



Market-Based Resource Allocation for Utility Data Centers

Andrew Bye, Mathias Sallé, Claudio Bartolini
HP Laboratories Bristol
HPL-2003-188
September 9th, 2003*

market,
markets,
outsourcing,
UDC,
data center,
agents,
resource
allocation

This technical report concerns the application of market-based control to dynamic resource allocation within a utility data center or other distributed computing infrastructure. The report is intended to do two things. First, it demonstrates a possible instantiation of the market paradigm in an outsourced-services context, describing a suitable market mechanism and exemplifying agent bidding algorithms; second, it provides evidence in the form of simulation results to support the argument that dynamic resource allocation in general, and market-based resource allocation in particular, can contribute measurable value to business.

Market-Based Resource Allocation for Utility Data Centers

Andrew Bye, Mathias Sallé, Claudio Bartolini
HP Labs

September 3, 2003

Abstract

This technical report concerns the application of market based control to dynamic resource allocation within a utility data center or other distributed computing infrastructure. The report is intended to do two things. First, it demonstrates a possible instantiation of the market paradigm in an outsourced-services context, describing a suitable market mechanism and exemplifying agent bidding algorithms; second, it provides evidence in the form of simulation results to support the argument that dynamic resource allocation in general, and market-based resource allocation in particular, can contribute measurable value to business.

1 Introduction

In recent times, IT managers have become more demanding of the infrastructure they use to support business processes. They expect it to be reusable, flexible, fault tolerant, and to admit dynamic reallocation between tasks. On top of this, they want to be able to quantify return on their IT investment. Several large IT providers, such as HP and IBM are articulating a similar vision of computing “on demand”, supported by internet data centers, which addresses the first four issues, but only begins to address the all-important fifth. This issue is not just about having a stable, high performance IT layer; it’s about providing a credible, accountable linkage between business objectives and IT resource allocation.

In this paper we examine a decision-layer for the dynamic reallocation of resources between competing tasks so as to demonstrate the optimisation of business value: we describe a market-based dynamic resource allocation mechanism and associated reasoning components that adaptively manage the allocation of resources within a data center such as HP’s Utility Data Center (UDC) [6].

The business context we assume is one in which the owner of a UDC makes money by hosting outsourced services such as payroll, account management, web-services, etc. The UDC owner writes Service Level Agreements (SLAs) with its customers, for whom it is running the outsourced services, specifying service levels over mutually agreed metrics – e.g. throughput and latency on page requests to a hosted web-service – and payment terms contingent on these service levels. The UDC owner must choose whether to agree to host a service with a given SLA, and must manage the deployment of its IT resources to those services it has chosen to host.

The value of dynamic resource allocation in this context is that under-utilised resources can be redeployed wherever they are most needed. Thus, a service that is experiencing low load can relinquish resources (e.g. bandwidth or CPU cycles) to another service that is experiencing high load. The value of *market-based* resource allocation in this context is that the re-allocations that are implemented best reflect the business value of the UDC operator, because an agent’s ability to pay is determined by income to the UDC owner from its SLA.

The high-level components of our market-based resource allocation system are depicted in Figure 1. The most important are:

- A market for resources within the data center. The market aggregates and processes bids from agents, releases resources to agents that are successful in bidding for them, and has the power to remove resources from agents if necessary. See Section 3 for more details.
- Agents that manage the provisioning of applications. The functionality of an agent is broken down into an application-level component, and a business-level component, which interact through service level specifications and predictions of utility. See Section 4 for more details.

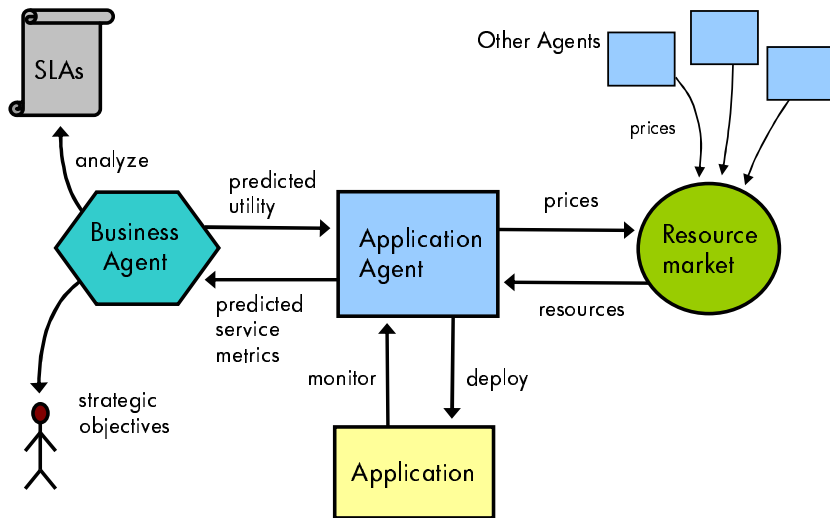


Figure 1: Diagram of the Market-Based Resource Allocation system architecture

The remainder of the paper is arranged as follows. After briefly reviewing previous work in this area in Section 2, high-level discussions of markets and agents follow in Section 3 and Section 4 respectively. In Section 5 we describe an example scenario, specifying the market mechanism, service type, and agent algorithms; in Section 6 we describe the results of simulations experiments that demonstrate the potential for market-based resource allocation, and conclude in Section 7.

2 Related Work

One approach to resource allocation between customers of a data center is to avoid the question of placing value on allocations altogether. The infrastructure operator simply provides resources on demand, so long as those demands conform to previously agreed profiles (see for example [10]). The infrastructure operator’s decision problem is one of admission control. This is a valuable framework because of its generality. It allows a great deal of independence between the analysis of what resources are needed, and the decision about whether those resources are available. This separation between resource availability and resource requirements planning is also the problem with such an approach: a collection of resource planners will each ask for more resources than are strictly speaking necessary so as to manage their individual risks, whereas those risks could be more effectively managed in aggregate. Such an approach leads to low resource

utilisation.

Other work treats the case of optimally assigning specific infrastructure components to a single application [11, 13], assumes that allocation values are externally imposed [7] or seeks to optimise measures, such as “fairness” that are not business-driven [4].

For example, [4] addresses the issue of balancing the needs of competing services within a utility computing environment. As in our work, applications are modelled so as to extract performance expectations from possible resource allocations. However, their optimisation is based on treating each request to each service as having identical value. Our work focuses on maximising business value, primarily via optimisation of return on SLAs.

Liu et al. [8] consider the provisioning of resources to maximise return on SLAs, and in this respect share our goal of driving resource allocation decisions according to business need. However, their application model is quite specific, and it is difficult to see how their optimisation approach would extend to more diverse application types, such as the type presented in our example (see Section 5).

Menascé et al. [9] also focus on business value, but address the resource allocation problem within a service, not between services, via the adjustment of priorities. This work is similar in focus, if not in method, to [14], which also looks at differentially varying the quality of service to different customer classes. In our framework such functionality resides within the application agent; our equivalent

of their optimisation algorithms is the analysis in Section 5.6.

Markets have been used to resolve resource allocation problems in a variety of domains for thousands of years. More recently some researchers have found success in developing market-based resource allocation in computer systems [3, 2, 12]. Our work is similar to the MUSE system described in [2], in as much as we balance demand between competing services by a market mechanism, in order to optimise return on service level agreements. The system we describe allows trading in specific resources not just generic resources (such as “CPU time”), and thus supports the deployment of complex applications with specific resource requirements. We also avoid the security and performance isolation problems of shared servers by assuming that each server hosts at most one application at any time.

3 Market Mechanism

The term “Market-based” is a broad one, which can be understood to encompass institutions as diverse as the New York Stock Exchange, Christie’s, Safeway, or the “market-place of ideas” [5]. At its most basic, a market is a venue for comparing the willingness of several agents to pay for various resources in a common monetary unit, and for determining, as a result, if and how goods should change hands. A mechanism will specify the sorts of information that agents can receive and messages they can send, and how messages will be resolved into an allocation.

Central to the market-based approach is the idea that different agents have different knowledge, goals and preferences, but nonetheless must express their desires in a common currency. This paradigm is suitable for the management of heterogeneous services within a UDC because different applications are provisioned in different ways. The knowledge of how to relate resource allocations to application performance and hence, via service metrics, to business value, is application and business-specific; it is natural therefore to consider a paradigm in which the concerns of different services are separated. On the other hand when two agents both want a resource, for reasons that are determined in complex ways from the operational behaviour of the applications they manage, there needs to be a rational way to resolve the issue of who gets the resource. In the market-based ap-

proach this resolution proceeds via bidding: whoever bids the highest gets the resource, and bidding is constrained, via budgets, by appealing to the business value derived from different allocations.

Markets are at their most useful when truthful preference information is difficult or impossible to extract - for example when the various parties are unwilling to reveal such information directly. This is not the case in the outsourced services context, because the agents are owned and operated by the UDC owner, on its behalf, and are thus essentially cooperative. Nonetheless we believe that the market-based approach is useful for its intrinsic separation of concerns and focus on business value.

Our choice of context motivates some of the following mechanism choices (see Section 7 for a discussion of future work to extend and generalise these assumptions):

1. Servers or other computing devices are the essential resources.

Different choices of the level of abstraction in the resource description give rise to different challenges and opportunities. The advantage of organising a market for more abstract resources is that the agents’ decision problems are easier, and the market protocol is cheaper to compute. The advantage of organising a market for more specific resources is that more allocation features, constraints and optimisation problems can be subsumed within the pricing function.

For example, if each “resource” that can be traded is a specific device within the UDC, then imposing a cost on an agent for its application’s use of internal network bandwidth will give rise to different prices for different identical servers depending on where they are located. This, in turn, will lead to application placements that minimise use of internal bandwidth – a generally desirable outcome. On the other hand, if resources are only available within the market at a more abstract level, with no information about location being visible, then agents need not reason about traffic levels between servers, and placement of specific application components onto specific devices can be delegated to some other algorithm (see, e.g. [11]).

2. Resources are *rented* from the UDC in time periods: at the end of each time period an agent retains no rights over its resources.

The choice of a rental model is designed to ensure that efficient allocations are not avoided by agents retaining resources that others need more.

3. The market is centralised: The UDC proposes trades, swaps and other adjustments of the current allocation, and solicits bids for the given trade to go ahead.

This protocol was chosen because a necessary condition for a market to function properly is *liquidity*: there must be sufficiently many bids that trades can occur and that prices reflect real need. In order to ensure liquidity we force agents into a mostly reactive role: their primary function is to reply to requests for quotes. Thus in a given bidding round, the market polls agents for their willingness to pay for certain reallocations. At a minimum these bidding rounds should occur every short while¹ in order to ensure that allocations are up-to-date with changing willingness to pay. Ideally agents should be provided with the opportunity to request extra rounds of trading, so that they can attempt to buy resources when needed.

The choice of *which* trades the UDC should request bids for, is a design decision that will trade off the messaging complexity of extracting information about different reassignments with the potential increase in utility to be gained from exploring a wider space of reallocations.²

A combinatorial auction is one extreme of this spectrum, in which the UDC extracts a bid for each subset of resources from each agent, and decides which resources to allocate on this basis. In general such an auction is computationally infeasible due to the large number of sets of resources. If the servers are classified into types, and agents not offered specific servers, but instead types of servers in variable quantities, it may be that the complexity of such an auction can be brought down to acceptable levels [7].

4. It is not necessary to collect payments for resources.

In the context we consider, agents are acting cooperatively, and the only effect of accounting for internal currency is to enforce budget limits. This might be useful as an automatic control on buggy software (e.g. to limit the damage caused by a mis-coded application agent that bids one hundred times higher

¹Quite what this means in practice will depend on the mix of applications, the rate at which the load on those applications changes, the time scale of the SLAs with respect to which allocations are intrinsically being optimised, and on pragmatic considerations such as the length of time it takes to reboot a server or bring a database on line.

²See the end of Section 6.6 for the surprising results of a comparative study of this trade-off for an example scenario.

than it should) but serves no rational purpose otherwise: if an agent says it is willing to pay a certain amount for a resource, if the agent is trusted to be telling the truth, and if the agent's goal is to maximise return to the UDC operator, the UDC operator would be foolish to block the trade because of an internal budget limit. Clearly accounting is necessary in untrusted environments, and is, in any case, a useful metric of a given outsourced service's value to the UDC operator.

4 Agents

The participants in the market are agents – they bid for resources to provision applications. The functionality of an agent is broken down, as in Figure 1 into a part related to the prediction of service-level metrics – the application agent – and a part concerned with linking these service levels to business objectives – the business agent.

4.1 Application Agent

The application agent

- associates utility values to given resource collections by predicting service metrics, and querying the business agent for the consequent business value;
- bids for resources accordingly; and
- triggers deployment of successfully acquired resources. The agent is also responsible for relinquishing control of resources that it is using, but has failed to acquire for the next time period.

The application agent provides the essential “glue” between the business level and the IT level: In order to interface with the resource market it must understand resource descriptions; in order to interface with the business agent it must understand service metric specifications. To work out what to bid to acquire a given resource, the application agent speculates, via the business agent, about the return on its service contracts in the foreseeable future if it had the resource, and its return if it didn't. The difference between these represents the rational maximal willingness to pay.

Predicting service performance metrics for a given collection of resources requires the application agent to understand different deployment options – i.e. how specific application components

can be mapped onto the resources it may have. Therefore, the form of the application agent will depend most critically on the application to be provisioned, or at least on the application type (e.g. three-tiered or workflow), and as such little can be said in general about the algorithms upon which its decision-making component will be based.

4.2 Long-term versus Short-term planning

A vital design decision open to the UDC operator is whether the service-metrics to be evaluated by the business agent are long-term or short term. Should the application agent seek to predict the *average* performance level of the application, perhaps taking job arrival rates and other such statistical data into consideration; or should it concern itself with the service levels that can be predicted for those jobs that are already being processed?

This design decision depends on the nature of the application being managed, the level of dynamism of the underlying infrastructure, specifics of the market mechanism, and the level of competition for resources. If the infrastructure or market mechanism operates slowly relative to the average processing time of a job, then bringing resources on-line for long-term performance makes sense. This is because if the agents attempts to do “just-in-time” provisioning, it may not be possible to acquire, boot and configure new resources for a processing node fast enough to save jobs that are in jeopardy of failing to meet their deadlines.

In the simulations we run in Section 5 we have no such infrastructure constraints, and so we focus on the extremely dynamic case, in which the time it takes to re-configure servers that have been moved to a different application is small by comparison with the length of a typical round of bidding, which is itself small compared to the average duration of a job within the system. This corresponds to a scenario in which (for example) an application component can be moved from one server to another in a matter of minutes, while bidding rounds occur every hour, and jobs usually take about a day.

A related issue is that of planning for sudden spikes in demand. The application agent should speculate about the possibility that demand will increase dramatically in less time than it takes to request and warm up resources. In general, we expect that the more sophisticated is the analysis that the application agent performs, the better the performance of the agent’s application, and thus

the market-based resource allocation system as a whole. Much therefore rides on getting the service-metric prediction algorithms to function properly.

In Section 5.6 we describe the application agent reasoning algorithm for a specific example scenario in greater detail.

4.3 Business Agent

The business agent analyses the business engagements and objectives of the enterprise with regard to the application, in order to associate business value to forecasted service metrics. The analysis necessary for this can be separated into two strands, SLA-derived value, and strategic value.

4.3.1 Value derived from SLAs

The most direct way to account for the business value of a service is via SLAs between the enterprise and its customers specifying, for the service in question,

- service metrics (e.g. number of transactions per second, total time to delivery, system up-time, etc.);
- constraints over the values of these service metrics (e.g. total time to delivery less than 3 days, system up-time greater than 99% in each month, etc), known as Service Level Objectives (SLOs); and
- consequences (e.g. rewards or penalties) of meeting or violating these SLOs. These consequences could depend directly on service metric values (e.g. a payment that is proportional to the number of items delivered, above some minimum).

4.3.2 Strategic Value

In any utility-based system, utility modelling is the most critical piece. SLAs are an excellent way to specify the value that an enterprise extracts from its customers, and the fact that they are formal makes it easier to extract preferences from them for use in an automated management system such as the one we propose. Therefore, whenever possible, we prefer to measure business value in such terms. However, the enterprise will often have strategic objectives that it prefers not to capture as explicit service level agreements. An example of such strategic value is the value an enterprise gains from reputation: an

enterprise might strive to maintain higher service levels than are rational so as to support a reputation for excellence that defines its market niche. In some sense these objectives can be seen as implicit service level agreements; in any case we should attempt to take them into account when making provisioning decisions, but if they are not expressible in terms of service metrics, then no decision system (except “intuition”) will be able to do so.

5 Simulations

In this section we describe a simulation of a market-based resource allocation system; in Section 6 we report the results of running a selection of example scenarios in this simulated system.

In order to specify the example scenarios we must specify

- the market mechanism;
- the application-level structure of the outsourced service, and the algorithms the application agent uses to predict service metrics from resource allocations; and
- the service metrics over which the SLAs are specified, and example SLAs.

5.1 Market Mechanism

Collecting together the mechanism choices described in Section 3, we implemented the following two market mechanisms, the first of which was used for most of the experiments that follow.

5.1.1 Simple Market

1. Find the bids of all agents for all servers; A server is *tradable* if the highest bid for it is not that of the server currently in possession of it; the *trade value* of a tradable server is the difference between these two bids.
2. If there is at least one tradable server, find the server whose trade value is greatest, and effect the trade (i.e. remove the server from its owner, and give it to the highest bidder).
3. If a server was traded then start again from step 1.

A bidding round only ends therefore when the resource market is in *equilibrium*, i.e. no trades can

occur. Optionally we can choose an upper bound for the number of times the market loops back to step 1. This might be a good idea either to bound the computation time for each bidding round, or in order to dampen allocation fluctuations. For example, it may be that the application agent’s performance metric estimates are less and less accurate the further proposed resource allocations are from the current allocation. This is the case in [2], where a crude linear model is adequate for estimating the effect of *small* allocation changes, but is not accurate for large changes.

In our experiments in Section 5, we always continued each bidding round until equilibrium was reached.

5.2 Globally Optimal Assignment Market

1. Get the bids $bid_a(n)$ of each agent a for each *number* of servers it may receive, n , from 0 up to the total number of servers available.
2. Construct the set of all assignments (n_1, \dots, n_A) of numbers of servers to agents; choose the assignment (n_1^*, \dots, n_A^*) for which total value is maximised:

$$value(n_1, \dots, n_A) = \sum_a bid_a(n_a)$$

3. Assign n_a^* servers to agent a , for each a .

This mechanism relies on the fact that servers are indistinguishable, which, in this section, they are.

These two mechanisms represent, to some extent, the extremes of rationality within our system. By comparing the performance of the simple market with the globally optimal assignment market we can determine just how sub-optimal are the allocations that the simple market finds.

5.3 Service Specification

We consider an abstract service, modelled as a linear workflow consisting of a chain of N processing nodes, each having a job queue; see Figure 2. Jobs arrive from end customers of the service, and must pass through each processing node en route to completion. Each job is identified as belonging to one of a small set of *service classes* – such as “Gold”, “Silver” and “Bronze” – which define acceptable total processing times. The time it takes a given processing node to process a given job is influenced

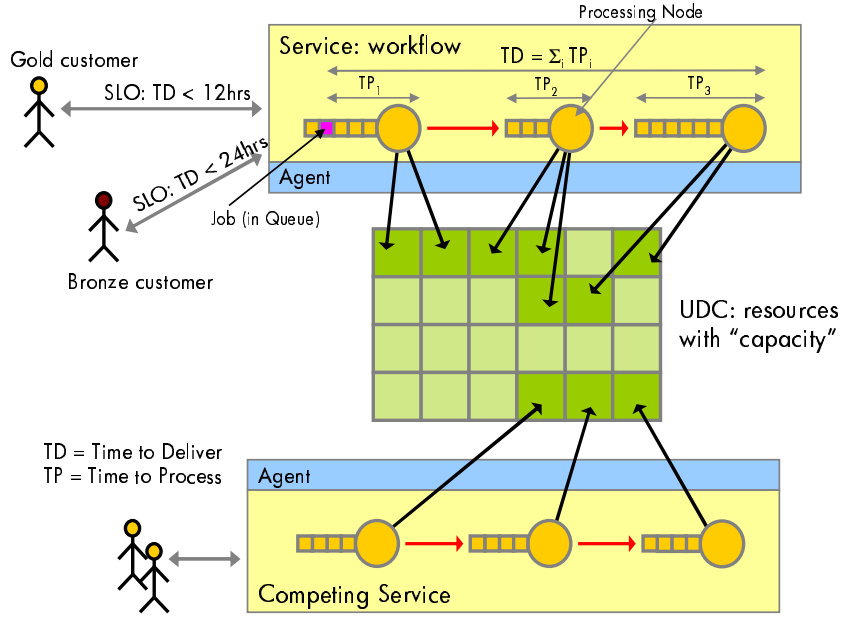


Figure 2: Diagram of the workflow application model

by the amount of computational resource the UDC operator commits to that processing node. Broadly speaking, the more resources committed to a node, the faster jobs move through it. More precisely, we assume that an intrinsic feature of each job is the amount of *work* that must be done in each processing node in order for the job to be completed. Likewise, servers in the UDC have an intrinsic *capacity* for doing such work, which is the amount of work they can do in a single time step. Let C_n be the sum of the capacities of all servers allocated to processing node n ; we use the following formula for the time taken for the node to do a unit of work:

$$(\text{time per unit work})_n = m_n + \frac{v_n}{C_n}. \quad (1)$$

Here m_n is an un-parallelisable component of the processing time that does not depend on the set of resources allocated to the node, whereas v_n represents the part that is parallelisable, and hence which will be affected by adding resources to the node.

Jobs arrive to be processed according to a Poisson arrival process, and have work loads for each processing node that are also given by an exponential distribution.

5.4 Service-Level Agreements

We assume that the UDC operator is rewarded for completing jobs on time, and pays penalties for delivering jobs late or not at all. The rewards and

penalties will typically be different for different service classes so as to incent the UDC operator to prioritise customers in higher-ranked service classes, but in this paper we will not investigate service classes, and restrict our attention to a single service class. We consider a scheme in which the reward for delivering a job at time t is given by a function $f_{\text{reward}}(\text{work})$ of the total work in the job, multiplied by a factor $f_{\text{discount}}(t)$ depending on the time t at which the job is delivered. The function f_{reward} captures the reward for doing a job on time. It may be constant across jobs, or may be skewed to value bigger jobs more. In the simulations of Section 6, this function was taken to be simply 1. The function f_{discount} captures the dependence of payments on time; an example of such a function is shown in Figure 3

We take a short-term view of provisioning, so that the service metrics of interest will be the predicted times to delivery of all jobs currently active in the workflow.

5.5 Business Agent Algorithm

The business agent must convert job service time estimates from the application agent into utility specifications using these SLAs. One way to do this would be simply to add up the rewards and penalties, as given by the SLA, for all jobs. However,

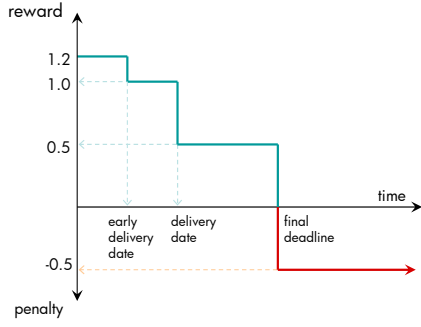


Figure 3: Example time-dependent payment scaling function. The UDC operator earns a certain job-dependent reward if the job is delivered by the *delivery date*; if it is delivered on or before the *early delivery date* it yields a 20% bonus; if it is only delivered before the *final deadline* it yields 50% of the reward amount; if not delivered by the final deadline, the UDC operator is fined 50% of this amount.

when deadlines are long relative to the duration of a time slice (as we have assumed they are), this tends to *over-value* each job: the value of the whole job is added to each time step during which the job is being processed so that, depending on the market mechanism, the agent might end up paying for the same job multiple times. When all other agents are using the same algorithm under the same conditions, this sort of scaling problem is hidden. But in a heterogeneous system, all agents must scale their utility values so that the impact of having a particular resource for a particular time slice is fairly valued. In particular, we would like the average amount bid for the resources that the agent actually received, over the lifetime of a service, to be as close as possible to the actual revenue generated by that service.

For this application we accomplish this by spreading the predicted utility of all jobs in the queue out over the period of time that is predicted for the last job to exit the queue – the longest service time. Under approximately steady arrival and work rates, this duration should be approximately the duration of a job in the system, so that the value of a job is spread over its lifetime, and counted once.

5.6 Application Agent Algorithm

The task of predicting service times for all jobs given particular resources can be tackled at various levels of complexity. At the most complex level, the entire workflow can itself be simulated *within the*

*agent*³ to determine accurate delivery time predictions for each job. However, in a sense this is being too kind to the agent: within the simulated experimental environment, the model the agent has of the workflow can be exactly the same as the model with respect to which it is being tested; in the real world applications and models of applications can behave quite differently. In most of the following experiments we use a cruder model to investigate how sensitive the agent’s performance is to model quality, and compare the performance of this model with perfect knowledge of completion times in Section 6.6. Thus we use a simplified model of processing time that is easy to compute, and, as we will see in Section 6 gives good performance.

The agent maintains an estimate of the average amount of work per job for each processing node, $(av\ work)_n$, and an estimate $(av\ length)_n$ of the queue length at each node. The estimate of queue length is updated at the beginning of each bidding round, according to the following formula:

$$\lambda = \max(\lambda_{min}, \frac{1}{n+1})$$

$$(av\ length)_n := \lambda \times (Q\ length)_n + (1 - \lambda) \times (av\ length)_n$$

This update rule smoothes out the observations of actual queue length over multiple time steps, but ensures that $(av\ length)$ is kept up-to-date by placing a lower limit of λ_{min} on the weight given to new observations is not less than λ_{min} . A higher value of λ_{min} leads to a value of $(av\ length)$ that is more responsive to changes in the load on the application, but is more volatile⁴.

The following formula is used to estimate the time at which job j , currently either being processed or waiting to be processed at node n , will finally be delivered:

$$\frac{(Q\ position)_j \times (av\ work)_n + work_{n,j}}{(time\ per\ unit\ work)_n} + \sum_{m>n} \frac{(av\ length)_m \times (av\ work)_m + work_{m,j}}{(time\ per\ unit\ work)_m}$$

where $(Q\ position)_j$ is the number of jobs ahead of j in the queue for processing node n , and $work_{m,j}$ is the amount of work node m has to do on job j .

³I.e. a simulation within a simulation.

⁴When λ_{min} is 1, the queue length estimate is equal to the current queue length, when λ_{min} is 0, the queue length estimate is equal to the average of the observed lengths at all time steps so far. We used $\lambda_{min} = 0.1$ in Section 6.

In order to account for the fact that this estimate is only approximate, and hence allow the agent to manage risk sensibly, the expected utility of processing each job is evaluated not just with respect to this delivery time, but with respect to a cluster of times around this value, with corresponding approximate likelihoods. We have a collection of scale factors $s_j, j = 1, \dots, S$ and associated weights, w_j and evaluate the expected utility the job as

$$\begin{aligned} \text{expected_utility}(\text{job}, t) &= \\ &= \sum_j w_j \times \text{utility}(\text{job}, s_j \times t). \end{aligned} \quad (2)$$

Ideally the weights and scale factors should be tuned to the observed statistics of the application and its job stream. In our simulations we simply set these parameters arbitrarily to have values $s = \{2.0, 1.5, 1.0, 0.75, 0.5\}$ and $w = \{0.08, 0.2, 0.44, 0.2, 0.08\}$.

As described in Section 4, these estimates are used to derive utility estimates for particular resource bundles, and a bid for a given resource calculated as the difference between the expected utility *with* the resource, minus the expected utility *without* the resource. In this context, with the simple SLA structure whereby jobs either earn a reward if they're on time, or a penalty if they're late, there will be a difference between the "with" and "without" utilities only when not having a resource means that jobs are expected to miss their deadlines. This agent algorithm therefore procures resources "just-in-time". The fact that delivery times are only approximate, and utility estimates are based on weighted averages of nearby delivery times helps to mitigate this effect.

6 Results

In this section we describe the results of a series of experiments that were run to investigate the performance of a market-based resource allocator.

6.1 Basic Setup

The basic experimental scenario consisted of two statistically identical services implemented as two node workflow applications. The resource pool for these services typically consisted of 20 identical servers, each able to process 1000 work units per time step. A typical "run" lasted 1000 time steps, and results regarding average performance were taken over 100 such runs. The typical SLA

structure was that each job was due to be delivered 10 time steps after arriving in the first queue, would pay 1 unit of money on successful delivery, and would incur a fine of 1 if late.

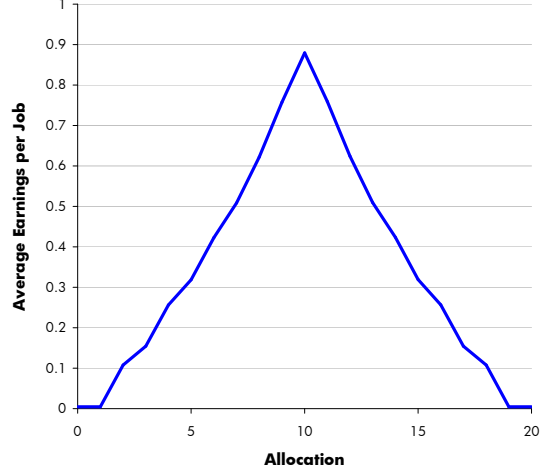


Figure 4: Average total reward for two services as a function of the static allocation of resources to their underlying applications.

Figure 4 shows simulation results for the average total revenue per job in such an experiment, where the average work per job per node was 300 and the average number of new jobs per time step per service was 17.

Since the two services are statistically identical, and jobs processed in each give the same reward, it is no surprise that the optimal static assignment in this case is to give half the servers to each service, in which case almost all (94.0%) jobs are completed on time, the remaining cases being accounted for by unusual situation in which a flood of arrivals at a service leads to very long queue waiting times and hence the occasional deadline violation. When all the resources are allocated to one service or the other, the average total revenue per job is approximately zero. This is because both services receive about the same number of jobs, one service succeeds in processing all of these jobs on time, earning 1 for each; the other service fails to process any jobs, paying a fine of 1 for each – the earnings and fines cancel each other out on average.

In terms of admission control, the equivalent earnings per share for only admitting one application are 0.5: all jobs will clearly be completed successfully, but there will only be half as many of them.

Figure 5 shows the resource levels of each ser-

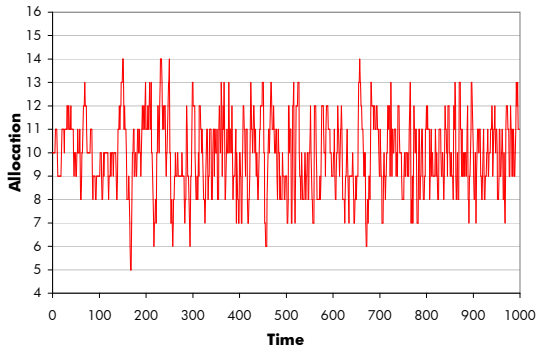


Figure 5: Resource levels for one of two applications in a 20 server UDC.

vice in the market-controlled system. In this first example, the ability to re-provision resources dynamically in response to business impact does not lead to dramatic changes in the combined effectiveness of the two services: the average earnings per job for the market controlled scenario is 0.94 as opposed to 0.88 for the symmetric static allocation, which corresponds to completing 97% of jobs on time as opposed to 94%.

6.2 Market Volatility

The volatility of the allocations, as depicted in Figure 5, is due to the last-minute nature of the bidding algorithms, and the fact that no costs are incurred for buying and selling resources (i.e. for stopping and starting application components). In a more realistic scenario, the costs of reallocation itself should be taken into consideration; this aspect will be studied in future work.

The correspondence between the volatility of the job stream and the volatility of the allocations that the market produces is illustrated in Figure 6, in which the allocation probability distribution function is plotted for various choices of arrival and work rates. The product of the two is held fixed (at 4000), so that both correspond to the same total amount of work, on average. Job streams corresponding to a few big jobs lead to more volatile demand, and hence to greater differences in demand between the two agents. This in turn leads to large swings in the allocation, and hence a more spread-out allocation distribution function. When the job streams consist of a large number of small jobs, the total amount of new work arriving per time step is more reliable, demand fluctuates less, and hence so does the allocation, which is almost always (for arrival rate 20, work rate 200) at (10, 10).

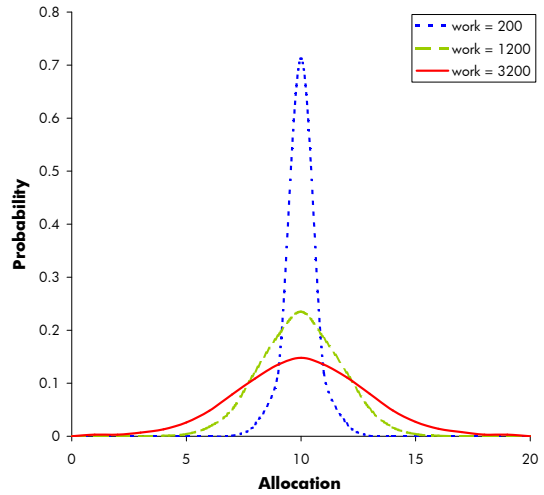


Figure 6: Smoothed allocation probability distribution function for three different work/arrival rate cases such that $work \times arrival = 4000$.

6.3 Emergent Time Sharing

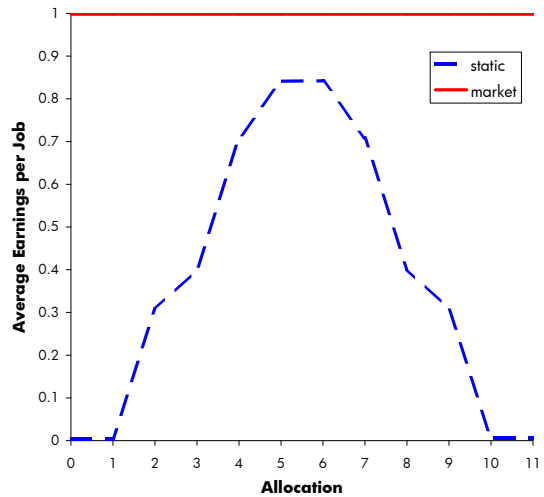


Figure 7: Average reward for two services when allocations are static, and earnings for market-based system for comparison.

For a more dramatic example of the benefits of market-based dynamic resource allocation, consider Figure 7. In this graph we depict a similar situation to Figure 4, but for 11 servers instead of 20, and for jobs of average work content 80 per node, of which on average 30 arrived at each service per time step.

In this scenario, the market finished all jobs on time in all 100 runs, earning 1 per job, whereas

the best static allocation was only able to earn on average 0.84 per job. A probable reason for the clear advantage of the dynamic allocation mechanism here is that in this scenario there is no symmetric static allocation. The return on a static allocation of 5.5 resources for each application would probably be comparable to that generated using the market-based allocation scheme, but in the resource model we use, such allocations are impossible. It follows that one of the two services necessarily has more resources than the other; if the work load is great enough, the queues on the less well endowed application will necessarily increase until it is unable to deliver all jobs on time. Since each application needs 5 resources most of the time, and a sixth only once in a while, a suitable solution to this problem is time-sharing of the 11th resource.

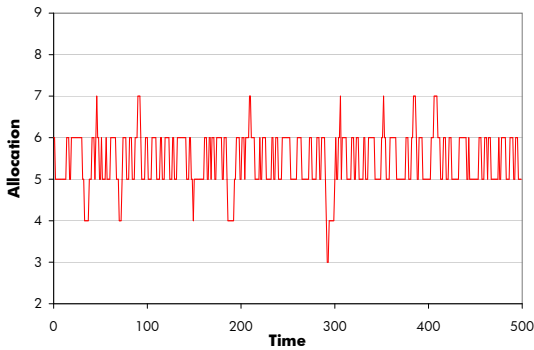


Figure 8: Example of emergent time-sharing.

Figure 8 shows a graph of the number of servers belonging to one of the applications, as a function of time. Most of the time the number shifts between 5 and 6 in response to critical queue-length increases: when the queue lengths on one service build up to dangerous levels (as measured by an increased probability of delivery failure), the agent responsible is willing to pay enough to buy a resource from the other agent. After doing so, the situation eases, but the other agent’s queues soon lengthen to the point at which it buys the resource back. Thus the market mechanism “discovers” the time-sharing solution to this problem.

6.3.1 Statistical Service Assurances

In the situation described above, the market-based paradigm is not the only one that could have been used to resolve server allocation decisions. If we measure the number of time steps in which an application is willing to bid a non-zero amount for

more than 5 servers, it is about 11% of the time. It follows that the expected number of time steps in which *both* agents will request more than 5 is on the order of 1% of time slices. We could therefore, as in [10], only guarantee the requested number of resources 99% of the time, and decide the 1% of cases in which both agents request 6 or more resources arbitrarily. This highlights the fact that although the market paradigm is clearly useful in low-load scenarios, it really becomes valuable when demand is high.

For example, in the basic scenario whose static pay-off graph is depicted in Figure 4, the average number of servers demanded by each agent is 18.6. Both agents have reason to ask for many more resources than they truly need, in order to cover the risk of unusually long processing times. We cannot guarantee that demand will be unconditionally met even 10% of the time, even though the symmetric static allocation completes 94% of jobs on time.

6.4 Admission Control

As an example of how market-based resource allocation can improve resource utilisation and hence enable more services to be hosted, consider Figure 9, which shows the earnings per job when the average number of arrivals per time step is dropped to 1, but the average amount of work per job is increased to 5000 (per node).

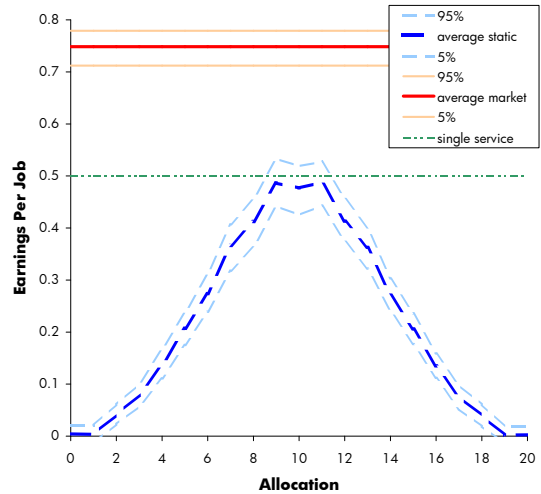


Figure 9: Admission Control example: no static allocation to two applications gives expected return as high as hosting a single service; the market-based resource allocator does.

This scenario is much more volatile than those

considered in Section 6.1 and Section 6.3; demand exceeds 10 in 92% of all time steps, and no static allocation consistently provides the same level of income as would be earned from only hosting one service. The market-based income however, is expected to be 0.75 per job, well above the earnings from a single service. Thus using the market-based resource allocator allows more services to be hosted, by increasing utilisation of the underlying infrastructure.

6.5 Heterogeneous Value

The next example demonstrates another desirable feature of market-based control, namely the fact that allocation decisions are intrinsically driven by business value. In this example, the two services have identical job arrival and workload characteristics, but the rewards and penalties for servicing jobs that arrive at the second service are twice those corresponding to the first service. A dynamic allocation scheme based on job arrivals or observed workloads would tend on average towards the symmetric allocation, since the two services are statistically identical from this respect. If, however, work-load increases to the point where there is contention for resources, the *rational* choice is to shift resources to the service whose SLA specifies greater penalties and rewards: if quality of service is forced to degrade, it should degrade first on the service from whose hosting the UDC operator extracts least utility.

Figure 10 shows graphs of the earnings per share against increasing workload for a collection of different static allocations, and for the market. The maximum possible reward per job here is 1.5, the corresponding reward from only hosting the second service is 1.0. Points to note are:

- As the workload increases, the optimal static allocation gives increasingly many resources to the second application, which is to be expected.
- The market performs better than every static allocation, even as different static allocations become optimal.
- The workload above which admitting application 1 is not advisable is much higher for the market-controlled system than for any static system.

Figure 11 shows allocation distributions for a variety of workload scenarios. As workload in-

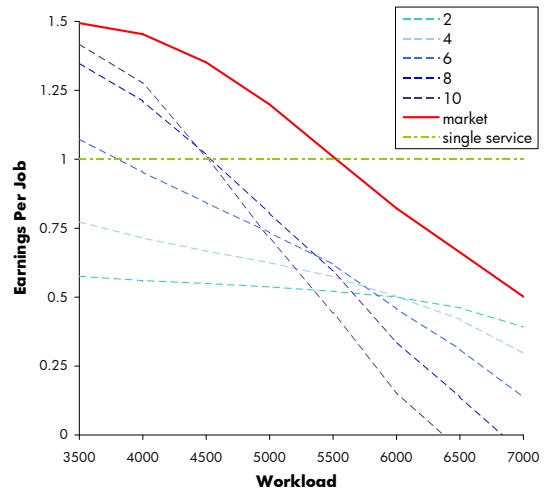


Figure 10: Relationship between average earnings and workload for a variety of static allocations, and for the market-based resource allocator. The label for static graphs indicates the number of resources given to the first application.

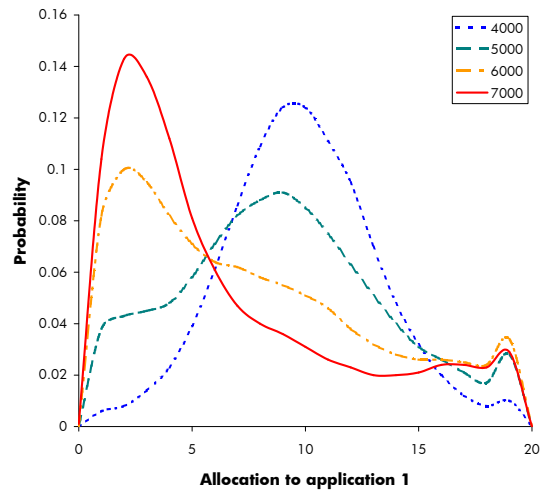


Figure 11: Heterogeneous value example. The graph shows smoothed allocation probability distribution functions for market-based systems under a range of workloads, and fixed arrival rate (of 1.0). As workload increases the system dedicates fewer resources to application 1.

creases, the market automatically makes the necessary trade-offs to provision more resources to application 2 than application 1 – for higher workloads, the allocations are, on average, skewed in favour of application 2.

6.6 Mechanism Comparison

In this final experimental section we compare different market mechanisms and application algorithms to investigate the degree to which the effectiveness of the market-based system is dependent on these design choices. The alternative market mechanisms considered is the “global optimisation” mechanism described in Section 5.2, whereby demand functions are requested from each agent for all possible allocations, and the allocation which maximises combined utility is selected. Graphs corresponding to this choice of market mechanism are labelled “global”. As an alternative algorithm for predicting job delivery times, we allow the agent to simulate the workflow queue network accurately, so that it has access to the *actual* delivery times. Graphs corresponding to this algorithm are labelled “simulation”.

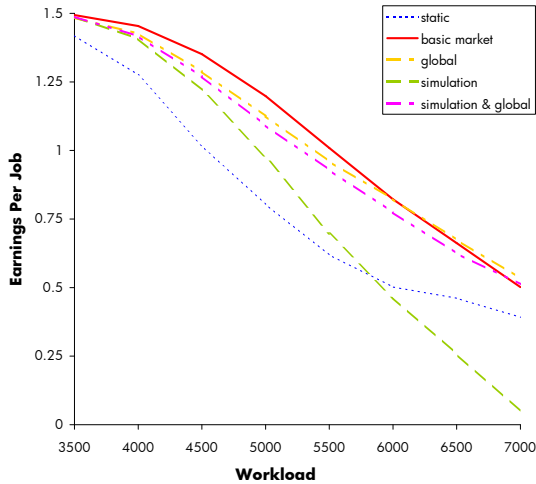


Figure 12: Comparison among market mechanisms and agent algorithms.

Figure 12 compares the degradation of earnings with respect to increasing workload for the same heterogeneous value scenario discussed in Section 6.5, and for all four combinations of simple/global market and simulation/no-simulation agent algorithm.

The surprising result is that, for a fixed mechanism, using the true job delivery times as opposed to distributional estimates of delivery times actually leads to *worse* performance. We hypothesise that this is because an unintended but beneficial effect of estimating delivery times and queue lengths is that the agent effectively places a strategic value on having short queue lengths: The estimation method

may predict a small probability of violation in cases that are not in fact going to violate; the longer the queues it observes, or the greater the workload, the greater this negative effect on predicted utility is likely to be.

Preferring short queues is good even when both long and short queues can be serviced with the same resources, because shortening a queue *now* leads to a lower probability of violation *later*, when more jobs have arrived. In essence, estimating delivery times with distributions intrinsically encourages it to pre-empt later demand, and hence reduces the dependency of the agent on last-minute provisioning. A corollary of this is that we expect other application agent algorithms that explicitly perform such longer-term planning to perform even better.

The effect of using globally as opposed to locally optimal allocations seems to be small for this example, possibly because the space of allocations is very small. If anything, the effect of using a “global” as opposed to “simple” market may be to reduce utility slightly, perhaps because the increased volatility tends to destabilise job processing; only further work will shed light on this issue.

7 Conclusions and Further Work

In this paper we have described a market-based system for managing resource allocation within a data center that is used for hosting outsourced services. In this business context, the market participants – agents that manage the provisioning of specific services – act cooperatively to maximise the utility of the data center operator. We described a particular sub-class of mechanisms suitable for this context, and tested two of them in an example scenario, in which it was shown that the market-mechanism never performed worse than a static allocation, and often performed significantly better. We also demonstrated scenarios in which demand was sufficiently high that no assurances could be given regarding the underlying infrastructure’s availability, but nevertheless the market-based system was able to balance application requirements so as to give earnings that exceeded those from hosting fewer services.

In our experience of building the simulation environment (and as demonstrated in Figure 12), market mechanism choices have far less impact on allocation value than bidding algorithm design. This is equivalent to saying that the allocation scheme is

highly sensitive to the methods for predicting service levels, which would, of course, be the same for any dynamic allocation mechanism that based allocation decisions on quality-of-service predictions. Nevertheless it indicates that a key aspect of future work is to provide greater accuracy and detail in our application models, and study a variety of applications. This includes, for example, introducing realistic cost models to describe the true dynamism of an infrastructure: how long does it take to move application components; what business is lost if a server is shut down, etc.

In addition to application modelling, there are several other important directions for future work.

- To complement accuracy in application modelling, we would like to provide greater accuracy and detail in our business, contract and SLA models, so that we can be sure that it is *business value* that is being optimised.
- We would like to study more sophisticated markets, including, for example, markets for other types of resources, such as storage and bandwidth. Perhaps more important still is the issue of scheduling. The system described in this paper is a spot market: resources are acquired when they are needed, and cannot be reserved ahead of time. For many applications the inability to reserve resources in advance may lead to severely sub-optimal performance, if resources are not available when they are required. Providing a venue where agents can buy or trade future access to resources is clearly, therefore, a priority.
- Moving away from the cooperative scenario, we intend, in further work, to address data center resource markets in which the participants are not owned by the infrastructure owner, and thus may act strategically to gain unfair advantage over one other.
- Finally, and most importantly, we intend to apply this research, and the fruits of further work as described above, to the design and implementation of market-based resource allocation within a deployed data center.

References

- [1] Karen Appleby, Sameh Fakhouri, Liana Fong, Germán Goldszmidt, Michael Kalantar, Srirama Krishnakumar, Donald Pazel, John Pershing, and Benny Rochwerger. SLA based management of a computing utility. In *Proc. 7th IFIP/IEEE Int. Symposium on Integrated Network Management*, May 2001.
- [2] Jeffrey Chase, D. Anderson, P. Thakar, A. Vahdat, and Ronald Doyle. Managing energy and server resources in hosting centers. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.
- [3] Scott H. Clearwater, editor. *Market-Based Control*. World Scientific, Singapore, 1996.
- [4] Ronald Doyle, Jeffrey Chase, Omar Asad, W. Jen, and A. Vahdat. Model-based resource provisioning in a web service utility. In *Proc. USENIX Symposium on Internet Technologies and Systems*, March 2003.
- [5] F. A. Hayek. The use of knowledge in society. *The American Economic Review*, 35(4), 1945.
- [6] *HP Utility Data Center Architecture*. <http://www.hp.com/solutions1/infrastructure/solutions/utilitydata/architecture>.
- [7] Terrence Kelly. Utility driven allocation. In *Proc. 1st Workshop on Algorithms and Architectures for Self-Managing Systems*, June 2003.
- [8] Z. Liu, M. S. Squillante, and J. L. Wolf. On maximizing service-level agreement profits. In *Proc. ACM Conf. on Electronic Commerce (EC '01)*, October 2001.
- [9] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. A. Mendes. Business-oriented resource management policies for e-commerce servers. *Performance Evaluation*, 42:223–239, 2000.
- [10] Jerry Rolia, Xiaoyun Zhu, Martin Arlitt, and Artur Andrzejak. Statistical service assurances for applications in utility grid environments. Technical Report HPL-2002-155, HP Labs, June 2002.
- [11] Cipriano Santos, Xiaoyun Zhu, and Harlan Crowder. A mathematical optimization approach for resource allocation in large scale data centers. Technical Report HPL-2002-64R1, HP Labs, March 2002.

- [12] Carl. A. Waldspurger, Tadd Hogg, Bernardo A. Huberman, and Jeff O. Kephart. SPAWN: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, February 1992.
- [13] Alex Zhang, Cipriano Santos, Dirk Beyer, and Hsiu-Khuern Tang. Optimal server resource allocation using an open queuing network model of response time. Technical Report HPL-2002-301, HP Labs, October 2002.
- [14] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In *Proc. IEEE Infocom 2001*, April 2001.