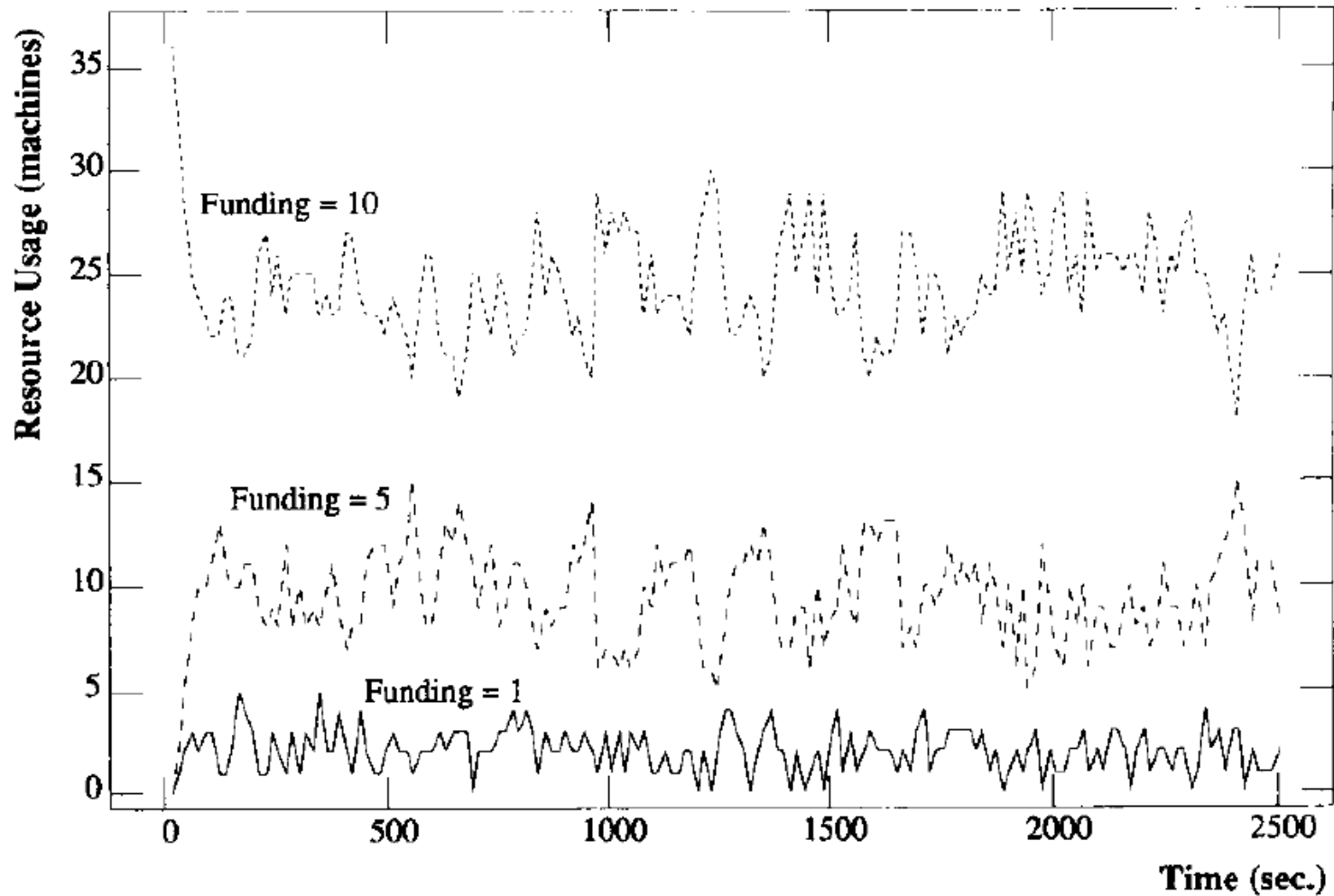
Experiments: Fairness

- Recall that we want the ratio of funding and the ratio of time allocated to match (assuming homogenous tasks).

<i>Funding Ratio</i>	<i>Time Allocation</i>		
	<i>16 Node Torus</i>	<i>36 Node Torus</i>	<i>64 Node Torus</i>
1:1:1	1.15:1.06:1	1.06:1.04:1	1.08:1.04:1
3:2:1	2.89:1.84:1	2.89:1.75:1	2.95:1.77:1
10:5:1	9.46:4.22:1	12.11:4.68:1	8.18:3.34:1

<i>6 Nodes, Fully Connected</i>	
<i>Funding Ratio</i>	<i>Time Allocation</i>
1:1	1.04:1
2:1	1.85:1
10:1	12.36:1
3:2:1	2.79:2.00:1

Experiments: Fairness



Experiments: Fairness

- Large, less-connected networks tended to exhibit more uneven pricing across homogenous machines.
- Authors concluded that this stemmed from groups of wealthy (usually high-level) managers that shared neighbors, and groups of poor managers that shared neighbors.

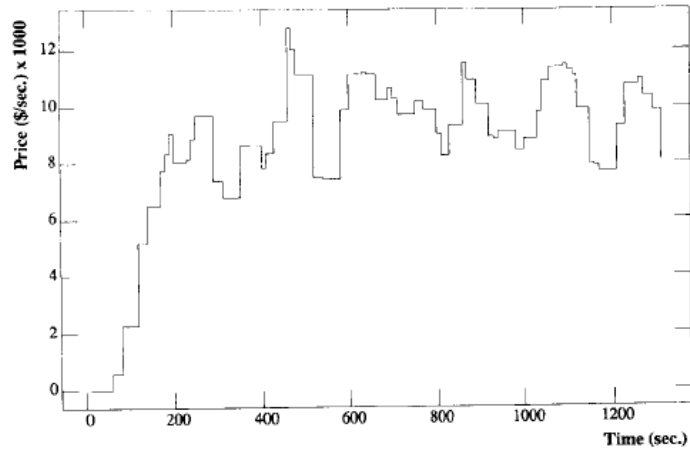
Experiments: Fairness

- Can remedy problem by smarter fund allocation.
- If managers allocate by cost-effectiveness:
 - Poor subordinates near cheap machines receive more money
 - Other subordinates receive less
- If every manager does this, prices soon equalize.

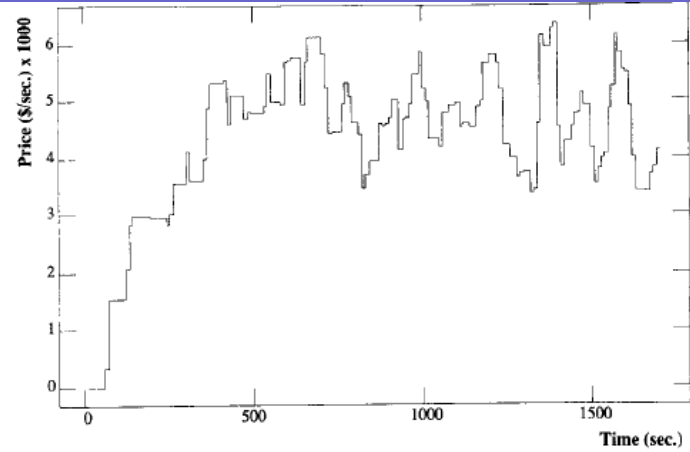
Experiments: Fairness

- An alternative to auctions is direct allocation.
- Every task seeking to run on a seller runs at once, time multiplexed, given a proportion of time equal to its proportion of funding.
- Same outcome as auction, with possibly less overhead.

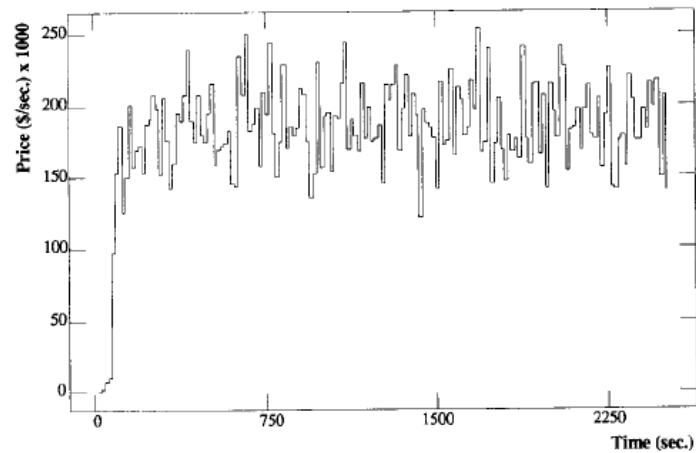
Experiments: Pricing



(a)



(b)



(c)

Experiments: Pricing

- Observe that prices oscillate significantly and frequently.
- This is due to the simple strategies of the buying agents (e.g. escalator algorithm).
- Smarter agents would mean less volatility but greater overhead.

Spawn Strengths

- A way to prioritize applications to run over a well-understood network of otherwise idle machines.
- Overhead not excessive.
- More-or-less fair on average.

Spawn Questions

- Performance with heterogeneous tasks?
- What if multiple resources are sought (e.g. both machine time and bandwidth)?
 - Requires more intelligent buyers and sellers
- What about larger sparse networks?
 - What is the trade off between connectivity and fairness?

Spawn Weaknesses

- Spawn assumes a completely trusted set of users and machines.
 - Cheating the system is trivial
 - A cheater can quickly bring all other applications to a standstill.
- Spawn assumes well-know machine configurations and tasks
 - Specifically, need to know how long each task takes on each configuration

Spawn Weaknesses

- Decentralized auction mechanism does not scale well
 - Need central authority?
- Users cannot choose their buying/selling strategies; must accept Spawn's simplistic choices.
 - Would be unacceptable if real money were involved.

Additional Experiments

- Price differentials in heterogeneous systems: a Sun4/260 runs an application's tasks 1.4 times as quickly as as Sun4/110.
- Network: 3 260's, 6 110's.
- Prices at equilibrium differ by less than a factor of 20%
- \$0.0032/s for 110's, \$0.0053/s for 260's
- Given \$0.0032/s for 110's, 260's *should be* \$0.00448/s.

Additional Experiments

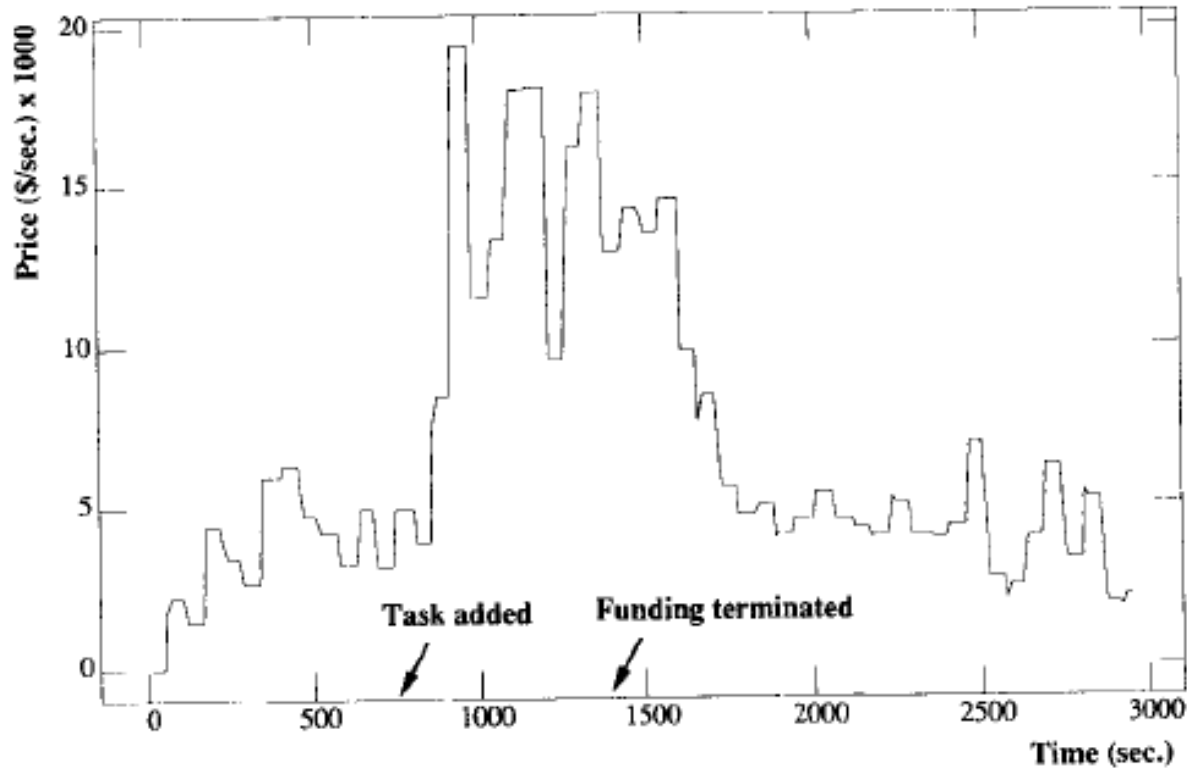


Fig. 10. **Market Response to a High Priority Task.** Average price as a function of time when a high priority task is injected into a 6-node, fully connected system with low-priority jobs. The introduction and termination of the high-priority task's funding are indicated on the time axis.