J Grid Computing
DOI 10.1007/s10723-007-9094-4

Can Economics-based Resource Allocation Prove Effective in a Computation Marketplace?

Kevin L. Mills · Christopher Dabrowski

Received: 15 June 2007 / Accepted: 28 November 2007

Abstract Several companies offer computation on demand for a fee. More companies are expected to enter this business over the next decade, leading to a marketplace for computation resources. Resources will be allocated through economic mechanisms that establish the relative values of providers and customers. Society at large should benefit from discoveries obtained through the vast computing power that will become available. Given such a computation marketplace, can economics-based resource allocation provide benefits for providers, customers and society? To investigate this question, we simulate a Grid economy where individual providers and customers pursue their own ends and we measure resulting effects on system welfare. In our experiments, customers attempt to maximize their individual utilities, while providers pursue strategies chosen from three classes: information-free, utilization-based and economics-based. We find that, during periods of excess demand, economics-based strategies yield overall resource allocation that benefits system welfare. Further, economics-based strategies respond well to sudden overloads caused by temporary provider failures. During periods of moderate demand, we find that economics-based strategies provide ample system welfare, comparable with that of utilization-based strategies. We also identify and discuss key factors that arise when using economic mechanisms to allocate resources in a computation marketplace.

Keywords Computation economy · Distributed resource allocation · Global behavior · Grid computing

K. L. Mills (✉) · C. Dabrowski
National Institute of Standards and Technology, Gaithersburg, MD, USA
e-mail: kmills@nist.gov

1 Introduction

Several companies [1–3] offer computation on demand for a fee. More companies [4] are expected to enter this business over the next decade, leading to a marketplace for computation. Society at large should benefit from discoveries obtained through the vast computing power that will become available for tasks such as discovering drugs, designing engines and analyzing financial risks. Successful operation of a computation market requires some means to allocate resources among competing demands. Given the global scale expected in such a marketplace, centralized allocation schemes will prove infeasible; thus, some form of decentralized resource allocation must be used. Economic mechanisms, usually based on individual self-interest, have successfully allocated resources in many markets [5, 6].

Can economics-based resource allocation in a computation marketplace prove effective at yielding benefits for providers, customers and society? To investigate this question, we simulate a Grid economy where individual providers and consumers pursue

their own ends and we measure resulting effects on system welfare. We build our Grid simulation [7] on emerging industry standards for web services [8] and Grid services [9], which are prerequisite to construct interoperable, large-scale computational Grids that span the globe. We extend our simulated Grid by introducing behaviors that allow consumers and providers to enter into service-level agreements, modeled after the WS-Agreement specification [10]. In our simulated Grid, computation resources are supplied for prices that change dynamically based on supply and demand. Our simulated consumers strive to maximize their individual utilities while completing assigned computing tasks. Our simulated providers adopt strategies for admission control (deciding which jobs to admit) and task selection (deciding execution order for admitted jobs). We investigate provider-decision strategies chosen from three classes: information-free, utilization-based and economics-based (as described in Section 4.1). Our experiments compare how system performance differs with provider-decision strategy given varying customer demands. We find that, during periods of excess demand, economics-based strategies yield overall resource allocation that benefits system welfare. Further, economics-based strategies respond well to sudden overloads caused by temporary provider failures. During periods of moderate demand, we find that economics-based strategies provide ample system welfare, comparable with that of utilization-based strategies.

The findings reported in this paper expand significantly on the scope and realism of previously reported research [11–18] into the use of economic mechanisms to allocate computation resources. First, we model a distributed marketplace through interactions of competing providers and consumers with realistic differences. Prices offered by providers vary based on individual experiences of supply and demand. Providers have differing cost bases and profit targets. Consumers have differing budgets and profit targets. Previous research either: (a) considered economic mechanisms to allocate the resources of only a single provider among multiple customers or (b) investigated centralized application of economic mechanisms for allocating resources across a collection of providers and consumers. Second, we define a metric for system welfare, conceived as aggregate utility across all providers, consumers and society (as indirect stakeholders). Most previous research considered welfare of an individual provider and did not include indirect stakeholders in a computation marketplace. Third, we adopt scenarios congruent with likely requirements of commercial operations. We relate deadlines of competing jobs to a defined workday and we consider demand ranges consistent with existing commercial experience [19]. Fourth, we investigate market robustness in the face of sudden, temporary failures among providers. Such failures may arise in computation marketplaces due to problems in the power distribution grid [20] or to attacks on selected provider sites [21].

The remainder of the paper is organized as six sections (Sections 2–7). Section 2 outlines related work on allocating computation resources using economic mechanisms. Section 3 gives a brief overview of our model for a Grid economy and defines key metrics used to measure and compare system performance. Section 4 describes our experiment design. Section 5 presents our simulation results and discusses our findings. Section 6 outlines future work. We close in Section 7 with conclusions.

2 Related Work

The emergence of cluster and Grid computing has stimulated significant interest in finding effective methods to allocate computation resources. Several researchers have investigated how economic and market-based mechanisms could be used to allocate computation among competing users. After surveying the work of other researchers, we distinguish our contributions.

Chun and Culler [11] compare value-based scheduling against traditional alternatives (e.g., first-in, first-out and priority scheduling) for ordering job executions in a cluster of processors. Job value is expressed as a piecewise linear utility function where jobs have a given value until a deadline and then decay at a defined rate until becoming valueless. Chun and Culler assume a job mix where 20% of jobs have a high value and also high decay rates (100 times steeper than for low-value jobs) and then compare how well various scheduling algorithms perform when jobs are submitted in batches to a simulated compute cluster. Job arrival times and job lengths are drawn from a normal distribution and job widths (i.e., number of required processors) are

sampled from a uniform distribution. The paper by Chun and Culler is notable because it sets out some assumptions adopted by subsequent researchers. First, the paper defines the notion of a piecewise linear utility function for job values (though subsequent researchers extend the function to allow negative utility). Second, the paper defines the notion of low-value vs. high-value jobs and low-urgency vs. high-urgency jobs and then defines an 80/20 mix of jobs by value and urgency. Subsequent researchers also adopt this job mix.

Yeo and Buyya [12] investigate how economic reasoning can be used to control job admission in a single compute cluster when arriving jobs have varying deadlines, values and decay functions—but where decay is unbounded and utility may become negative. They adopt the same 80/20 mix of job values as Chen and Culler, but divide jobs into two classes: hard deadlines and soft deadlines. For jobs with soft deadlines, 20% have high urgency. An arriving job that enhances the aggregate value of the cluster's schedule is admitted. Admitted jobs are scheduled using a proportional share algorithm. Yeo and Buyya vary the proportion of jobs with hard vs soft deadlines and quantify the improvement of allowing soft deadlines with decreasing utility functions against insisting that all jobs have hard deadlines.

Irwin et al. [13] investigate admission control and scheduling in a single cluster where arriving jobs have varying deadlines, values and decay functions (with either bounded or unbounded penalties). An arriving job is provisionally admitted to the schedule and then a resulting reward and risk are computed. Reward is the increase in schedule value from accepting the new job and risk estimates the effects of accepting the job on the ability to accept future jobs. The admission function computes a score based on relative weights assigned to reward vs risk and then admits the job if the score exceeds a threshold. Irwin et al. adopt an 80/20 mix of job values and urgencies but experiment with various skews for high value (up to nine times) and urgency (three to seven times) and with various admission thresholds.

Popovici and Wilkes [14] investigate a single service provider where admission decisions are made based on whether an arriving job is likely to increase provider profit given that jobs are executed according to a schedule chosen from among various economic and traditional (e.g., shortest-job first or longest-job first) criteria. The service provider rents processors from multiple resource providers in a secondary market in order to serve arriving jobs. The secondary market enables provider costs to vary with time; however, a single provider is unable to reflect price changes due to competition with other providers. Popovici and Wilkes model jobs that can be spread across a variable number of processors, depending upon resource availability. Such jobs run faster when more processors are available and more slowly otherwise. Job values have a piecewise utility function with unbounded penalties.

AuYoung et al. [15] extend the work of Popovici and Wilkes by relating submitted jobs into a larger framework of contracts negotiated between clients and service provider. The framework allows clients to specify the overall value of a set of jobs in addition to the value of individual jobs. The service provider makes admission decisions on contracts and then considers the contract along with job value in deciding individual job admissions. AuYoung and colleagues experiment with various contract admission functions, such as always accepting contracts, accepting contracts when utilization remains below a desired level and accepting contracts when the added value exceeds some threshold. Previously accepted contracts may be cancelled in order to accept more lucrative contracts. Maximum cancellation penalty is the value of a job. Provider costs may vary over time (through a secondary market for resources). The job mix consists of the usual 80/20 proportion of low vs. high value and urgency. AuYoung et al. introduce a delay of 1.5 times job length before a job begins to decay in value. The service provider is assigned a target profit margin of 50%.

Wolski et al. [16] allocate computation (and disk storage) resources among multiple providers and clients using a centralized marketplace: either a commodity market or an auction. Jobs are injected in batches at the beginning of each simulated workday and also arrive sporadically over the course of each day. Wolski and colleagues simulate a commodity market based on an engineering approximation of Smale's [22] algorithm and compare resource allocation under that regime against an optimal implementation of Smale's algorithm and against allocation decided through an auction clearinghouse. They demonstrate that auctioning leads to lower resource utilization given situations where

supply exceeds demand. In other results, they demonstrate that auctioning performs more competitively during overload.

Gomoluch and Schroeder [17] compare continuous double auctions, proportional share and round-robin resource-allocation algorithms to determine conditions under which market-based resource allocation proves superior. They simulate multiple clients and service providers interacting through a centralized clearinghouse that matches client requests with available provider capacities. As in Wolski et al., service providers and clients are given no possibility to reject matches assigned by the market. Gomoluch and Schroeder experiment with a restricted set of characteristics including identical job size, identical server capacity, identical prices and loads that never exceed 90%.

Vengerov [18] investigates tractable algorithms that a server could use to schedule jobs that may be repackaged during run time to use more or fewer processors in tradeoff for faster or slower execution time. He uses a piecewise linear utility function without penalty but allows job value to begin decaying only after twice the shortest feasible run length. Run lengths and maximum job widths are sampled from uniform distributions. All jobs have the same basic value and decay rate. Vengerov experiments with three clusters each with 24 processors, where each cluster appears to have been given an independent stream of job arrivals.

As we describe in our survey, most of the related research considers multiple jobs arriving at a single server or considers a centralized market when clients are matched up with a service from among those available. Examining a single provider reveals nothing about the effects of competing customers and providers interacting in a marketplace. The use of centralized markets is probably unrealistic for computation economies of global size. The spatiotemporal demands of a global market make it unlikely that a central clearinghouse could capture all available providers and serve all available customers. Further, most real economies allow customers and providers to negotiate, rather than to accept assigned matches. More likely, indices will be deployed and customers will search independently for providers with which to conduct negotiations. In such a distributed market, most customers will be unable to find and process information from all providers; thus, customers will adopt heuristics for finding and selecting providers for negotiation and providers will use heuristics for admitting and scheduling customers and will also be free to adjust prices based on such interactions. This, then, is one key contribution of our paper—we represent a plausible distributed market where multiple customers and providers interact over space and time while pursuing their self-interests.

Our survey of related work also reveals that most researchers measure system performance through observing effects on the value or profit generated by a single provider. In a system with competing providers and customers a broader measure is required. The interests of the providers must be aggregated but the interests of the customers must also be assessed. Further, customers are running computations for some perceived social benefit; thus, society at large must be represented. This, then, is a second contribution of our paper—we define a metric to represent aggregate system welfare and we compare our candidate admission and task selection algorithms using this metric.

Most of the work we surveyed represents jobs (or batches of jobs) as a continuous stream where inter-arrival times are modulated to vary system load. This abstract approach causes job deadlines to vary so as to mask real-world effects that appear when jobs arrive in batches early within a workday and must be completed before day's end. Most extant research also considers system operation under conditions where capacity remains unchanged during experiments. This does not allow investigation of system performance when power outages or cyber attacks cause temporary disruptions. Our paper adopts realistic scenarios where jobs arrive in a concentrated period at the beginning of a workday and thus compete to finish by similar deadlines. We also investigate how a computation marketplace behaves under sudden, temporary disruptions.

The bulk of the simulations we surveyed use abstract mathematical representations of component behaviors, which enable reasonable evaluation of algorithms for admitting jobs and for orchestrating their execution. Using abstract representations does not, however, enable consideration of the influence of system configuration details or allow one to evaluate whether emerging standards for web services and Grid services can support proposed algorithms. Our models simulate underlying web services and Grid

services in significant detail, which allows us to formulate admission control and execution control schemes consistent with emerging standards. Elsewhere [23], we provide an in-depth investigation of the ability of web services and Grid standards to support a computation marketplace.

3 Overview of Model and Metrics

We model a computation marketplace as a collection of components (clients, service providers and directory servers) interacting via messages across a communications network. Details of the fundamental components can be found elsewhere [7]. Here, we provide an overview of the model and then discuss task characteristics (Section 3.1), provider behaviors (Section 3.2) and client behaviors (Section 3.3). In Section 3.4, we define metrics relevant for the experiments reported in this paper, especially a metric for system welfare, conceived as aggregate utility across all providers, consumers and society (as indirect stakeholders).

Our model represents a marketplace in which providers sell computation resources for a profit while clients seek to maximize their utility while obtaining resources on which to execute tasks under budget and time constraints. The average market price is chosen initially to balance provider costs with client budgets; we define behaviors to vary price in response to supply and demand. Model components are distributed over a simulated topology of sites, where relative distances among sites influence communication delays experienced by messages exchanged among components. Clients, service providers and directory servers are constructed from simulated web services [24–26]. The directory servers, modeled after the Globus Toolkit 4 [27], also rely on service groups [28], which are linked together through simulated query aggregators to form a global directory.

Service providers insert descriptions of offered computation resources into local directory servers and those descriptions propagate throughout the global directory. Clients query selected directory servers for a bounded list of providers with resources matching their needs and then query each of those providers for pricing and availability. Using this additional information, a client prioritizes providers based on some criteria and then negotiates with the highest priority provider to establish a contract, or service-level agreement (SLA) [29], to execute a task under specified terms and conditions. A SLA allows clients and providers to understand exactly what each party is expected to provide in order to complete a task and resolve payment.

Our model implements the concepts that underlie SLAs as expressed in the WS-Agreement specification [10]. An agreement identifies relevant parties involved in a contract and describes the terms that each party is expected to fulfill. The agreement also specifies rewards each party is to receive upon fulfilling terms and penalties incurred in the event of failure to do so. In our model, agreement terms include: expected task duration, number of cycles needed, number of processors needed, task deadlines, fee to be paid to provider if target deadline is met, rate at which fees decay if tasks are delayed beyond the deadline, penalty fees to be paid if terms are violated, and network addresses where input data can be retrieved and where output is to be sent. We map SLA terms into data structures reflecting the structure of agreements as specified in WS-Agreement [10]. Once task execution completes or is aborted or otherwise terminated, agreement terms are used to determine the fee or penalty to be paid, and then funds are exchanged electronically.

As specified by WS-Agreement [10], providers in our model make available a downloadable template that provides an outline of proposed agreement terms and creation constraints, i.e., bounds on the values terms may assume. In our model, creation constraints include the provider's current level of resource utilization and an associated fee schedule. The level of resource utilization allows clients to estimate the likelihood of task completion by a particular time. Clients download the template and instantiate it with specific values, using the creation constraints to determine appropriate ranges for deadlines and fees. An instantiated template is forwarded as an offer to the associated provider. Our model follows the negotiation protocol employed in WS-Agreement: a provider immediately acknowledges receipt of an offer and subsequently determines whether to accept or reject an offer (determination procedures implemented in our model are described below in Section 4.1). The provider then sends the client an acceptance or rejection, which concludes the negotiation (in WS-Agreement, further counteroffers by client or provider

are not allowed; thus, renegotiations are viewed as new negotiations). Upon acceptance, a client submits the accepted task to the provider. After a rejection, the client negotiates with the next highest priority provider and so on. If all providers reject offers, the client waits for some time and then queries the directory server again. Each client continuously repeats this cycle of discovery and negotiation until its tasks are completed or overtaken by other events.

3.1 Task Model

Each client is assigned a set of independent tasks and a budget. Each task has a pF_{tsk} (number of processors required in parallel) and pL_{tsk} (duration for which the processors are required), where $pF_{\text{tsk}} \times pL_{\text{tsk}} \times pC_{\text{cyc}} = pC_{\text{tsk}}$ (total processor cycles required for a task). Once selected, a task's pF_{tsk} and pL_{tsk} remain unchanged. Each task also has an associated (piecewise linear) value function (see Fig. 1) describing its revenue (r_{tsk}) if completed by a target deadline (D_{tsk}) and a rate of decay in revenue (k_{tsk}) if the task is completed after D_{tsk}. If a task completes at $E_{\text{tsk}} > D_{\text{tsk}}$, then r_{tsk} is reduced. In general, a client receives revenue for a task according to the following function:

$$r'_{\text{tsk}} = \begin{cases} r_{\text{tsk}} & \text{if } E_{\text{tsk}} \le D_{\text{tsk}} \\ r_{\text{tsk}} - k_{\text{tsk}} \times (E_{\text{tsk}} - D_{\text{tsk}}) & \text{otherwise} \end{cases} \quad (1)$$

If a task does not complete before its maximum deadline, then the client receives no revenue. In our simulation the target (D_{tsk}) and maximum (M_{tsk}) deadlines for a task are drawn from distributions (discussed in Section 4.2) that provide a cushion of about 3 to 3.5 times the typical run length before a task begins to decay in value. Once D_{tsk} and M_{tsk} are known, k_{tsk} can be determined as follows:

$$k_{\text{tsk}} = r_{\text{tsk}} / (D_{\text{tsk}} - M_{\text{tsk}}). \quad (2)$$

In order to make a profit, a client must obtain computation resources that cost less than the revenue gained from completing its tasks. Each client is assigned a target profit margin m_{trg} that represents the percentage of its budget that should be profit. A minimum profit margin m_{min} constrains spending on tasks. To establish task budgets, we first determine a system-wide per cycle average base price pC_{Avg}, by averaging individual provider per cycle base prices b_{base} across the set of all providers, where (6) below defines how each provider computes a b_{base}. Given pC_{Avg} a task's cost can then be estimated initially as

$$c_{\text{tsk}} = pC_{\text{Avg}} \times pC_{\text{tsk}} \quad (3)$$

From this, the maximum revenue for a task (r_{tsk}) can be calculated as

$$r_{\text{tsk}} = c_{\text{tsk}} / (1 - m_{\text{trg}}). \quad (4)$$

The maximum task budget (i.e., the most a client is willing to pay) is set to $r_{\text{tsk}} - (r_{\text{tsk}} \times m_{\text{min}})$, while the minimum task budget is set to $r_{\text{tsk}} - (r_{\text{tsk}} \times m_{\text{trg}})$. When the potential revenue for a task begins to decay ($E_{\text{tsk}} > D_{\text{tsk}}$), the client willingness to pay decays at the same rate (k_{tsk}). If a provider accepts a task but does not complete the task prior to M_{tsk}, then the task is canceled and the client is reimbursed a penalty equal to $p_{\text{tsk}} \times r_{\text{tsk}}$, where p_{tsk} is the penalty rate established in the agreement between client and provider.

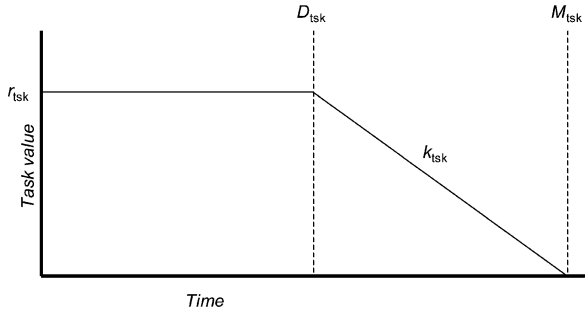

Fig. 1 Piecewise linear decay over time in the value of a task

3.2 Provider Model

We model each provider as a set of compute resources, where each resource offers some number of parallel processing elements that can supply computation cycles to execute application code uploaded by clients with which the provider establishes agreements. Each provider admits and executes jobs independently of other providers. A provider's processing resources are not accessible directly to clients. Instead, each provider implements a front-end

process to advertise characteristics of the processors, to accept requests from clients for current pricing and availability, and to spawn client proxies to process client offers and to monitor execution of client tasks. Advertisements from a provider for each of its processing resources include: number of parallel processing elements (pF), speed of each processing element (pS cycles/second) and a price schedule (pC_{cyc} cents per compute cycle). For the experiments conducted in this paper, we limited each provider to manage a single resource, which here represents a compute cluster. We adopted this restriction to enable us to better analyze our results. We varied pC_{cyc} among providers to represent differences in operating costs for the providers. We adopted the same pS for all processing elements under the assumption that our providers were in a competitive market where they maintained their processing elements at the forefront of the price-performance curve available from chip manufacturers.

The client proxies spawned by a provider interact with its compute resources through a simulated job manager modeled after the Distributed Resource Management System [30]. Our model adopts a queuing-based resource management system, which is the class used most widely among high-performance computers today [31]. A proxy submits a client offer to an admission controller that makes acceptance decisions. If an offer is accepted, then the admitted task is placed on a queue of pending tasks until a sufficient number of processors become available to execute the task. A task selector decides when queued tasks may execute. Multiple tasks may execute simultaneously, provided their combined pF_{tsk} requirements do not exceed the pF of a provider's processing resource. When a provider's processing resource does not have sufficient free processors to execute a task then the task is delayed until some executing tasks finish (i.e., our model does not include preemption). When a task at the head of the queue must be delayed, a task selector may backfill with tasks from deeper in the queue—provided those tasks fit within the free processors. The job manager monitors task status on behalf of the client proxies associated with executing tasks.

The provider architecture in our model separates activities of the admission controller, the task selector and the job manager into logically distinct processes. The admission controller has access to the entire queue of accepted tasks (both pending and executing) and the characteristics of those tasks. The admission controller can use task characteristics when making an admission decision; however, the admission controller in our model cannot be certain about the precise time remaining for executing tasks. This uncertainty arises because the task run-time estimates might prove inaccurate and because task execution times might be perturbed by checkpointing overhead and by overhead associated with overcoming failures in processing elements. For this reason, when considering admission decisions, the admission controller uses heuristics to estimate the run-time remaining for currently executing tasks.

Each provider has a *price monitor* that adjusts its price schedule to reflect resource utilization and clients' willingness to pay. In our model, the price schedule is set initially based on provider cost and desired profit margin. Each provider is assigned a cost for executing a compute cycle, pC_{cyc}, and a minimum profit margin, $m_{\min}$. Knowing the pF, pC_{cyc}, and pS associated with a provider resource, one can calculate total system cost for the provider (c_{prv}) over a given system lifetime (L_{sys}) as

$$c_{\text{prv}} = pF \times pS \times pC_{\text{cyc}} \times L_{\text{sys}} \tag{5}$$

Knowing cost pC_{cyc}, and the minimum profit margin $m_{\min}$, and having an estimate of the expected load, u_{exp}, the provider can determine a base price b_{base} for a compute cycle, which the provider must minimally charge to obtain an acceptable profit,

$$b_{\text{base}} = pC_{\text{cyc}} \Big/ u_{\text{exp}} \times (1 + m_{\min}), \tag{6}$$

where $pC_{\text{cyc}}/u_{\text{exp}}$ represents a breakeven cost.

The provider forms a price schedule, in which L_{sys} is divided into a set of n intervals, $i=1\ldots n$ with the asking price b^i_{ask} listed for each interval. Initially, each b^i_{ask} is set to b_{base}. To adjust the price schedule, the price monitor periodically computes the average per cycle price b^i_{avg} from fees for jobs pending in each future interval i, calculates the processor utilization u_i for each future interval, and updates b^i_{ask}

$$b^i_{\text{ask}} = b^i_{\text{avg}} + \left(u_i/u_{\text{exp}} - 1\right)^{1/2}. \tag{7}$$

In this scheme, asking prices fluctuate to reflect expressed willingness to pay (demand) by clients and

provider utilization (supply). Higher willingness to pay and higher utilization lead to higher asking prices. The model implements a policy that providers will not advertise prices that yield profit margins below $m_{\min}$; therefore, if $b^i_{\text{ask}} < b_{\text{base}}$, the provider advertises b_{base}. As providers enter into agreements and complete tasks, they accumulate revenue (r_{prv}) from which they calculate their profit, $p_{\text{prv}} = \left(r_{\text{prv}} - c_{\text{prv}}\right)$. Recall that any task delayed beyond an agreed time (D_{tsk}) decays in revenue at an agreed rate (k_{tsk}), so providers receive decreased revenue for delayed tasks. Further, a task that is not completed by an agreed deadline (M_{tsk}) incurs a penalty that is subtracted from provider revenue (and added to client revenue).

Each provider strives to maximize its profit (p_{prv}) over system lifetime (L_{sys}). To do this, a provider must be selective about which offers to accept and about the order in which accepted tasks are executed. To that end, each provider executes an *admission algorithm* to determine which offers to accept and which to reject. Each provider also employs a *task-selection procedure* to decide which task (if any) to run when processors become available. In this paper, we compare various admission algorithms and task-selection procedures that consider economic as well as other criteria. We postpone describing these algorithms and procedures until Section 4.

3.3 Client Model

We model each client as a set of independent tasks that proceed in parallel. Here, we describe the actions of the client on behalf of each task. The client discovers a subset of providers that each has a sufficient number of processors ($pF_{\text{tsk}} \leq pF$) to execute the task. The client then iterates over the price schedule of each discovered provider to find the interval where the provider yields maximum utility, defined as $r'_{\text{tsk}} - c_{\text{tsk}}$, where r'_{tsk} is determined by (1) and c_{tsk} for each interval i is computed by $b^i_{\text{ask}} \times pC_{\text{tsk}}$. The client then orders providers on the basis of decreasing maximum utility and sends an offer to the provider that promises greatest utility. An offer includes a target deadline (D_{tsk}), maximum deadline (M_{tsk}), rate of decay (k_{tsk}), and proposed fee. The proposed fee is a minimum of c_{tsk} and the client's maximum budget for the task. This prevents the client from overspending. If c_{tsk} is less than the minimum budget, then the proposed fee is the minimum budget. This gives enhanced incentive for the provider to accept and execute the task. If the provider accepts the offer, then the client submits any required data and monitors execution progress. If negotiation fails, then the client makes an offer to the provider promising next greatest utility, and so on down the list until all discovered providers are exhausted. If no offer is accepted, then the client waits some time before restarting the discovery process.

3.4 Metrics

We instrumented our simulation to collect various measurements, divided into two general categories: system-related metrics and causal metrics. We formally define our system-related metrics and we informally describe the others.

System-Related Metrics We devised an overall measure of *system welfare* to reflect (a) aggregate monetary benefits to clients (B_{cli}) and providers (B_{prv}) within a computation marketplace and (b) indirect benefits (P_{fin}) to a larger community who benefit from completed computations. We combine client benefit, provider benefit and social benefit to compute system welfare (S) as

$$S = \left(B_{\text{cli}} + B_{\text{prv}}\right) \times P_{\text{fin}}. \tag{8}$$

Indirect benefits to the larger community are generally difficult to quantify because some computations may result in discoveries that lead to greater societal value than others. Absent specific information about the ultimate social value of individual computations, we assume that a higher rate of task completions will yield a higher social benefit in aggregate; thus, we measure indirect benefits through the proportion of completed tasks, regardless of revenue or profit realized by clients or providers. Let V be the set of all tasks submitted by all clients during L_{sys} and let W be the set of tasks completed by all providers during L_{sys}. The proportion of tasks that are finished by L_{sys} is defined as $P_{\text{fin}} = |W|/|V|$.

We estimated aggregate client and provider benefits more directly. Aggregate *client benefit* (B_{cli}) is defined

as the ratio of client profit (revenue minus cost) to potential revenue for all tasks

$$B_{\text{cli}} = \frac{\left(\sum_{V} r'_{\text{tsk}} - \sum_{V} c_{\text{tsk}}\right)}{\sum_{V} r_{\text{tsk}}}. \tag{9}$$

Note that aggregate client benefit might decline at high load levels as available processors become harder to find, causing tasks to cost more and to incur higher delays (leading to lower client revenue).

Similarly to client benefit, aggregate *provider benefit* (B_{prv}) is defined as the ratio of profit (p_{prv}) obtained by all providers P to the potential of revenue available from all clients U during L_{sys}, i.e.,

$$B_{\text{prv}} = \frac{\sum_{P} p_{\text{prv}}}{\sum_{V} r_{\text{tsk}} - \left(\left(\sum_{V} r_{\text{tsk}}\right) \times \frac{\sum_{U} m_{\min}}{|U|}\right)}. \tag{10}$$

The denominator defines all revenue that could possibly be available to providers. This is the revenue available to clients less the minimum profit amount that clients are willing to accept. In our model, individual clients may have different minimum profit margins and so we compute an average profit margin from the individual profit margins and use that average to determine the aggregate amount of client revenue that will be reserved to the clients, and thus unavailable to the providers.

Omitted from system welfare (8) are communication costs for clients to discover and negotiate with providers. We decided to account for communication costs as system overhead, measured in terms of a normalized number of messages transmitted to discover providers and negotiate for task admissions.

We measured discovery overhead in terms of normalized messages transmitted to complete discovery. As a normalizing constant, we estimated an expected number, X_{dis}=56, of query and response messages needed for a typical task to complete one round of discovery by retrieving service descriptions from all directory servers for an average number of relevant providers. We measured the actual number of discovery messages (M_{dis}) for each task in V. The ratio of M_{dis} over X_{dis} defines relative discovery overhead (in units of X_{dis}) for a task. We computed the average discovery overhead ($\overline{O}_{\text{dis}}$) across all tasks in V as

$$\overline{O}_{\text{dis}} = \frac{\sum_{V} (M_{\text{dis}}/X_{\text{dis}})}{|V|} \tag{11}$$

$\overline{\overline{O}}_{\text{dis}}$ denotes $\overline{O}_{\text{dis}}$ averaged over all experiment repetitions for a given load.

We measured *negotiation overhead* in terms of normalized messages transmitted to complete a successful negotiation. As a normalizing constant, we estimated an expected number, X_{ngt}=23, of inquiry, offer, decision, and response messages exchanged between a client and an average number of providers for a typical task to conclude a successful negotiation. We measured the actual number of negotiation messages (M_{ngt}) for each task in V. The ratio of M_{ngt} over X_{ngt} defines relative negotiation overhead (in units of X_{ngt}) for a task. We computed the average negotiation overhead ($\overline{O}_{\text{ngt}}$) across all tasks in V as

$$\overline{O}_{\text{ngt}} = \frac{\sum_{V} \left(M_{\text{ngt}}/X_{\text{ngt}}\right)}{|V|} \tag{12}$$

$\overline{\overline{O}}_{\text{ngt}}$ denotes $\overline{O}_{\text{ngt}}$ averaged over all experiment repetitions for a given load. Summing discovery and negotiation overhead provides a measure of the communication cost ($C = \overline{\overline{O}}_{\text{dis}} + \overline{\overline{O}}_{\text{ngt}}$) incurred to achieve the corresponding system welfare.

Causal Metrics We computed P_{Adm}, the probability a task was admitted, P_{Can}, the probability a task was canceled, and P_{Acc}, the likelihood that providers accepted offers. Further, we categorized task types based on relative potential value (high or low) and recorded the proportion of task completions in each category.

4 Experiment Description

Our model consists of about 40 factors that might justifiably be investigated for influence on the behavior of a computation marketplace. Even limiting each factor to two levels would require at least 2^{40} simulation runs, which would be computationally infeasible. For this reason, we limited our experiments

to a computationally feasible set that we believed would yield significant insights. In this regard, we focused our current investigation on the influence of provider decision strategy over system welfare under selected mixes of provider types. We also investigated whether the effects of site outages would change our observed results. To accomplish our aims, we defined three experiments to compare up to eight provider strategies; each strategy paired an admission algorithm with a task-selection procedure. For a given experiment, we instantiated topologies (each consisting of a fixed number of sites for providers, clients and directory servers that were placed at random locations) and subjected those topologies to varied system workload from low levels (5 to 45%) through normal operating ranges (50 to 100%) and finally into overload (105 to 200%). Each experiment considered a different scenario: (a) homogeneous, (b) heterogeneous or (c) failures-prone. In this section, we first describe our provider strategies for deciding admission and task-execution order and then specify experiment topologies and workload generation. We close the section by outlining the experiment scenarios.

4.1 Provider-Decision Strategies

Our simulated providers adopted paired strategies for admission control (deciding which tasks to admit) and task selection (deciding execution order for admitted tasks). We investigated provider-decision strategies chosen to represent three classes: information-free, utilization-based and economics-based. Information-free decision strategies are easy to implement, requiring no information and little provider processing cost. Such strategies serve as a baseline for experiments because any provider-decision strategy that requires information and significant calculation should yield improved system performance over that obtained when using information-free strategies. Utilization-based strategies represent the state-of-practice for admission control and task selection in computational Grids. Such strategies restrict admission only after a provider reaches a specified threshold for a selected measure of utilization (e.g., number of tasks queued or percentage of resources committed). Utilization-based admission-control algorithms can be combined with a variety of well-known task-selection algorithms (e.g., shortest-job first, priority or first-come-first-served). Economics-based strategies consider the effects of admitting a task on the overall value or profits of previously admitted tasks. Economics-based strategies also consider relative values or profits of admitted tasks when selecting the order in which tasks will execute. To limit our study, we chose six (paired) strategies.

We defined two information-free strategies: *no control* (NoCon) and *random* (Rand). Providers using the NoCon strategy admitted all offered tasks (except those that could not be completed by their maximum end time) and executed admitted tasks in the order they were admitted (first-come-first-served). Providers using the random strategy flipped a fair coin to decide whether or not to admit each offered task. Subsequently, when resources become available Rand providers also flipped a fair coin to select which of the queued tasks to execute, where each eligible task had an equal chance of being selected.

We defined two utilization-based strategies: *shortest-job first* (SJF) and *first-come-first-served* (FCFS). Both strategies applied the same admission criteria, based on the percentage of resources committed over the period during which a task could execute. Resource commitment was computed using a straightforward procedure: the *pF* requirements and run times of the proposed task and previously admitted tasks were used to calculate potential utilization within the period delineated by the maximum end time (M_{tsk}) of the proposed task. If the calculated utilization exceeded a predetermined threshold, then the task was rejected. Otherwise, it was admitted. We chose a threshold of 95% utilization, which yielded the best performance among a variety of thresholds (ranging from 70 to 105%) that we tested. The two utilization-based strategies were distinguished by the procedure used to order tasks for execution. SJF ordered tasks by increasing run length (pL_{tsk}), allowing short tasks to run earlier. FCFS attempted to run tasks in the order they were admitted.

A number of researchers have proposed economics-based strategies for admission and execution of computational tasks [e.g., 11–18]. We defined two such strategies: *net-profit increase* (NetPro) and *net-value increase* (NetVal). The NetPro strategy, motivated by proposals from Popovici and Wilkes [14], admits a proposed task only when the potential *profit* of the task exceeds the lost value due to decay, $k_{\text{tsk}} \times (E_{\text{tsk}} - D_{\text{tsk}})$, in revenue that would accrue if the proposed task were admitted. Thus, when using

the NetPro strategy a provider may choose to delay less valuable tasks to admit or execute profitable tasks. Prior to admitting a proposed task, j_{new}, the set of previously admitted tasks, $\{j_1,\ldots j_n\}$, ordered by decreasing profit (i.e., revenue minus cost), which denotes the predicted execution order of the tasks, has an aggregate value, R, which is the sum of r'_{tsk} over all admitted tasks. Integrating a proposed task into the existing set, $\{j_1,\ldots j_{new}, j_k\ldots j_n\}$, yields an adjusted value R', which may reflect revenue decrease due to delay in executing $\{j_k,\ldots j_n\}$, previously accepted tasks ranked behind j_{new}. A proposed task is admitted only if the task's profit (including any decay caused by tasks ranked ahead, i.e., $j_1,\ldots j_{\text{new-1}}$) exceeds the revenue decrease ($R-R'$); otherwise, the task is rejected. In the NetVal strategy, motivated by Yeo and Buyya [12], tasks are executed in order of decreasing revenue. NetVal admits a proposed task using the same decision mechanism as NetPro, except that only a task's *revenue* (not profit) must exceed $R-R'$. Once admitted, pending tasks are executed in order of decreasing profit (NetPro strategy) or revenue (NetVal strategy).

No matter which strategy (FCFS, SJF, random, decreasing value or decreasing profit) was used to order task executions, all providers adopted a *first-fit backfill* algorithm to increase resource utilization. Each provider invoked its task-selection procedure upon admission of an arriving task and upon completion of an executing task. The Rand strategy selected randomly from among only those tasks for which sufficient processors were available. For the other strategies, if insufficient processors were available to run the first task in an ordered set, then the provider moved down the set to select the first task with a pF_{tsk} requirement small enough to fit within the available processors. If no task would fit, then the provider deferred selecting a task.

4.2 Experiment Topologies and Workload Generation

We defined each simulated configuration to consist of a selected topology of sites for directory servers, clients and providers, where all providers in a given configuration adopted identical provider-decision strategies. We subjected each configuration to varying workloads; thus, a configuration and a workload comprise one data point (failure-related experiments also defined a pattern of outages). To provide statistically significant comparisons among provider strategies, we replicated each data point at least 100 times, using a different stream of random numbers for each repetition. The random numbers provided differences in distance between nodes, location of providers with various capabilities, task workloads, and where applicable, failure start times and durations. To ensure that provider strategies experienced the same set of conditions, we repeated the same stream of random numbers for each repetition of each simulated configuration. Executing these experiments required around 50,000 simulation runs that took about two months of processing time when spread across 16 processors. The remainder of this section defines the conditions (i.e., parameters and associated values or value ranges) we used across all experiment scenarios.

For each experiment repetition, we generated a random topology by varying (uniformly on an 8,000 × 8,000 grid) the x–y coordinates defining the location of each site, which limited maximum inter-site distance to 16 router hops. Each topology consisted of 43 sites: 30 provider sites, 10 client sites and three directory sites. Table 1 denotes global constants used across all repetitions. We simulated operation of a typical 8-h workday (L_{sys}=28,800 s). Table 2 defines statistical distributions from which various experiment parameters (discussed below) were selected.

Each provider managed a single cluster of processors, where each processor operated at the same simulated speed but with varying costs. For homogeneous scenarios, all clusters had the same number of processors. For heterogeneous scenarios, clusters had varying numbers of processors, as described below (Section 4.3). Each provider site contained local directory servers, which injected service descriptions into three higher-level directory servers, accessible to clients. For each provider, we randomly selected (see

Table 1 Global experiment constants

Symbol	Explanation	Value
D_{avg}	Average global task deadline for all tasks in the system	21,960 s
E_{day}	Time at which 8-h day ends	28,800 s
L_{sys}	System lifetime (duration of an 8-h day)	28,800 s
u_{exp}	Expected provider utilization	0.7
pL_{Avg}	Average task length	4,320 s

Table 2 Definition and parameters for key distributions

Symbol	Name	Distribution	Parameters and constraints
pC_{cyc}	Cycle cost	Normal	μ=0.001, σ=0.001, 0.0009≤pC_{cyc}≤0.0011
m_{min}	Minimum profit margin	Normal	μ=0.175, σ=0.0875, 0.15≤m_{min}≤0.2
m_{trg}	Target profit margin	Normal	μ=(0.3−m_{min})/2, σ=(0.3−m_{min})/4, (0.3−m_{min})/2≤m_{trg}≤0.3
pL_{tsk}	Task length	Normal	μ=4320 s, σ=1,500 s
S_{tsk}	Task start time	Exponential	μ=3,600 s, S_{tsk}<6,221 s
D_{tsk}	Task target deadline	Normal	μ=D_{avg}, σ=3,420 s, (E_{day}−D_{avg})/2
M_{tsk}	Task target deadline	Normal	μ=E_{day}, σ=1,710 *s*, (E_{day}−D_{avg})/4
F_{time}	Failure start time	Exponential	μ=3,600 s, 1,800 s<F_{time}<7,200 s
F_{dur}	Failure duration	Exponential	μ=3,600 s, F_{dur}<1,800 s

Table 2) a cycle cost pC_{cyc} and minimum profit margin m_{min}. We assumed all providers (see Table 1) had an expected utilization u_{exp} of 70%. After using (5) to compute the provider cost c_{prv}, we used (6) to determine the provider's base price b_{base} and initial price schedule. The price monitor then periodically adjusted asking price as described above (Section 3.2).

Each task was defined by a required number of processors (pF_{tsk}) and a required number of cycles (pC_{tsk}). Guided by workload studies [32, 33], we chose an average task length (pL_{Avg}) of 4,320 s. For each task, we randomly selected task lengths (pL_{tsk}) and widths (pF_{tsk}) and then transformed them into the required number of cycles (pC_{tsk}). The system was calibrated so that, for the homogeneous scenario, allotting one average task per client (ten tasks total) generated an average system load of 5%. Thus, for every 5% increase in system load, each client created an additional task. For heterogeneous scenarios, a comparable workload required three times as many tasks (e.g., 30 tasks for each 5% load) because the addition of larger providers increased the number of available processors. To simulate a workday, we allowed clients to submit their tasks at an exponentially distributed random time (S_{tsk}) within the first three hours of the day. We chose this approach to simulate the arrival of a competing workload of tasks early in a workday.

Prior to task arrivals, each client conducted an initial round of discovery to obtain a set of available providers. The discovery process iterated throughout a workday to detect changes in availability of provider resources. Upon arrival, a task used the client's list of available providers to determine which providers seemed suitable for negotiation and then requested more detailed information from those providers. Based upon this more detailed information, the task selected providers with which to negotiate. If the task concluded successful negotiation with a provider, then the task awaited the outcome of its execution. If task execution failed or if the task was unable to conclude successful negotiation, then the task found another set of suitable providers and iterated the negotiation process. This iterative negotiation continued until the task was completed successfully or failed to meet its maximum deadline (M_{tsk}) or until the simulation ended.

Each task also had a set of other associated characteristics. The task cost (c_{tsk}) was calculated from (3). Having randomly selected the target (m_{trg}) and minimum (m_{min}) profit for each client, we used (4) to calculate the maximum task revenue (r_{tsk}) and to derive the maximum and minimum task budgets. We included a penalty rate in order to use penalty terms concepts from WS-Agreement. For all tasks, we set a small fixed value ($p_{tsk} = 0.05 \times r_{tsk}$) because in our experiments we chose not to examine the effect of penalty rate, which is a separate area of investigation that we leave for future work. Following previous studies of computational Grid economies [11–15], we randomly designated 20% of tasks to have high value and increased their maximum revenues by a factor of five. By increasing budgets by a corresponding factor, clients were allowed to offer more money in hopes of completing high-value tasks sooner. For each task, we randomly chose a task target end time (D_{tsk}) using an average global task deadline, D_{avg}=21,960 s, representing the time by which all tasks could complete under optimal conditions assuming u_{exp}=0.7. Similar to previous studies [11–15], we designated a subset (20%) of tasks as high-urgency; their target and maximum deadlines were made an average of 20%

earlier (with an equivalent number of other tasks equally delayed). Finally, for each task, we computed a decay rate k_{tsk} from (2). The value of k_{tsk} was proportionally increased by a factor of five for high-value tasks. Using these techniques to generate workload ensured a mix of four distinct task types: 4% high-value, high-urgency, 16% high-value, low-urgency, 16% low-value, high-urgency and 64% low-value, low-urgency.

4.3 Experiment Scenarios

We defined three experiment scenarios (homogeneous, heterogeneous and failures-prone), where each scenario compared six (or eight) provider strategies under increasing workload (in 5% increments up to 200%). In the homogeneous scenario, every provider offered a 500-processor cluster (i.e., pF=500) and every client task required 500 processors (i.e., pF_{tsk}= 500). In the heterogeneous scenario, we varied the size of provider clusters as well as the number of processors required for each task; however, we chose to limit the amount of heterogeneity in this study in order to produce results that could be analyzed and verified reasonably. Guided by previous research [34], we selected 20% of the providers as large clusters with 5,500 processors each; the remaining clusters had 500 processors each. Consistent with our choice of cluster capacities, we defined four task sizes: 50% of tasks required 250 processors, 30% of tasks required 500 processors, 10% of tasks required 750 processors and 10% of tasks required 1,500 processors.

For the failures-prone scenario, we aimed to investigate how well various task admission and selection strategies would perform in the face of large failures, such as distributed denial of service attacks and power outages, which cause entire provider sites to fail. We selected such scenarios because power outages and denial of service attacks are increasing threats [20, 21]. For the failures-prone scenario, we repeated the conditions of the heterogeneous scenario but introduced the possibility that providers could fail and then recover. As specified in Table 2, we randomly failed providers with a probability of 0.25 (uniformly distributed) for an average duration of 3,600 s (exponentially distributed)—but each failure lasted at least 1,800 s. Each failure began at a random (exponentially distributed) time with a mean of 3,600 s from the start of each simulated day; however, no failures began in the first 1,800 s or after the first 7,200 s of a simulated day. For tasks already accepted on nodes that suffered a failure, clients were notified of the failure and were thus able to seek other service providers. We chose to delay the start of failures in order to allow time for some tasks to have concluded successful negotiations and entered execution. We chose to limit the period during which failures occurred in order to allow time for affected tasks to locate and negotiate with alternate providers. The results of our three scenarios are discussed next.

5 Results and Discussion

For purposes of analysis, we compared simulation results in three load ranges: 5–45% (insufficient demand), 50–100% (expected demand) and 105–200% (excessive demand). Reflecting this comparison, results shown in each table (Tables 3, 4, 5, 6, 7, 8) are divided accordingly. A pair of tables gives results for each scenario: homogeneous (Tables 3 and 4), heterogeneous (Tables 5 and 6) and failures-prone (Tables 7 and 8). In each pair of tables, the first table presents system-related metrics and the second shows

Table 3 System-related metrics for the homogeneous scenario

Load range	Strategy	S	B_{prv}	B_{cli}	P_{fin}	C
5–45%	NetVal	−0.101	−0.355	0.258	0.986	5
	NetPro	−0.086	−0.348	0.263	0.996	5
	SJF	−0.108	−0.363	0.257	0.993	5
	FCFS	−0.092	−0.351	0.259	1.000	4
	Rand	−0.146	−0.408	0.268	0.972	6
	NoCon	−0.158	−0.396	0.244	0.965	5
50–100%	NetVal	0.602	0.438	0.229	0.906	7
	NetPro	0.619	0.442	0.233	0.921	6
	SJF	0.566	0.378	0.225	0.939	9
	FCFS	0.627	0.415	0.230	0.973	8
	Rand	0.460	0.313	0.216	0.871	8
	NoCon	0.369	0.259	0.193	0.819	8
105–200%	NetVal	0.435	0.508	0.179	0.631	40
	NetPro	0.421	0.494	0.180	0.622	62
	SJF	0.309	0.312	0.146	0.663	51
	FCFS	0.328	0.325	0.143	0.684	52
	Rand	0.277	0.287	0.145	0.633	13
	NoCon	0.227	0.244	0.138	0.588	13

Table 4 Causal metrics for the homogeneous scenario

Load range	Strategy	P_{Adm}	P_{Can}	P_{Acc}	Completion probability	
					High value	Low value
5–45%	NetVal	1.000	0.014	0.924	0.999	0.983
	NetPro	1.000	0.003	0.720	0.999	0.996
	SJF	1.000	0.007	0.938	0.991	0.993
	FCFS	1.000	0.000	0.939	1.000	1.000
	Rand	1.000	0.027	0.500	0.978	0.971
	NoCon	1.000	0.034	1.000	0.964	0.966
50–100%	NetVal	0.994	0.088	0.472	0.996	0.883
	NetPro	0.948	0.027	0.107	0.997	0.903
	SJF	0.980	0.040	0.397	0.941	0.939
	FCFS	0.973	0.000	0.387	0.974	0.973
	Rand	1.000	0.129	0.500	0.885	0.868
	NoCon	1.000	0.181	1.000	0.817	0.820
105–200%	NetVal	0.822	0.191	0.016	0.990	0.541
	NetPro	0.686	0.063	0.006	0.992	0.530
	SJF	0.734	0.070	0.009	0.664	0.663
	FCFS	0.684	0.000	0.008	0.682	0.684
	Rand	1.000	0.367	0.500	0.646	0.630
	NoCon	1.000	0.412	1.000	0.589	0.87

Table 6 Causal metrics for the heterogeneous scenario

Load range	Strategy	P_{Adm}	P_{Can}	P_{Acc}	Completion probability	
					High value	Low value
5–45%	NetVal/D	1.000	0.002	0.335	0.996	0.999
	NetPro/D	0.999	0.001	0.286	0.997	0.998
	NetVal	1.000	0.018	0.857	0.993	0.980
	NetPro	0.999	0.006	0.557	0.996	0.993
	SJF	1.000	0.013	0.923	0.987	0.987
	FCFS	1.000	0.006	0.928	0.993	0.993
50–100%	NetVal/D	0.984	0.030	0.040	0.969	0.951
	NetPro/D	0.940	0.010	0.030	0.969	0.920
	NetVal	0.991	0.085	0.297	0.953	0.894
	NetPro	0.954	0.033	0.070	0.961	0.911
	SJF	0.982	0.044	0.392	0.938	0.938
	FCFS	0.981	0.026	0.389	0.954	0.955
105–200%	NetVal/D	0.847	0.077	0.008	0.888	0.742
	NetPro/D	0.724	0.028	0.006	0.888	0.648
	NetVal	0.837	0.171	0.019	0.838	0.623
	NetPro	0.738	0.067	0.008	0.845	0.628
	SJF	0.742	0.064	0.011	0.675	0.678
	FCFS	0.726	0.042	0.010	0.681	0.685

causal metrics. For each load range in each table, results compare the performance of various provider-decision strategies for admission and task selection. To highlight relative differences in overall performance, we plot system welfare (Figs. 2, 3, 4) achieved by the provider-decision strategies for each scenario; however, we graph only the 50–200% load range, which covers expected and excessive demand.

Table 5 System-related metrics for the heterogeneous scenario

Load range	Strategy	S	B_{prv}	B_{cli}	P_{fin}	C
5–45%	NetVal/D	−0.046	−0.331	0.286	0.998	5
	NetPro/D	−0.044	−0.332	0.288	0.998	5
	NetVal	−0.065	−0.345	0.284	0.982	4
	NetPro	−0.049	−0.337	0.290	0.993	4
	SJF	−0.076	−0.354	0.282	0.987	4
	FCFS	−0.066	−0.348	0.283	0.994	5
50–100%	NetVal/D	0.616	0.412	0.235	0.954	16
	NetPro/D	0.592	0.402	0.236	0.930	19
	NetVal	0.569	0.397	0.233	0.906	7
	NetPro	0.581	0.396	0.237	0.921	13
	SJF	0.564	0.374	0.228	0.938	7
	FCFS	0.589	0.388	0.230	0.955	7
105–200%	NetVal/D	0.473	0.442	0.170	0.771	47
	NetPro/D	0.422	0.434	0.169	0.696	53
	NetVal	0397	0.422	0.170	0.666	32
	NetPro	0.392	0.411	0.168	0.671	47
	SJF	0.326	0.325	0.148	0.678	43
	FCFS	0.336	0.335	0.148	0.684	43

Table 7 System-related metrics for the failures-prone scenario

Load range	Strategy	S	B_{prv}	B_{cli}	P_{fin}	C
5–45%	NetVal/D	0.216	−0.054	0.274	0.991	6
	NetPro/D	0.213	−0.058	0.277	0.988	7
	NetVal	0.190	−0.070	0.274	0.969	5
	NetPro	0.207	−0.063	0.278	0.983	5
	SJF	0.179	−0.082	0.270	0.980	4
	FCFS	0.190	−0.075	0.270	0.988	4
50–100%	NetVal/D	0.577	0.436	0.209	0.893	23
	NetPro/D	0.539	0.429	0.209	0.849	27
	NetVal	0.519	0.423	0.208	0.822	13
	NetPro	0.519	0.417	0.208	0.829	22
	SJF	0.489	0.377	0.198	0.848	16
	FCFS	0.509	0.390	0.198	0.862	16
105–200%	NetVal/D	0.371	0.418	0.147	0.653	54
	NetPro/D	0.320	0.405	0.145	0.578	60
	NetVal	0.297	0.387	0.145	0.553	43
	NetPro	0.289	0.374	0.142	0.554	55
	SJF	0.214	0.264	0.119	0.548	54
	FCFS	0.217	0.268	0.118	0.549	55

Table 8 Causal metrics for the failures-prone scenario

Load range	Strategy	P_{Adm}	P_{Can}	P_{Acc}	Completion probability	
					High value	Low value
5–45%	NetVal/D	0.999	0.008	0.243	0.991	0.992
	NetPro/D	0.994	0.006	0.206	0.991	0.987
	NetVal	1.000	0.031	0.720	0.987	0.964
	NetPro	0.997	0.013	0.387	0.989	0.982
	SJF	0.999	0.020	0.775	0.980	0.979
	FCFS	0.999	0.012	0.775	0.987	0.988
50–100%	NetVal/D	0.952	0.059	0.026	0.938	0.882
	NetPro/D	0.885	0.039	0.020	0.939	0.823
	NetVal	0.965	0.143	0.106	0.910	0.800
	NetPro	0.906	0.077	0.033	0.918	0.807
	SJF	0.938	0.089	0.089	0.847	0.849
	FCFS	0.935	0.073	0.086	0.861	0.862
105–200%	NetVal/D	0.760	0.107	0.008	0.844	0.605
	NetPro/D	0.648	0.070	0.006	0.835	0.514
	NetVal	0.780	0.227	0.012	0.763	0.501
	NetPro	0.674	0.121	0.007	0.771	0.499
	SJF	0.686	0.138	0.007	0.545	0.549
	FCFS	0.672	0.124	0.007	0.545	0.549

5.1 General Findings

System welfare curves have the same general shape across all scenarios and provider strategies. System welfare (S) rises through the range of insufficient demand (not shown in Figs. 2, 3, 4), reaching a peak somewhere within the range of expected demand (usually between 70 and 80%) and then declines with excessive demand. Under insufficient demand, average system welfare was negative because there was insufficient work to allow providers to cover their costs (B_{prv} is negative). Nearly all jobs were completed (P_{fin}), communication costs (C) were low and clients received a significant proportion of their possible benefits (B_{cli}). What sets the strategies apart is the peak value achieved for system welfare under expected demand and the rate of decline with increasing excess demand.

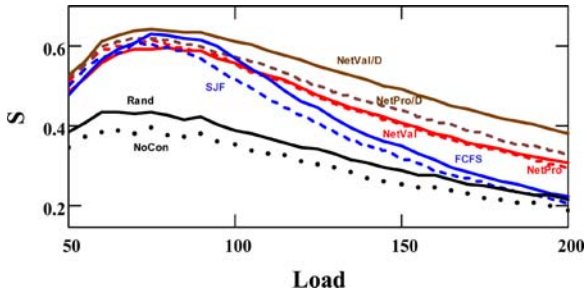

Fig. 3 System welfare (S) for eight resource-allocation strategies during periods of expected (50–100) and excessive (105–200) load under a heterogeneous scenario

Within the expected range of demand, the utilization-based and economics-based strategies achieved similar peak system welfare (S). Among the strategies discussed in Section 4.1, the FCFS strategy achieved highest peak performance, followed by NetPro. Providers using FCFS accepted all offers until resources were fully committed and then accepted no further offers. This ensured a high rate of task completions (P_{fin}) and few cancellations (P_{Can}), but gave no special priority to high-value tasks. NetPro allowed providers to reject offers (low P_{Acc}) that did not appear profitable, which enabled acceptance and completion of more high-value tasks at the cost of finishing fewer low-value tasks. Accepting fewer offers also increased communication costs (C) for NetPro. The NetVal and SJF strategies did not fair quite as well due to lower task-completion rates (P_{fin}), arising from a higher rate of cancellations (P_{Can}). SJF prioritized shorter tasks

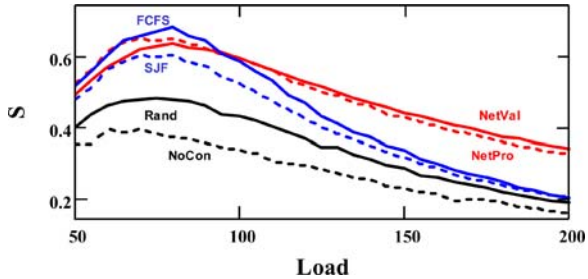

Fig. 2 System welfare (S) for six resource-allocation strategies during periods of expected (50–100) and excessive (105–200) load under a homogeneous scenario

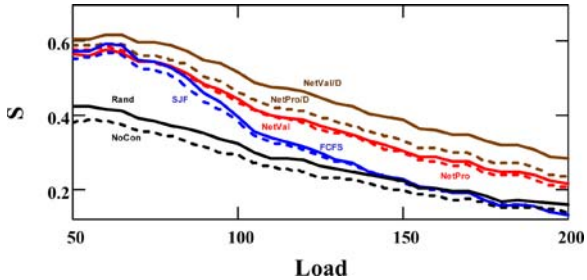

Fig. 4 System welfare (S) for eight resource-allocation strategies during periods of expected (50–100) and excessive (105–200) load under a failures-prone scenario

over longer tasks, which increased the probability of cancellations (P_{Can}) for longer tasks and thus lowered the provider benefits (B_{prv}) available from more expensive jobs. NetVal aggressively accepted higher-value tasks, which led to an over commitment of resources, and subsequent cancellations. As expected, the two information-free strategies yielded the worst peak system welfare.

The economics-based strategies (described in Section 4.1) excelled under increasing excessive demand—yielding higher benefits for both providers (B_{prv}) and clients (B_{cli}) and completing almost as many tasks (P_{fin}) as the utilization-based strategies and for comparable communication costs. The two utilization-based strategies declined most steeply with increasing excess demand. In fact, when demand became sufficiently high, system welfare provided by the utilization-based strategies converged with that achieved by the information-free strategies. Overall, the utilization-based strategies failed to adapt to an increasing number of high-value tasks.

Despite achieving similar system welfare under excess demand, the NetPro and NetVal strategies showed quite different behaviors. NetPro providers refused to accept new tasks unless the profits gained by the tasks more than offset revenue lost from delaying already accepted tasks. This appeared as a lower task-admission rate (P_{Adm}) coupled with a lower cancellation rate (P_{Can}). On the other hand, NetVal providers accepted tasks whenever the revenue gained more than offset revenue lost from delaying currently pending tasks. Thus, NetVal providers both admitted and canceled significantly more tasks than NetPro providers. By admitting more tasks, NetVal providers also reduced the system-wide communication costs.

5.2 Homogeneous Scenario

The homogeneous scenario was unrealistic because all tasks required the same number of processors, which exactly matched the number of processors offered by each provider. We considered this scenario because it was easy to analyze and illustrated the main behavioral differences among provider-decision strategies. Figure 2 reveals that the economics-based strategies are able to extract higher system welfare during periods of excessive demand. This occurs because NetVal and NetPro are both able to complete a higher proportion of high-value tasks at the expense of completing fewer low-value tasks, which yields substantially larger benefits for both providers and clients. This is illustrated in Table 4, where the utilization-based and information-free strategies complete a balanced proportion of high-value and low-value tasks throughout all ranges of demand, while the economics-based strategies complete nearly all high-value tasks regardless of demand.

Within the range of expected demand, FCFS yielded superior system welfare and achieved a higher peak than either economics-based strategy (see Fig. 2). The primary advantage (see Table 3) is that FCFS completed more tasks overall (P_{fin}) and substantially more low-value tasks (see Table 4). FCFS ensured that processor resources were never over allocated; thus, tasks were not canceled no matter what the demand. Instead, the task-admission rate (P_{Adm}) dropped, along with the probability of accepting offers (P_{Acc}; see Table 4). Though SJF uses an identical strategy for admission, cancellations rose for longer tasks because shorter tasks were executed first; thus, both the completion rate (P_{fin}) and provider benefit (B_{prv}) for SJF fell below that for FCFS. Economics-based providers admitted (P_{Adm}) more tasks, because higher-value tasks could be substituted for lower-value tasks, which led to an increased rate of cancellations (P_{Can}) for low-value tasks (which exhibited a lower completion rate). The increased rate of cancellations was compensated for by an increased rate of completion for high-value tasks, which enabled the economics-based strategies to achieve better provider benefits (B_{prv}) than FCFS. Unfortunately, increased cancellations lowered the completion rate for the economics-based strategies. Given excess demand, acceptance rate (P_{Acc}) among economics-based and utilization-based providers achieved comparability, which indicates that most of the communication costs arose from tasks that were not accepted by any provider.

Beyond an overall similarity in system welfare, the behavior of NetPro and NetVal were quite different in one aspect. NetPro admitted fewer tasks (P_{Adm}) than NetVal, which led NetPro to cancel fewer tasks (P_{Can}). Since task revenues were much larger than task profits, NetVal was more likely to accept an offer than NetPro. Consequently, more previously admitted tasks had to be cancelled. When NetPro rejected offers those tasks were free to seek other providers.

Of course, the increased rejection rate by NetPro increased communication costs (C), as rejected tasks worked harder to find appropriate, available providers.

The 19% cancellation rate (P_{Can}) for NetVal under excessive demand could prove unacceptable in a real marketplace. A better outcome might be achieved if NetVal were modified to accept fewer low-value tasks in any given time period, which would allow those tasks to seek other (less loaded) providers in the same time period, or to return to rejecting providers in a future time period when more resources might be available. This insight formed the basis for variants of NetVal and NetPro. We introduce these variants below, as we discuss the heterogeneous scenario.

5.3 Heterogeneous Scenario

The heterogeneous scenario introduced variation into workloads (tasks required varying numbers of processors) and resource availability (20% of service providers offered large clusters with 5,500 processors). In addition, we considered variants of NetPro and NetVal that included an additional level of filtering before accepting a low-value task that would otherwise be accepted under the normal procedures. The filter accepted a low-value task only when the existing queue of low-value tasks did not require all of the processors available at the provider. This filter had the effect of *deflecting* some low-value tasks to other, less-loaded providers or until some later time. The intended result of this filter was to reduce the number of low-value tasks that were accepted and then canceled. We denote these variants as NetPro/D and NetVal/D.

In the main (see Fig. 3), the heterogeneous scenario exhibited the same relationships among the strategies previously compared in the homogeneous scenario. The NetPro and NetVal strategies extracted higher system welfare under excessive demand because they gave priority to high-value tasks. The utilization-based strategies achieved good system welfare within the range of expected demand and then declined under excessive demand to the point where they achieved no better results than the information-free strategies, which perform worst overall. The FCFS strategy achieved higher peak system welfare than either NetPro or NetVal, and SJF remains less competitive. The reasons for these relative differences were the same as discussed for the homogeneous scenario. Note, however, that the heterogeneous scenario led to overall lower system welfare across all strategies than was measured under the homogeneous scenario. This lower performance stemmed from a combination of varying tasks sizes and the use of first-fit backfilling.

The smaller clusters (500 processors) tended to delay 500-processor tasks. This occurred whenever a 500-processor task could not fit (i.e., a 250-processor task was executing) and one or more 250-processor tasks were available in a provider's queue. A 250-processor task would then be elevated to run ahead of the blocked 500-processor task and the delay would be perpetuated as each executing 250-processor task finished and the queue contained additional 250-processor tasks. This led 500-processor tasks to be canceled more frequently than tasks with other sizes, which lowered provider benefits (compare Tables 3 and 5).

The two new variants (NetPro/D and NetVal/D) of the economics-based strategies yielded significant improvement in system welfare, most particularly under excessive demand. In fact, NetVal/D achieved the highest peak system welfare, besting FCFS. The NetPro/D variant showed significant improvement over NetPro when the system experienced excess demand. Neither NetVal/D nor NetPro/D changed the fundamental nature of their corresponding base strategy (NetVal and NetPro). As a result, NetVal/D showed a higher admission rate (P_{Adm}) and also a higher rate of offer acceptance (P_{Acc}) than NetPro/D. The key to the success of NetPro/D and NetVal/D was their refusal to accept too many low-value tasks, which composed the highest proportion of the workload. This forced tasks to try other providers, or to return to rejecting providers later when those providers might have more resource availability. As a result, tasks had a lower chance of being cancelled (P_{Can}). These factors increased the probability of job completion (P_{fin}) and the provider benefit (B_{prv}), which together raised system welfare (S). Of course, NetPro/D and NetVal/D incurred higher communication cost (C) because the increased rate of rejections stimulated more tasks to continue discovering and negotiating with providers for a longer time.

The most notable difference between NetVal/D and NetPro/D appeared in the form of task completions (P_{fin}), included as a measure of social welfare. NetVal/D completed a significantly higher portion of

tasks than did NetPro/D. These additional tasks were taken from the set of low-value tasks that remained unaccepted near the end of a typical workday. Such tasks might still have time to meet their maximum deadlines, if some provider would accept them—and many providers could be idle near the end of a workday. Of course, late tasks would decay in revenue substantially because time was running short. When NetPro/D computed the profit (decayed revenue minus cost) for a late task, the net was negative; thus, the task was not accepted. On the other hand, NetVal/D would accept such tasks whenever they could be completed in time to garner revenue. As a result, NetVal/D accepted low-value tasks late in the day and ran them at a loss (i.e., the decayed revenue did not cover the provider's cost). Running these late tasks at a loss substantially boosted the rate of task completions; thus, NetVal/D produced an increased social benefit.

5.4 Failures-Prone Scenario

The failures-prone scenario created a challenging environment for all provider-decision strategies. Sudden outages led to an increase in task cancellations (compare Tables 6 and 8). The occurrence of outages early in the workday created an effective drop in resource availability and increased relative demand. As a result (see Fig. 4), peak system welfare was left-shifted by between 0.1 and 0.2. For the expected range of demand, average system welfare dropped by about 0.05 for economics-based strategies and 0.08 for utilization-based strategies. Comparing Tables 5 and 7 reveals that this decrease was due largely to declines in client benefit (B_{cli}) and completion probability (P_{fin}), which both showed greater decline for utilization-based strategies than for economics-based strategies. On the other hand, provider benefit (B_{prv}) actually increased slightly. Providers that were unaffected by failures could absorb some of the additional demand at higher average prices. Economics-based strategies were able to discriminate in favor of higher-valued tasks and thus were able to realize a larger price increase (16–18%) than the 5% increase realized by utilization-based strategies. Under excess demand, average system welfare decreased by about 0.10 for economics-based strategies and between 0.11 and 0.12 for utilization-based strategies. Since the range of demand exceeded provider capacity, all three components of system welfare (B_{prv}, B_{cli}, and P_{fin}) declined. Despite the general deterioration in global performance, relative rankings and behaviors of all strategies remained the same as seen in the heterogeneous scenario.

Why does system welfare decline less for economics-based strategies? Recall from Table 2 that failures occur early—between 30 and 120 minutes after the beginning of the workday. Careful analysis of the distribution of task admission times (not given here) shows that utilization-based providers admit (on average) about 95% of tasks within the first 117 min of the workday. In contrast, economics-based providers admit fewer tasks within this period. The strategies with deflection filters, NetPro/D and NetVal/D, admit around 80% and 70% of tasks, respectively, while NetPro admits about 85% and NetVal admits about 90%. In addition, economics-based strategies can potentially admit higher-value tasks even when a provider's capacity is saturated. Thus, in the failures-prone scenario, economics-based strategies have advantages when clients need to reschedule tasks aborted by failed providers. First, economics-based providers that do not fail are likely to be less loaded than utilization-based providers during the first few hours of the day and hence in better position to accept new tasks. Second, even when saturated, economics-based providers are able to give priority to high-value tasks. These advantages mitigate the decline in provider benefit (B_{prv}), client benefit (B_{cli}) and probability of task completion (P_{fin}). This suggests economics-based decision strategies can significantly improve system welfare when some providers experience outages during key periods. Further, experiments are needed, using a variety of failure scenarios, to confirm whether economics-based decision strategies could yield widespread improvement in fault-tolerance.

5.5 Discussion of Key Factors in Economics-based Strategies

While all the economics-based strategies proved adept at prioritizing high-value tasks, treatment of low-value tasks (80% of the workload) differed markedly. Our results showed that the NetPro and NetVal strategies provided similar completion rates for low-value tasks; however, this similarity arose from differing behaviors. NetPro was less likely to accept offers; thus, fewer tasks were admitted. This led to fewer cancellations and drove up communi-

cation costs, as tasks that could not gain acceptance continued to search for providers until task deadlines could not be satisfied. NetVal was more likely to accept offers; thus, more tasks were admitted. This led to more cancellations and also reduced communication costs, as accepted tasks sat in provider queues until they could be executed or their deadlines expired. This suggests that one might try to modulate acceptance and cancellations for low-value tasks by tuning some key parameters. For example, one might require that profit (or revenue) increase from a task exceed revenue loss by some factor before accepting a task—or one might increase the cancellation penalty in order to reduce the task acceptance probability. We found that tuning these parameters of the economics-based strategies did not alter temporal behavior. Instead, algorithm changes were required.

The use (in NetPro/D and NetVal/D) of a deflection filter for low-value tasks mitigated the consequences of a provider accepting too many low-value tasks within a given time period. Given that NetVal was more likely to accept low-value tasks, the deflection filter proved more helpful than was the case for NetPro. Further, temporal deflection had an amplifying effect in the case of NetVal, which was more likely to admit previously unaccepted jobs near the end of a workday. These jobs, which were run at a loss, improved the societal (completion rate) component of the system welfare metric. This suggests that deferring some decisions may prove helpful in regimes where agreements are binding. This also suggests that willingness to run some tasks at a loss could improve benefits to society.

In the task model used in this paper, client and provider benefits were related quite closely. Client revenues were subjected to the same piecewise linear utility function as provider revenues; thus, clients and providers had a joint stake in completing tasks in a timely manner. Within this framework, clients and providers struggled over who extracted more profits from the tasks. In general, clients extracted higher profit when there was an excess supply and providers extracted higher profit under excess demand. This cooperative-competitive model likely has a key influence on the global behavior of the system under economics-based resource allocation. One could investigate alternate models where the fortunes of the clients and providers were not so tightly coupled.

6 Future Work

Future work remains to extend investigation of computational economies using our model. One area of interest is increasing the variety of provider types and the mix of task characteristics (including lengths, widths, values and urgencies) to ensure that our findings hold broadly. Of additional interest is assessing influence of provider penalty rate on system welfare under economics-based decision strategies. Conceived more generally, future work might investigate the sensitivity of our model to a wide range of factors, including also choice of backfilling algorithm and variations in processor speeds. Advanced methods of experiment design using orthogonal fractional factorial approaches [35] suggest, given sufficient processor availability to conduct 2^{16} simulation runs, sensitivity analysis using a $2^{40\text{-}24}$ design seems feasible. With insights gained from such an analysis, one might be able to reduce the simulation model to a smaller set of key factors and then to recast those key factors into an analytical model.

Given either our current model or a simpler model derived from sensitivity analysis, selected parameter values should be motivated by data collected from real systems. In our study, we adopted such values where available. Additional work remains to collect a wider range of measurement data, especially from systems that use economic bases to allocate resources. The characteristics of input parameters from such systems could be used to establish reasonable values for model factors, such as penalty rates. Behavioral and performance measures from such systems could also be used to validate model behavior against empirical data.

Another area for future work is motivated by growing interest in workflows [36] and related co-allocation [37] of resources, which suggests that our experiments should be extended to consider how well economics-based strategies apply in these more complex situations. Additionally, future work is needed to investigate the properties of economics-based resource allocation as system size expands to hundreds and thousands of competing providers serving a suitably large client population.

7 Conclusions

We simulated a computational economy where individual providers and consumers pursued their own interests and where providers used alternate admission and task-selection strategies chosen from three classes: information-free, utilization-based and economics-based. Our experiments compared how system welfare differs with provider-decision strategy given varying customer demands. We found that, during periods of excess demand, economics-based strategies yield overall resource allocation that benefits system welfare. Further, economics-based strategies responded well to sudden overloads caused by temporary provider failures. During periods of moderate demand, we found that economics-based strategies provided ample system welfare, comparable with that of utilization-based strategies.

The findings reported in this paper expand significantly on the scope and realism of previously reported research into the use of economic mechanisms to allocate computation resources. We modeled a distributed marketplace through interactions of competing providers and consumers. We defined a metric for system welfare that aggregates the interests of all consumers and providers together with the interest of society at large. We employed scenarios inspired by the likely requirements of commercial operations. We investigated market robustness in the face of sudden, temporary outages among selected providers. We demonstrated that emerging standards for web services and Grid services could support a computation marketplace.

References

1. Lundquist, E.: IBM bets $10 billion on 'on-demand' computing. eWeek.com (October 30, 2002)
2. Morgan, T.: Sun Grid compute utility opens for public business. The Unix Guardian (March 23, 2006)
3. Michelson, B.: StrikeIron's web services marketplace a glimpse of the future, available today. Report from Patricia Seybold Group (April 6, 2006)
4. Jeff Bezos' Risky bet. Business Week.com (November 13, 2006)
5. Seabright, P.: The Company of Strangers: A Natural History of Economic Life. Princeton University Press, Princeton (2006)
6. Landsburgh, S.: Price Theory & Applications, 6th edition. Thomson South-Western, OH (2006)
7. Mills, K., Dabrowski, C.: Investigating global behavior in computing Grids. Self-organizing systems. Lect. Notes Comput. Sci. **4124**, 120–136 (2006)
8. Organization for the Advancement of Structured Information Standards: Web site (http://www.oasis-open.org)
9. Open Grid Forum: Web site (http://www.ogf.org)
10. Web Services Agreement Specification (WS-Agreement). GFD.107, Open Grid Forum (May 2007)
11. Chun, B.N., Culler, D.E.: User-centric performance analysis of market-based cluster batch schedulers. Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (May 2002)
12. Yeo, C.S., Buyya, R.: Service level agreement based allocation of cluster resources: handling penalty to enhance utility. Proceedings of the 7th IEEE International Conference on Cluster Computing. Boston (September 27–30, 2005)
13. Irwin, D.E., Grit, L.E., Chase, J.S.: Balancing risk and reward in a market-based task service. Proceedings of 13th IEEE Symposium on High Performance Distributed Computing (June 2004)
14. Popovici, F.I., Wilkes, J.: Profitable services in an uncertain world. Proceedings of Supercomputing (November 2005)
15. AuYoung, A., et al.: Service contracts and aggregate utility functions. Proceedings of 15th IEEE International Symposium on High Performance Distributed Computing, pp. 119–131 (2006)
16. Wolski, R., et al.: Grid resource allocation and control using computational economies. In: Berman, F., Fox, G., Hey, T. (eds.) Grid Computing: Making the Global Infrastructure a Reality, pp. 747–772. Wiley, New York (2003)
17. Gomoluch, J., Schroeder, M.: Market-based resource allocation for Grid computing: a model and simulation. Proceedings of the First International Workshop on Middleware for Grid Computing. Rio de Janeiro, pp. 211–218 (June 16–20, 2003)
18. Vengerov, D.: Adaptive utility-based scheduling in resource-constrained systems. Proceedings of the 18th Australian Joint Conference on Artificial Intelligence. LNCS, Vol. 3809, Sydney, pp. 477–488 (December 5–9, 2005)
19. Avraham, L., Rayfield, J.T., Dias, D.M.: Service-level agreements and commercial Grids. IEEE Internet Computing, 44–50 (July–August 2003)
20. Technical Analysis of the August 14, 2003, Blackout: What Happened, Why, and What Did We Learn? Report to the North American Electric Reliability Council (NERC) Board of Trustees (July 13, 2004)
21. Gregg, D.M., et al.: Assessing and quantifying denial of service attacks. Proceedings of the Military Communications Conference, pp. 76–80 (MILCOM 2001 Vol. 1)
22. Smale, S.: Price adjustment and global Newton methods. Frontiers of Quantitative Economics **IIIA**, 191–205 (1975)
23. Dabrowski, C.: Investigating resource allocation in a standards-based Grid compute economy. NISTIR 7463. National Institute of Standards and Technology, October 2007
24. SOAP V1.2 Part 1: Messaging Framework. W3C Recommendation (June 24, 2003)
25. WS Addressing. BEA Systems, International Business Machines Corporation and Microsoft Corporation (March 2004)

26. The WS Resource Framework, V 1.0, Computer Associates International, Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation and the University of Chicago (2004)
27. Foster, I., et al.: A globus primer describing globus toolkit version 4, Draft May 8, 2005
28. WS Service Group, V 1.0, Computer Associates International, Fujitsu Limited, Hewlett-Packard Development Company, International Business Machines Corporation and the University of Chicago (March 2004)
29. Czajkowski, K., et al.: SNAP: a protocol for negotiating service level agreements and coordinating resource management in distributed systems. Job Scheduling Strategies for Parallel Processing. Lect. Notes Comput. Sci. **2537**, 153–183 (2002)
30. Distributed Resource Management Application API Specification. V 1.0 Global Grid Forum (June 2002)
31. Hovestadt, M., Kao, O., Keller, A., Streit, A.: Scheduling in HPC resource management systems: queuing vs. planning. Job Scheduling Strategies for Parallel Processing. Lect. Notes Comput. Sci. **2862**, 1–20 (2003)
32. Shan, H., Oliker, L. Job superscheduler architecture and performance in computational Grid environments. Proceedings of the 2003 ACM/IEEE Conference on Supercomputing. Phoenix, p. 44 (November 15–21, 2003)
33. Parallel Workloads Archive. The Hebrew University of Jerusalem. http://www.cs.huji.ac.il/labs/parallel/workload/
34. Ernemann, C., Krogmann, M., Lepping, J., Yahyapour, R.: Scheduling on the top 50 machines. Proceedings of the 10th Job Scheduling Strategies for Parallel Processing, LNCS, Springer Berlin (3277), pp. 17–46 (2004)
35. Dey, A.: Orthogonal fractional factorial designs. Technometrics **29**, (3), 387–388 (1987) Aug.
36. Yu, J., Buyya, R.: A taxonomy of workflow management systems for Grid computing. J. Grid Comput. **3**, (3–4), 171–200 (2005) September
37. Czajkowski, K., Foster, I., Kesselman, C.: Resource co-allocation in computational Grids. Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8), Redondo Beach, CA, p. 37 (August 3–6, 1999)