UNIVERSITY OF CALIFORNIA
Santa Barbara

"Eliciting Honest Behavior on Computational Grids"

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Andrew Mutz

Committee in Charge:

Dr. Rich Wolski, Chair

Dr. Kevin Almeroth

Dr. Subhash Suri

December 2008

The Dissertation of
Andrew Mutz is approved:

Dr. Kevin Almeroth

Dr. Subhash Suri

Dr. Rich Wolski, Committee Chairperson

December 2008

"Eliciting Honest Behavior on Computational Grids"

by

Andrew Mutz

Acknowledgements

This dissertation would not exist without the efforts of many others. Any attempt to list the names of those whose actions supported this work, directly or otherwise, would necessarily be incomplete. There are four individuals, however, who cannot go unmentioned.

Without the leadership of my advisor, Rich Wolski, none of this would have been possible. His support and guidance for the last five years was unwavering. I thank him for his vision, his flexibility and his unending energy.

I thank my mother and father for decades of support and nurturing. They worked tirelessly for eighteen years to prepare me for the challenges of higher education, and then continued to support everything I've subsequently attempted.

I thank my brother for serving as an example of excellence worthy of emulation. Throughout these years I've turned to him as a sounding board for many of the ideas contained in this dissertation. I respect his opinions, technical and otherwise, to the point that if I'm unable to convince him of the merits of a new idea, then I believe it likely has none.

Curriculum Vitæ

Andrew Mutz

Education

2001 Bachelor of Science in Computer Science, University of California, Santa Barbara.

Experience

2000 – 2001 Intern, Citrix Online.

2001 – 2002 Software Engineer, Citrix Online.

2003 – 2008 Graduate Research Assistant, University of California, Santa Barbara.

June 2004 – September 2004 Intern, IBM Research.

Selected Publications

A. Mutz, R. Wolski, and J. Brevik: "Eliciting Honest Value Information in a Batch-Queue Environment," In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing.* , 2007.

Y. Kee, D. Nurmi, G. Singh, A. Mutz, C. Kesselman, and R. Wolski: "VGES: the Next Generation of Virtualized Grid Provi-

sioning," In *IEEE/IFIP International Workshop on End-to-end Virtualization and Grid Management.* , 2007.

A. Mutz and R. Wolski: "Efficient Auction-Based Grid Reservations using Dynamic Programming," In *Proceedings of the 20th ACM International Conference on Supercomputing.* , 2008.

Abstract

"Eliciting Honest Behavior on Computational Grids"

Andrew Mutz

During the last decade, a computing paradigm known as "grid" computing has seen a surge in research activity. Drawing inspiration from the electrical power grid, this paradigm is an approach to high-performance computing that seeks to serve a large number of users from multiple, geographically distinct computing centers. As grids exist today, users are competing for overcommitted resources. Without a well-designed mechanism to mediate the competing interests of users, the outcome can be chaotic and inefficient. Additionally, scheduling decisions are made based on user-submitted metadata, data that can be manipulated to increase a user's share of resources.

This dissertation explores the efficacy of auction-based schedulers as a means for mediating these competing interests. In particular, the use of auctions to incentivize honest disclosure of job metadata is investigated. We investigate this problem in the context of best-effort batch queues and reservation systems.

Contents

List of Figures

Chapter 1

Introduction

> *It is necessary for him who lays out a state and arranges laws for it to presuppose that all men are evil and that they will always act according to the wickedness of their spirits whenever they have free scope.*
>
> Niccolo Machiavelli (c. 1517)

1.1 Computational Grids: Current Practice

During the last decade, a computing paradigm known as "Grid" computing has seen a surge in research activity. Drawing inspiration from the electrical power grid, this paradigm is an approach to high-performance computing that seeks to serve a large number of users from multiple, geographically distinct computing centers. As they exist today, these centers are very large scale systems that provide computing power to support science research.

As a group, the scientists who use these resources have resource demands that are unpredictable and insatiable. The consequence of underprovisioned resources

is uncertain turnaround times, with jobs sometimes waiting weeks before executing. These users are competing for execution time and are generally not interested in the performance seen by others, and care primarily about user-centric metrics such as total turnaround time to job completion. This self-interested focus can lead users to game the system [13] in order to increase their share of resources.

Resource providers, on the other hand, are funded by government agencies and are tasked with purchasing and maintaining computing resources to support scientific research. In this role, these institutions are primarily concerned with maximizing the usefulness of their computing infrastructure to their user community. More specifically, they use social-welfare-type metrics such as system utilization to measure and optimize this usefulness. The infrastructure outlays involved are extremely expensive, and these institutions are interested in accounting for and maximizing the usefulness of these computing investments. Two examples of such institutions are the San Diego Computing Center (SDSC), and the National Center for Supercomputing Applications (NCSA).

Without a well-designed mechanism to mediate these competing interests, the outcome can be chaotic and inefficient. For example, if a time-slicing scheme were employed, and all users were given interactive access to the machines, the performance seen by each user would rapidly degrade as many users attempted to hoard resources through any means possible. A resource allocation mechanism

must be used to reconcile competing interests to avoid such chaos. Additionally, if the mechanism to mediate these interests can accurately measure the underlying user-centric value of the work being performed, we can use that information to expedite the most urgent and important jobs.

Grid systems today rely primarily on two mechanisms to allocate resources: best-effort batch queues, and reservation systems. Best-effort batch queues have been used to allocate resources since the mid 1980's. In their simplest form, batch queues consist of virtual lines that users wait in for access to computational resources. Users are given no guarantees regarding how soon their jobs will run: they may see immediate exection, or they may wait weeks. Reservations systems, on the other hand, are a more recent model in the high-performance computing world. Rather than wait in lines for an uncertain execution time, reservation systems allow a user to get a guaranteed time slot at a future date.

1.2 Thesis Qustion

In general, this dissertation looks at the problem of mediating the competing interests of grid users, and seeks to do so in a manner that retains much of the semantics of existing schedulers. But more specifically, there is a particular sub-

problem we are attempting to solve: can we do this by incentivizing users to be honest when interacting with the system?

In summary, this dissertation addresses the following Thesis Question:

> Can we design grid scheduling systems that reward users for being honest when submitting jobs, and can we do this while both retaining much of the semantics of existing schedulers and adhering to the computational tractability requirements of high-performance computing centers?

1.2.1 Why Honesty Matters

In both batch queue systems and reservation systems, users submit metadata along with jobs (both explicitly and implicitly) in order to help these systems make scheduling decisions. For example, the number of nodes needed by a job is explicit information included in a submission, and the user account under which it was submitted provides implicit information. Because this metadata influences scheduler decisions, users can manipulate scheduling outcomes to their advantage by manipulating the metadata they submit.

As an example, consider a scheduling system that allowed the specification of a *priority* variable, say between 1 and 10. In this system, jobs with higher stated priority would be scheduled for earlier execution than those with lower priority. If such a system had no negative consequences for stating a high priority, it would be the best interest of any user to indicate that their job was of maximum priority,

or "10" in this example. As a result, this metadata would become useless for scheduling purposes.

If we can get users to submit honest and accurate metadata, there would be numerous advantages. First, this would likely lead to more accurate scheduling decisions. More accurate data driving scheduling decisions would lead to better decisions. Second, we would be able to accurately get at subjective information from users, such as job urgency and overall value. Using this information in scheduling decisions can lead to increased user satisfaction [5]. Third, this information about job value would give resource providers additional metrics with which to guide purchasing decisions. Rather than base decisions solely on system-centric measurements, administrators would be able to consider metrics such as aggregate job value when making new hardware investments.

1.3 Thesis Overview

In order to elicit honest behavior, we will be leveraging existing work in the fields of Auction Theory and Mechanism Design. As will be explaned at greater length in Chapter 3, we can't tractably deliver all the features that we'd like while adhering to the theoretical constraints of our payment mechanisms. Figure 1.1 depicts the "choose any two" nature of the design space. Because sacrificing

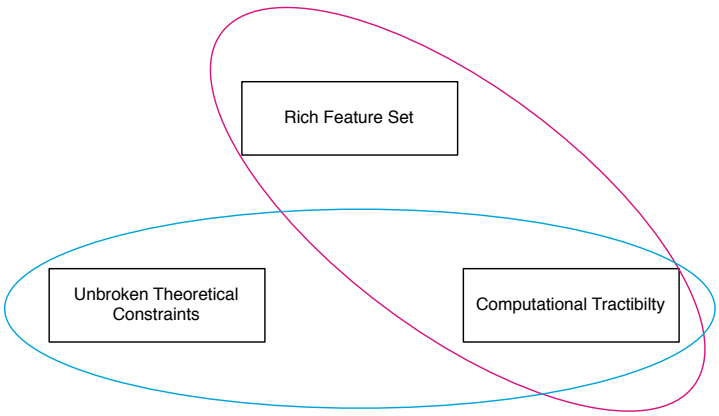

Figure 1.1: Choose any two: Rich Feature Set, Computational Tractibility, Unbroken Theoretical Constraints

tractability does not make sense in a high-performance computing environment, we've chosen to investigate the two circled combinations: solutions that are both tractable and are faithful to theoretical constraints, and solutions that are both tractable and are feature-rich.

These two approaches have been applied to both Batch Queues and to Reservation Systems. The relationship between these two design axes and the structure of this dissertation can be seen in Figure 1.2.

This dissertation is structured as follows. In the next chapter, we will present an overview of related work. In chapter 3, we will present the relevant concepts from game theory and mechanism necessary to understand the work that follows.

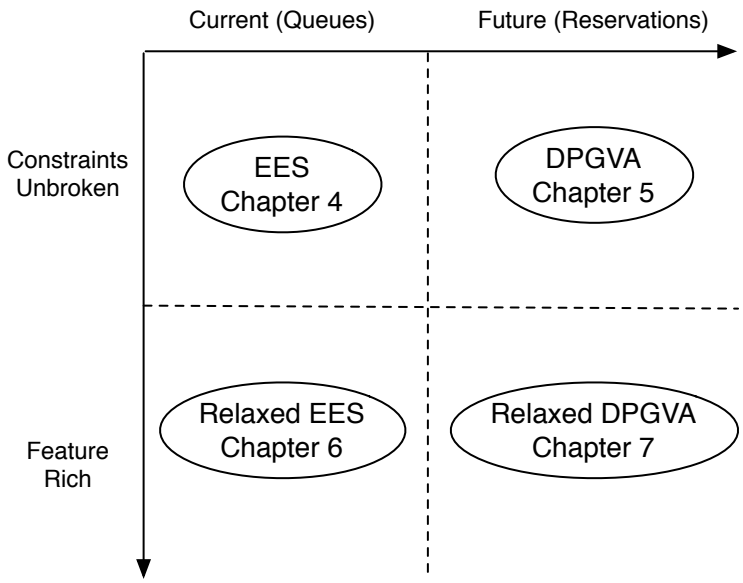

Figure 1.2: A four-paned figure showing the two axes: the batch-reservation axis (present vs. future), and the theoretical-practical axis

The relationship between chapters 4, 5, 6, and 7 is shown in Figure 1.2. In Chapter 4, we investigate batch queue mechanisms that are both tractable and are faithful to theoretical constraints. In Chapter 5, we investigate reservation mechanisms that are both tractable and are faithful to theoretical constraints. In Chapter 6, we investigate batch queue mechanisms that knowingly break theoretical constraints in order to deliver an increased feature set. In Chapter 7, we investigate reservation mechanisms that knowingly break theoretical constraints in order to deliver an increased feature set. Chapter 8 will present a discussion of the results found in chapters 4 through 7. Chapters 9 and 10 will present future research directions and concluding remarks, respectively.

Chapter 2

Related Work

This dissertation explores the viability of applying theoretical results from the field of mechanism design to scheduling systems on computational grids. In this chapter, we will discuss several landmark papers that are relevant to how these two fields intersect. We will first present work that focuses on the theoretical end of this problem, followed by work that focuses more on applied systems. We will conclude the chapter with work that, similar to this dissertation, sits between the two.

2.1 Auctions and Mechanism Design

The work in this dissertation applies results from the study of auctions in field of Mechanism Design in order to elicit honest divulgement of metadata. In this section, we discuss some of the results that this work relies upon, and some related work that could be used in the construction of other grid scheduling systems.

The two auction mechanisms that this work is directly based upon are the *Generalized Vickrey Auction* and the *Expected Externality Mechanism*. These mechanisms will be discussed at greater length in Chapter 3, but should be briefly mentioned in this chapter as well.

The Generalized Vickrey Auction is the generalized form of the sealed-bid second-price auction originally studied by William Vickrey in 1961 [23]. This auction was subsequently generalized in work done by Clarke in 1971 [6] and Groves in 1973 [8]. GVA payments can be applied to auctions of a variety of forms, but the well-known and powerful *combinatorial GVA* allows bids that express *complementarity*, *substitutability*, and *externality*. An auction that allows the expression of *complementarity* allows users to indicate that the value of goods A and B together is greater than the sum of their individual values. *Substitutability* can be thought of as the opposite of *complementarity* in that it is the expression of goods being worth less together than the sum of their individual values. An auction that allows the expression of *Externality* is one that allows a user to indicate that a transaction between other participants affects that user either positively or negatively. The Generalized Vickrey Auction is well-known for being truth-revealing, meaning that it is in a participants best interest to honestly divulge all relevant private information to the mechanism. These features, and others, are discussed at greater length in Chapter 3.

The *Expected Externality Mechanism*, is a truth-revealing auction mechanism developed independently in 1979 by both Kenneth J. Arrow [1] and by Claude dAspremont and Louis-Andre Gerard-Varet [7]. Like the GVA, in the EEM bids are submitted in sealed fashion and can exhibit complementary, substitutive, and external preferences. Unlike the GVA, the EEM provides *balanced budget* payments, but also unlike the GVA, it lacks the desirable feature of *individual rationality*. Additionally, the truth-revealing incentives in the EEM are weaker than in the GVA, as honesty is only *nash-optimal* in the EEM, but is a *dominant strategy* in the GVA. These incentives, and the EEM in general, are discussed at greater length in Chapter 3.

There are two significant theoretical results that limit the design of the schedulers in this dissertation: the incompatability certain desirable auction characteristics, and the intractability of many forms of combinatorial auction. The first limitation was shown by Myerson and Satterwaithe in [12]. This landmark work showed that the three properties of budget balance, efficiency and interim individual rationality cannot be simultaneously achieved in any mechanism. Any two of the properties can exist together, and the mechanisms that this dissertation relies upon exhibit two different pairs of these (the GVA exhibits efficiency and individual rationality, and the EEM exhibits budget balance and efficiency).

The second significant limitation that is a driving force in our work is that combinatorial auctions can be computationally intractable to solve. In [17], Sandholm et al. look at the impact of auction structure on the algorithmic complexity necessary to find optimal solutions, approximate solutions and feasible solutions. As will be discussed in Chapter 3, many of the truth-revealing characteristics of the GVA and EEM require allocation functions that optimally reconcile bids, and schemes that use approximate allocation functions violate the requirements of these mechanisms. In [17], the difficulty of optimally solving many different combinatorial auction forms is discussed, and while there exist approximation algorithms for a few of these, optimally solving them is NP-complete or worse in all cases. In some cases, finding a solution that is even *feasible* is NP-complete.

Approximation algorithms are the obvious response to this intractability, but as we've mentioned simply approximating the GVA can produce a mechanism that is not strategy-proof. One solution to this is found in [11], where Lehmann et al. present a strategyproof auction mechanism that is both approximately optimal and computationally tractable. In order to accomplish this, the authors restrict the preferences of the participants to those that are *single-minded*, which is where a participant is satisfied if he acquires a specified minimum bundle within a specified maximum price. This mechanism has been successfully used to schedule time on sensor-network testbeds, as discussed below in [3]. The restriction of the

mechanism to participants whose preferences are *single-minded* prohibits its use in our target environment, however, as there are many allocations between which users are genuinely indifferent.

In the scheduling systems in this dissertation, we use a simple representation of user preferences, that allows a participant to specify his satisfaction with a variety of outcomes with a few parameters. While this solution does a decent job of representing user preferences, future researchers may want to expand these representations. This can lead to an information transmission problem, as the GVA payment scheme requires the user to value all possible outcomes. In [15], Parkes presents an auction that reaches the GVA outcome without a complete, upfront valuation of all possible outcomes. The authors refer to this auction as an iterative version of the GVA. The key difference between the iterative GVA and its non-iterative version is that in a given round, agents are only required to bid on a single bundle of goods. As a result the communication requirements of this scheme are far lower than the traditional GVA, where each participant must submit a bid representing a valuation of all possible allocations of the auction.

As will be discussed at greater length below, the composition of multiple locally truth-revealing mechanisms does not always produce a single globally-truth-revealing mechanism. Ng et al. attempt to address this in a dynamic environment with the Virtual Worlds system presented in [14]. In a mechnanism whose partici-

pants are coming and going dynamically, each participant will face the decision of when to enter, and these decisions influence the amount of gain that user will see from the auction. In Virtual Worlds, the authors attempt to address this potential for strategization by keeping track of the gains that a user would have seen if he had delayed his entry into the mechanism and compensating him if he would have gained by delaying. As a result, a user's dominant strategy is to enter the mechanism as early as possible. The mechanism builds on the work of Lehmann et al. in [11], and is subject to the same restrictions regarding participant preferences and *single-mindedness*.

2.2 Auction-Based Scheduling Systems

The notion of applying auctions and market mechanisms to the scheduling of distributed computer resources has existed for some time. This section highlights some of the landmark work done in this direction. The work discussed in this section differs from section 2.3 in that the solutions don't focus primarily on truth-revelation and participant incentives.

The earliest of these ideas dates to 1968 [22], where Sutherland et al. allow users to indicate the relative worth of resources via a unique mechanism for allocating time on a shared computer. Users are allowed to bid on contiguous blocks

of time on the computer for the next day. Users compete for time by outbidding one another, but users can not bid in such a manner that they partition an existing block in two. Users pay for computation at the rate of their bid, but the currency used is only consumed for the remainder of the day. Each morning users' accounts are reset to the same quantity. This system is not intended to elicit honest information from users, and the designers make no claims to that effect.

In another early auction-based system, in [10] Kurose et al. designed and built a distributed, market-based system to manage file storage placement in a distributed system. The work differs from other market-based approaches in that uses a *resource-directed* allocation mechanism rather than a more traditional *price-directed* mechanism. In a *price-directed* scheme a central authority would adjust prices until the quantities supplied and demanded were equal for each good. In a *resource-directed* scheme, by contrast, the system starts with an arbitrary allocation of resources and a central authority collects from each agent the marginal value of each resource and reallocates resources from those with a lower-than-average marginal value to those with a higher-than-average marginal value. For environments where decentralization is desired, the scheme optionally uses a "pairwise interaction algorithm", where instead of exchange being coordinated by a central auctioneer, the participants pair up and make locally optimizing transac-

tions with each other. The scheme does not attempt to prevent user manipulation, with no assurance that participants will honestly reveal their true marginal values.

In the Spawn system [25], each hardware resource that is being offered in the system conducts a separate Vickrey auction where applications bid to use the system. Because each auction is independent and is executed on separate hardware, the system is very decentralized and fault tolerant. The consequence of this decentralization is the absence of combinatorial bidding, which prevents the system from being truth-revealing. Rather than bid their true value for the resource in question, participants must bid strategically if they have combinatorial preferences. For example, if one only valued resources A and B as a pair and for $\$X$, should one bid $\$X/2$ for each? If one of these resources was likely to be in high demand and the other in very low demand, should different quantities be bid? The optimal strategy in this environment isn't clear, but for many sets of preferences, it isn't honesty. This example highlights how the composition of multiple locally truth-revealing mechanisms are not necessarily globally truth-revealing.

Another early auction-based resource management framework is the Popcorn system [16]. In Popcorn, Nisan et al. experiment with three different auction mechanisms to allocate grid resources. The first is very similar to the technique employed in [25], where a second-price, sealed-bid auction is conducted for each resource. The second is a double auction where an upper and lower limit is spec-

ified, along with a rate to move between the two. The auctioneer moves each bidder from one limit to the other at the specified rate and matches buyers and sellers with compatible prices. The third is a double auction that is cleared periodically, with the auctioneer computing a market clearing price from the collection of existing bids and clearing the market at that price. Similar to the Spawn system, the scheme doesn't address the strategic issues that emerge when participant satisfaction is combinatorially conditional.

Stoica et al. present the application of a nontraditional microeconomic mechanism in [21]. At each time step, a sealed-bid first-price auction is held to determine who will use each resource, unless the resource is currently in use. If the resource is in use, than the process that occupies it is allowed to continue running. The rate that this process pays, however, is computed at the beginning of process execution and is based on a statistical prediction of price fluctuations. This approach is very computationally lightweight and avoids the challenges faced by systems that allow combinatorial bidding. The scheme allows participants to request resources on many machines at once, so avoids the strategic issues that Spawn and Popcorn face, but its design is system-centric and makes no claims about the optimal behavior of participants.

The use of the commodity market model for grid scheduling is explored by Wolski et al. in [26]. Through simulation, the authors compare the performance

of scheduling using sealed-bid, second-price auctions with that of allocation using a commodities market with two different price adjustment mechanisms. Both price-adjustment mechanisms construct an excess demand curve, representing the difference between the quantity supplied and demanded for each good at all price points. This curve is navigated by a technique known as "Smale's method" [19] to find a set of prices where excess demand is zero. The scheme differs from traditional Tatonnement price adjustment schemes in the respect that the market is transacted outside of equilibrium. This significantly reduces the computational cost of the scheme, however, as the system does not need to wait for the process to converge to the price equilibrium before exchanging goods.

In [5], Chun et al. demonstrate how grid schedulers that consider job value when making scheduling decisions yeild significantly higher performance when evaluated using user-centric metrics. In this work the authors use simulation to determine magnitude of this hidden inefficiency. They compare the aggregate utility of the participants between batch queue environments that alternately do and do not consider reported user value. Because the authors are doing their work in simulation, they are able to use aggregate global utility as their metric for comparison (in a non-simulated environment they would need to address the problem of accurately getting this information). The authors find that in the presence of variation of job value, the batch queueing systems that consider this

information achieve significantly greater global utility than those that don't. To address this, Chun et al. have built and tested in [4] a resource sharing system that allows the expression of value by conducting an uncommon "proportional" auction. The auction is such that if many users bid on the same resource, each user gets a fraction of the resource equal to the fraction that their bid represented of the bids submitted and pays the amount that they bid. This mechanism has some serious drawbacks, however. In particular, far from being truth-revealing, this auction mechanism necessitates the use of strategy, as it is impossible to even value a resource without predicting how popular the resource will be with other users.

2.3 Truth-Revealing Schedulers

Less work has been done directly in the field of truth-revealing grid schedulers than in the broader areas above. In this section we will discuss grid scheduling systems that prioritize truth-revelation.

The Bellagio system, presented by Auyoung et al. in [2], is a scheduler for computational grids that uses combinatorial bids to allocate resources. These bids can be very complex, with users valuing many different allocative outcomes, including many different resource bundles and start times. There is complete

combinatorial flexibility in valuing these outcomes. The Bellagio system uses a variant of Vickrey pricing in order to elicit honest bids from participants. The Bellagio system does an excellent job of eliciting honest, complex preferences from users, but sacrifices computational tractibility to do so. Solving the combinatorial auction produced by Bellagio bids is an NP-Complete problem. The authors state that approximation algorithms are intended in the future to address this, but don't address the impact that these approximations will have on the incentives faced by participants.

In similar fashion, the MACE system is described by Schnizler et al. in [18]. Standing for Multi-Attribute Combinatorial Exchange, MACE is a grid scheduling system that gives users a very rich biding language in which to express their preferences. Complex bundles can be valued, indicating both complementary and substitutive relationships. The authors experiment with using both a Vickrey pricing scheme and a "k-pricing" scheme. The k-pricing scheme is a budget-balanced pricing scheme that allocates the consumer-producer surplus between the participants with one getting k and the other getting $(1-k)$. In both the Vickrey pricing and the k-pricing schemes, solving this system is requires solving an NP-complete problem. In order to address this, the authors experiment with using the CPLEX solver to accelerate the average case. The worst case performance

can be poor, however, and this is handled by terminating the solver after four minutes.

In order to avoid such NP-complete problems, a more streamlined solution is presented in [20] by Stoesser at al. Rather than allow the valuation of arbitrarily complex combinations of goods, GreedEx allows only two goods: computation and memory. Users submit 5-tuple bids that indicate the value of the work to be performed, the amount of cpu and memory needed, and the expected running time. In this more constrained model, the authors are able to create an approximation algorithm to solve the allocation problem. Additionally, the payment scheme used provides approximate truth-revelation. With respect to the Myerson-Satterwaithe impossibility result discussed in Chapter 3, the GreedEx system explores a combination of permissible attributes that this dissertation does not explore: budget balance and individual rationality.

Chapter 3

Grid Computing, Game Theory and Mechanism Design

The work that we have done in truth-revealing grid schedulers straddles the divisions of some traditional disciplines, and as such we would not necessarily expect the reader to be familiar with all the fundamentals that this work builds upon. In this chapter, we will present sufficient supporting material so that the reader will better understand the work in the following chapters.

This chapter is structured as follows. We will first describe to the reader what Batch Queues and Reservation Systems look like in practice today. Next, we will provide an overview of the necessary concepts from Game Theory. We will conclude the chapter with an overview of Mechanism Design and a description of the mechanisms we will be experimenting with in later chapters.

3.1 Current Practice

3.1.1 Batch Queues

Batch Queues have been used to allocate resources since the mid 1980's. In their simplest form, batch queues consist of virtual lines that users wait in for access to computational resources. Users submit a job to the system along with accompanying metadata, such as the expected running time and the number of nodes that the job needs to run. Users are given no guarantees regarding how soon their jobs will run: they may see immediate exection, or they may wait weeks. These jobs are handled in a pseudo-FIFO order, as both backfilling and user priority heuristics affect the schedule.

Batch queues have many strengths: they are simple to use, they produce very egalitarian resource allocations, and they can yield a high degree of resource utilization. They are far from perfect, however, as they provide poor incentives for users to moderate usage during periods of high demand. For example, consider the incentives faced by users when some members of a research community are trying to meet a publication submission deadline. There would be a surge in job submissions and job turnaround times would increase, but no other factor would discourage non-urgent uses of the resource.

3.1.2 Reservaton Systems

Reservations systems are a more recent model in the high-performance computing world. Rather than wait in lines for an uncertain execution time, reservation systems allow a user to get a guaranteed time slot at a future date. At this time, some reservation systems even require direct administrator contact to create reservations.

Resource providers have historically been averse to reservation systems, as these systems tend to be more prone to gaps between time slots, and can lead to a reduction in total system utilization. Users, on the other hand, tend to be the proponents of such systems, as they value the increased certainty regarding execution time. Like batch queues, reservation systems don't have strong incentives in place to discourage consumption during peak times.

3.1.3 Preferences in Grid Systems

The users in grid systems tend to be interested in their own performance, and aren't too concerned about the performance seen by other users. Computational grids are commonly composed of federated resources from multiple institutions, and the users that consume these grid resources are themselves from different institutions. In this situation, it is unrealistic to expect users to act in a purely altruistic fashion. If we were to simply ask each user how important his job was,

without any consequences for overstating the value, it would be in his interest to reply that his jobs were maximally important. Using such responses to make scheduling decisions would, of course, not produce efficient results. In order to be effective, a scheme to manage these resources must assume that users will act in a manner that advances their own best-interests.

Resource providers, on the other hand, tend to be more altruistic in this environment. Providers of grid resources are government-funded organizations, tasked with purchasing resources to advance scientific research. These organizations use traditional system-centric performance metrics (such as utilization) to measure how well they advance the goals of users, and use these metrics to guide purchasing decisions. As a result, we don't really need to worry about malicious resource providers in our target environment.

3.1.4 Preferences Specific to Batch Queues

In a queued environment, consumption incurs a strongly negative externality on other users. Any time that the cluster spends executing the job at the head of the queue is increased turnaround time for all the jobs waiting in line. Because most batch queues do not execute jobs in a purely-FIFO order, this increase isn't necessarily equal to the execution time of the job at the head of the queue, but it is generally greater than zero.

To clarify what we mean when we say these preferences are *external*, A classic example is the polluting factory. The factory produces goods and sells them to consumers, but the pollution generated by the factory affects more than just those who buy the product: everyone living nearby is affected by the results of transactions in which they do not directly participate. An economic model that does not consider these external effects may not capture the true incentives of the participants or accurately represent the true valuations assigned to resources.

These externalities restrict the space of mechanisms that we have at our disposal. Some otherwise-desirable results in Mechanism Design (such the tractable, truth revealing mechanism developed by Daniel Lehmann in [11]) are out of our reach in queued systems due to these externalities.

3.1.5 Preferences Specific to Reservation Systems

User preferences in reservations systems lack the external element that is present in queued systems. The use or non-use of a resource at time t_i has no effect on the happiness of a different consumer of the resource at other times. The preferences in reservations systems do exhibit more complex complementary and substitutive properties, however. If we discretize the schedule of each resource, adjacent time units tend to be desired by the same user (a complementary rela-

tionship), and many different sequences of time blocks can satisfy the same job deadline (a substitutive relationship).

Like the effect that externalities have on our selection of mechanisms in batch queues, these complementary and substitutive relationships between time slots restrict our choice of mechanism. Just like above, the tractable, truth revealing mechanism developed by Daniel Lehmann in [11] is again out of our reach, although this time it is due to the substitutive nature of the preferences (multiple, equally-valued bundles violates the assumption of single-minded preferences).

3.2 Game Theory Concepts

Game Theory is a branch of mathematics that studies how self-interested participants behave in a system of rules. The discipline flows naturally from the idea that if one can quantitatively understand an actor's interests, and one can model that actor's interactions with other self-interested actors, then one can use mathematics to understand what behaviors advance those interests. The field was initially developed by John von Neumann and Oskar Morgenstern in the 1940's with the publication of [24].

Game theory predicts real-world behavoir only to the extent that real-world actors behave rationally. A critic of game theory (and work derived from it) could

claim that the assumption of perfect rationality on the part of human participants is invalid, as humans no not always act rationally. In the context of our work, however, individual participants may not understand all the theoretical reasons that honest behavior is optimal, but if they are assured by experts that honesty is the optimal strategy (and their personal experiences bear this out) it is likely that they will behave in this manner. For more information on these topics, see [9].

There are multiple ways to represent games mathematically, but for the purposes of this dissertation we will use the following.

Some outcome, $x \in X$, is produced by a game, κ. This game κ is a function that maps a set of player actions (also called strategies), s, to outcomes.

$$x = \kappa(s)$$

The set of all participants is I. The set of strategies s is composed of the strategy played by participant $i \in I$, called s_i, and the strategies played by everyone else, called s_{-i}. So this can also be written as:

$$x = \kappa(s_i, s_{-i})$$

For each participant i, there exists a function, U, that describes their satisfaction or *utility* seen from some outcome x, $U_i(x)$. U maps from the set of all

possible outcomes, X, to a real number $U: X \rightarrow \mathbb{R}$. We can rewrite this function with a parameter t_i, representing participant i's preferences, also called his *type*, as $U(t_i, x)$. So we can write a participant's satisfaction with the outcome of a game as:

$$U(t_i, \kappa(s_i, s_{-i}))$$

A participant i does not know how the other participants will act (s_{-i}), but he may have some impression of the likelihood of others' actions. In this case, in order to maximize his satisfaction, he will choose an action, s_i, that maximizes his expected satisfaction:

$$E_{s_{-i}}\left[U(t_i, \kappa(s_i, s_{-i}))\right]$$

For some games, there exists a strategy s_i^* that maximizes i's satisfaction, regardless of the strategies employed by other players, s_{-i}. In this case, we refer to that s_i^* as a *dominant strategy*. In other words, s_i^* is a dominant strategy if and only if, for all other strategies, $\bar{s_i}$, playable by i:

$$U(t_i, \kappa(s_i^*, s_{-i})) \geq U(t_i, \kappa(\bar{s_i}, s_{-i}))$$

for all s_{-i}.

The existence of a dominant strategy is a very strong predictor of participant behavior. If a participant knew of a strategy that would maximize his satisfaction, regardless of anything else, he would likely play this strategy. Dominant strategies, however, don't always exist.

Another, weaker type of predictable play is called a Nash-optimal strategy. If there exists a set of strategies, S^*, such that for each participant i, $s_i^* \in S^*$ is a best response to the rest of the strategies in s_{-i}^*, then that set is referred to as a Nash equilibrium. Symbolically, for all $\bar{s_i}$:

$$U(t_i, \kappa(s_i^*, s_{-i}^*)) \geq U(t_i, \kappa(\bar{s_i}, s_{-i}^*))$$

Nash-optimal play is a weaker prediction of behavior, since it is predicated on other participants also acting according to the equilibrium. This can be particularly problematic when more than one Nash-equilibrium exists. From the perspective of Mechanism Design, however, it is still useful as we will be using mechanisms where honesty is a Nash-equilibrium, and as a result multiple equilibria are not a problem.

3.3 Mechanism Design Concepts

Mechanism Design is a field of study that seeks to design a system of rules whereby self interested participants are incentivized to act in a manner that delivers a desired outcome. These interaction mechanisms can take many forms, with voting systems and auction systems being the most commonly studied. The desired outcomes that these mechanisms are designed to produce can vary, but one commonly studied outcome is the advancement of the aggregate welfare of all the participants. There are multiple ways to represent a mechanism symbolically, but for the purposes of this dissertation we will use the following, loosely following the notation in [9].

We will represent an auction mechanism with two functions: an allocation function κ that computes a resource allocation (called $x \in X$, where X is the set of all feasible allocations) and a payment function μ that computes the change in currency that each participant will see. Both functions take the set of bids submitted by each participant as argument. Each participant will pay into the system according to the payment function:

$$\mu_i(s_i, s_{-i})$$

and the resources will be allocated by the κ function:

$$\kappa(s_i, s_{-i})$$

As a result, the satisfaction seen by participant i from submitting s_i, when the other participants have submitted s_{-i} is:

$$U(t_i, \kappa(s_i, s_{-i})) - \mu_i(s_i, s_{-i})$$

Participant i, of course, doesn't know how the other participants will be bidding. If he has some sense at the likelihood of their bids, however, he will act to maximize:

$$E_{s_{-i}}\left[U(t_i, \kappa(s_i, s_{-i})) - \mu_i(s_i, s_{-i})\right]$$

There are many different desirable features that a mechanism can have. Four characteristics that are frequently discussed in designing auctions are truth revelation, individual rationality, budget balance and efficiency.

3.3.1 Truth Revelation

An auction is considered to be *truth-revelealing* if it is the optimal strategy of the participants to truthfully reveal their privately held information to the

mechanism. For example, in an auction for a single item, each participant has private information in the form of the amount he values the item. In a truth-revealing single-item auction, it would be in the best interest of each participant to submit a bid corresponding to this private valuation.

Symbolically, this means that the bid s_i^* that yields the highest satisfaction:

$$U(t_i, \kappa(s_i^*, s_{-i})) - \mu_i(s_i^*, s_{-i}) \geq U(t_i, \kappa(\bar{s}_i, s_{-i})) - \mu_i(\bar{s}_i, s_{-i})$$

for any $\bar{s}_i$, and s_i^* is the bid corresponding to the true value t_i.

3.3.2 Individual Rationality

An auction is said to be *individually rational*, if each participant is guaranteed that participating in the auction will leave them in an equal or better position than they would be if they had not participated at all. There are three types of individual rationality: *ex ante*, *interim*, and *ex post*. A mechanism is said to be *ex ante individually rational* if it is in someone's best interest to participate before they know their own preferences. A mechanism is said to be *interim individually rational* if it is in someone's best interest to participate after they know their own type, but before they know the types of other participants. A mechanism is said to be *ex post individually rational* if it is in someone's best interest to participate

after they know their own type, and the types of all other participants (i.e. after the mechanism has completed). As should be obvious, these three are listed in order of strength, with *ex ante* being the weakest form of individual rationality and *ex post* being the strongest.

Symbolically, we would say that ex post individual rationality is, for all s_i, s_{-i}:

$$U(t_i, \kappa(s_i, s_{-i})) - \mu_i(s_i, s_{-i}) \geq 0$$

3.3.3 Budget Balance

An auction is called *budget balanced* if it requires no inflows or outflows of currency to operate. Some mechanisms can only operate properly if there is an external agent paying into the system. Mechanisms that don't require such an entity are considered *budget balanced*.

So, for all s_i, s_{-i}:

$$\sum_{i \in I} \mu_i(s_i, s_{-i}) = 0$$

3.3.4 Efficiency

An auction is considered *efficient* if the allocation of goods that it produces allocates items to the individuals who value them most. If there exists no alloca-

tion for which the aggregate utility of the participants is greater, then the auction is considered *efficient*.

An efficient allocation, x^*, has the property

$$\sum_{i \in I} U(s_i, x^*) \geq \sum_{i \in I} U(s_i, x)$$

for all $x \in X$.

3.3.5 What we can not do

There are two limitations in mechanism design that are relevant to this dissertation. The first result is that we can't have all of the above in one mechanism simultaneously. As Myerson and Satterwaithe showed in [12], the three properties of budget balance, efficiency and interim individual rationality cannot be simultaneously achieved in any mechanism. Any two of the properties can exist together, and as will be described below, this dissertation experiments with two of the three allowable combinations.

A second relevant limitation is that the mathematics that provide us many of the theoretical guarantees in these mechanisms is very sensitive to the accuracy of the underlying allocation function used when reconciling competing bids. As you will see below, both the Expected Externality Mechanism and the Generalized

Vickrey Auction assume that the allocation function (referred to as κ) that maps a set of bids to an allocative outcome is accurate and optimal. Additionally, many of the allocation problems to which we would like to apply these mechanisms are intractably hard to solve optimally. As a result, we cannot simply replace an optimal allocation algorithm with an approximately-optimal algorithm and expect these guarantees to be preserved.

3.4 Mechanisms we Employ

3.4.1 The Expected Externality Mechanism

The *Expected Externality Mechanism*, is a truth-revealing auction mechanism developed independently in 1979 by both Kenneth J. Arrow [1] and by Claude dAspremont and Louis-Andre Gerard-Varet [7]. In this mechanism, bids are submitted in sealed fashion, and an allocation function optimally reconciles these bids to produce an efficient allocation. Its payments are budget balanced, and its allocations are efficient, so from the perspective of the Satterwaithe impossibility result, the Expected Externality mechanism sacrifices individual rationality.

Lacking individual rationality is a weakness, but for our target environment of batch queues, not a fatal one. It would be difficult for users to determine exactly when it would harm them to enter the system without knowing in advance the

rest of the users submitting jobs in the system. Additionally, even if a user was able to make this difficult determination, the consequence would simply be one fewer job submitted to the system.

Another concession that must be made for this mechanism is that it requires knowledge of the statistical distribution of user preferences. This is a minor concession in our target environment, as a centralized computing provider is perfectly placed to record and process the submitted user preferences in order to determine this information.

The Expected Externality Mechanism does properly incentivize truth-revelation, although truth-revelation is a Nash-optimal strategy, rather than a dominant strategy. As discussed above, this means that it is in a participants best interests to be honest, only if all the other participants are being honest. Creating a system where this self-reinforcing honesty exists is discussed at greater length in Chapter 8.

Symbolically, a participant in the Expected Externality Mechanism pays according to the function $\mu_i(s_i, s_{-i})$:

$$\left(\frac{1}{|I|-1}\right)\sum_{j\neq i} E_{r_{-j}}\left[\sum_{l\neq j} U(r_l, \kappa^*(s_j, r_{-j}))\right] - E_{r_{-i}}\left[\sum_{j\neq i} U(r_j, \kappa^*(s_i, r_{-i}))\right] \quad (3.1)$$

Where $|I|$ is the number of participants, $E_{r_{-i}}$ is the expected value over the stated preferences of everyone other than i, κ^* is a function that produces an efficient allocation from the set of bids.

3.4.2 The Generalized Vickrey Auction

The Generalized Vickrey Auction is the generalized form of the sealed-bid second-price auction originally studied by William Vickrey in 1961 [23]. Work was done by Clarke in 1971 [6] and Groves in 1973 [8] in generalizing this mechanism. This auction is perhaps the most well-known truth-revealing auction around today. In the GVA, truth-revelation is a dominant strategy, meaning that it is in a participant's best interests to be honest, regardless of the truthfulness of other participants. This is a stronger, and preferable, solution concept to that of the Expected Externality Mechanism.

Another strength of the GVA is that it is ex post individually rational, so no participant leaves the mechanism worse-off than when they entered it. Additionally, the function that reconciles competing bids produces efficient allocations. As a result of this individual rationality and efficiency, and the existence of the Myerson-Satterwaithe impossibility theorem, the GVA cannot be budget balanced. While the lack of budget balance in our target environment is certainly not a strength, our target environment uses virtual, cluster-issued currency, and

as such it is not as daunting a property as it is in systems using real-world currency. This is discussed at greater length in Chapter 8. Also, unlike the Expected Externality Mechanism, the GVA does not require knowledge of the participants preferences in order to operate.

The GVA can be thought of as charging each participant the difference between the aggregate social welfare of others with and without his presence. Symbolically, we represent this as:

$$\mu_i(s_i, s_{-i}) = \sum_{j \neq i} U(s_j, \kappa^*(0, s_{-i})) - \sum_{j \neq i} U(s_j, \kappa^*(s_i, s_{-i}))$$

3.5 Incentives and Tractibility

As discussed above, the guarantees that these mechanisms provide about user behavior only hold as long as the allocation function optimally reconciles bids. Unfortunately, many of the features we would like to provide in our scheduling system would require the use of an allocation function that cannot be tractably implemented. A common solution to computational intractability is the use of approximation algorithms, but their lack of optimality breaks the optimality constraint, and we lose the guarantees of truth revelation.

If we want to provide truth-revelation in a tractable manner, there is a gap between the feature set that we can deliver and the feature set we'd *like* to deliver. That gap is the focus of this dissertation. Chapters 4 and 5 approach this gap from the side that sacrifices features in order to be faithful to the theoretical constraints of the mechanism. Chapters 6 and 7 approach this gap from the other side, knowingly breaking constraints in order to deliver an expanded feature-set, and intending to examine to what degree truth-revelation has been broken.

Chapter 4

Eliciting Honesty in Batch Queues

Batch Queues have been used to allocate resources since the mid 1980's. In their simplest form, batch queues consist of virtual lines that users wait in for access to computational resources. As currently implemented, batch queues provide no guarantees as to when a user's job will be run. Jobs are handled in a pseudo-FIFO order, as backfilling and priority heuristics affect the schedule.

Batch queues have many strengths: they are simple to use, they produce very egalitarian resource allocations, and they can yield a high degree of resource utilization. They are far from perfect, however, as they provide poor incentives for users to moderate usage during periods of high demand. For example, consider the incentives faced by users when some members of a research community are trying to meet a publication submission deadline. There would be a surge in job submissions and job turnaround times would increase, but no other factor would

discourage non-urgent uses of the resource. A user whose job was not urgent would certainly see his results sooner if he submitted now, rather than waiting until a period of lower demand. As a result of this urgency-agnosticism, high batch queue utilization numbers can belie hidden inefficiency as resources are being allocated to the execution of the job at the front of the queue, regardless of the value of the work to be performed.

The work presented in this chapter is intended to address this issue. In order to distinguish between urgent, important work, and jobs that are less time sensitive or less important, we need to get at the quantification of these notions that exists only inside the heads of users. As discussed in the previous chapter, we are attempting to use the field of Mechanism design to do this, specifically the Expected Externality Mechanism.

The Expected Externality Mechanism was developed independently in 1979 by both Kenneth J. Arrow [9] and by Claude d'Aspremont and Louis-Andre Gerard-Varet [10] (for more information, see 3.4.1). Due to this theoretical basis, we have named our scheduler the Expected Externality Scheduler (or EES).

4.1 Our Batch Queue Scheduler

4.1.1 What Users See

In our scheduler, the Expected Externality Scheduler (or *EES*), users submit three parameters along with every job: called v, d, and r and explained below. The scheduler considers the jobs in First-in-First-out (*FIFO*) fashion, and based on the submitted parameters, decides to either run the job or to discard it. In general, a job accrues currency while waiting in the queue and pays out currency if it runs when it reaches the front of the queue. The net effect is that jobs which are eventually discarded by the mechanism gain currency overall, and those that are run by the mechanism lose currency.

The first two parameters are currency-based and indicate job priority: the value of the work to be performed. v, and the tolerance of the user towards delay in total turnaround time, d. Both v and d are specified in terms of a cluster-specific currency that has been allocated by the administrators of the resource. In addition to this value information, the user indicates a maximal running time of the job, r.

The value parameter, v, is simply the amount of currency that the user would be willing to pay in order to have the job executed. High-value jobs have a low

likelihood of being discarded by the mechanism when they reach the front of the queue.

The delay intolerance parameter, d, represents how eager a user is to have other preceding jobs discarded by the mechanism. This parameter can be thought of as being in units of currency-per-second, where the amount of dissatisfaction that is experienced by that user by waiting x seconds is $x * d$. The presence of many jobs in the queue with high delay intolerance parameters would mean a high likelihood of jobs being discarded by the mechanism.

4.1.2 Schedule Determination

When each job reaches the front of the queue, it is considered for running. This decision is driven by the parameters associated with that job and the parameters associated with the other jobs waiting in the queue.

Intuititvely, this decision function compares the value of the job in question with the collective inconvenience borne by the other jobs waiting in the queue. Symbolically, the decision to run job i can be represented as such:

$$a = v_i \tag{4.1}$$

$$b = \sum_{j \neq i} d_j * r_i \tag{4.2}$$

$$\kappa^*(a,b) = \begin{cases} discards, & a < b \\ runs, & a \geq b \end{cases} \tag{4.3}$$

If $a \geq b$, job i is run by the scheduler. Otherwise, job i is discarded and the next job in the queue is considered.

It is important that this allocation decision can be made quickly, as the Expected Externality payment mechanism (discussed in the following subsection) requires the recomputation of this function many times. The tractibility of this decision is in fact what has necessitated the simplicity of the job metadata. We would have preferred to be able to ask for any number of nodes when submitting jobs, but optimally computing the effect on turnaround time of a given job in that formulation becomes very hard. By assuming the number of nodes needed by each job is equal to the number of nodes available on the resource, we can be certain that the effect on turnaround time is equal to the length of the job in question. In Chapter 6, we consider the relaxation of this characteristic.

4.1.3 Payment Computation

As discussed in Chapter 3, we are using the Expected Externality Mechanism in this scheduler to provide incentives to users for being honest and accurate when

submitting the parameters associated with their job. The theoretical properties and history of this mechanism are discussed above in Section 3.4.1 of that chapter.

The Expected Externality Mechanism is budget balanced. In our scheduler, running jobs pay currency to jobs that are waiting in the queue. The net effect of this is that a job that waits in the queue and does not end up eventually running leaves with more currency than it entered with. A job that eventually runs leaves with less currency than it entered with.

Symbolically, the payment made by user i is (following notation from [9]):

$$\left(\frac{1}{I-1}\right)\sum_{j\neq i} E_{s_{-j}}\left[\sum_{l\neq j} U(s_l, \kappa^*(t_j, s_{-j}))\right] - E_{s_{-i}}\left[\sum_{j\neq i} U(s_j, \kappa^*(t_i, s_{-i}))\right] \quad (4.4)$$

where for a job l at the front of the queue,

$$U(s_l, \kappa^*(t_j, s_{-j})) = \begin{cases} v_l, & \kappa^*(t_j, s_{-j}) = runs \\ 0, & \kappa^*(t_j, s_{-j}) = discards \end{cases} \quad (4.5)$$

and for any job l not at the front of the queue,

$$U(s_l, \kappa^*(t_j, s_{-j})) = \begin{cases} -d_l * r_i, & \kappa^*(t_j, s_{-j}) = runs \\ 0, & \kappa^*(t_j, s_{-j}) = discard \end{cases} \tag{4.6}$$

In this expression, κ^* is the decision function described in Formula 4.3. t_m is the parameter stated by job m (in the case of the frontmost job, this is the v, for all other jobs this is d). In this expression, $E_{s_{-j}}(x)$ refers to taking the expected value of x over the possible parameter announcements of all jobs other than j. In this sense, t_m and s_m are variables of similar type, the difference being that t_m is the actual announced parameter of user m, whereas s_m is the parameter corresponding to job m that is drawn during the expected value computation. I is the number of jobs in the queue. The function U quantifies the amount that a user is affected by the allocation decision κ^* in terms of currency.

In our implementation, we use a Monte-Carlo sampling method for numerically evaluating the expected value computation. To do this, we repeatedly draw samples from the parameter distribution associated with each user and use those parameters to evaluate the expression inside the expected value. We then take the arithmetic mean of the result produced by each draw as the expected value. Because the expected value computation appears in both terms of the payment function, and because there are no nested expected value expressions, we would

expect the running time of the algorithm to scale $O(n)$ linearly as we increase the number of samples we draw when evaluating these expressions. In Subsection 4.2.2, we show results that confirm this expectation.

We expect the computation of the payment function to scale $O(n^3)$ cubically as we increase the number of jobs in the queue. This is because the allocation decision κ^* (Formula 4.3) scales linearly with the number of jobs, and its computation is inside two nested loops whose size also scale linearly with the number of jobs. In Subsection 4.2.2, we show results that confirm this expectation.

4.2 Results

In order to verify that our scheduler exhibited the characteristics predicted by theory, we developed a simulator to test it. In simulation, we have the ability to repeatedly run identical scenarios and compare the payoff seen by a user over a range of different declarations. This allows us to experimentally determine if honesty is really the optimal strategy when interacting with these payment mechanisms.

The following experiments were conducted on a quad-processor Intel Xeon running at 3.2 ghz with 1GB of RAM. The operating system was Debian Linux version 1:3.3.5-13. The simulator was written in the Java programming language

and was compiled and run with Sun Microsystem's java distribution version 1.4.2-03.

4.2.1 Incentives

In Chapter 3, we described how the Expected Externality Mechanism provides theoretical guarantees that honest preference revelation is a Nash equilibrium, meaning that it is an optimal strategy if all other participants are being honest. In order to demonstrate empirically that our mechanism has this property, we created simulated queues and explored the expected payoff for many different preference declarations. It should be noted here that we cannot experimentally *prove* that these mechanisms always incentivize honesty; that can only be done theoretically. We *can*, however, repeatedly examine whether these preferences hold in practice. In each case we have tested, our results show that expected payoff was indeed maximized when users honestly declared their preferences.

Figure 4.1 shows an example of one such run. We created a participant at the front of the queue whose preferences were defined by the parameter 129.8 (meaning the value of the job that was submitted). We then found the expected payoff that the participant would see from randomly selected groups of other participants for each of many declared types. As depicted in the graph, all other non-honest declarations lead to either a comparable or lesser expected payoff for

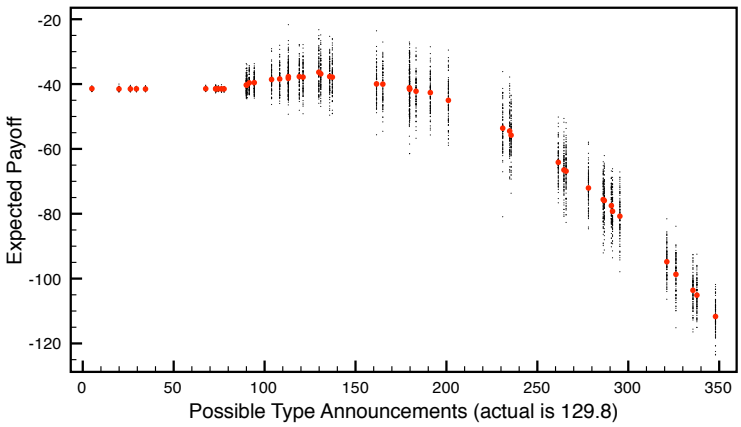

Figure 4.1: Expected payoff for a queued participant.

the participant in question. Each small dot in the graph indicates one expected value computation, the large red dot indicates the average of the black dots for each declaration.

Figure 4.2 is a similar graph for a participant further back in the queue. The participant's actual quantification of how much they dislike waiting an additional hour is 48.9, but we explore the payoff seen when declaring many different values. Similar to Figure 4.1, no other declaration produces a higher expected payoff.

As a result, both of these participants see no value from being dishonest, and in this example honesty is a Nash equilibrium.

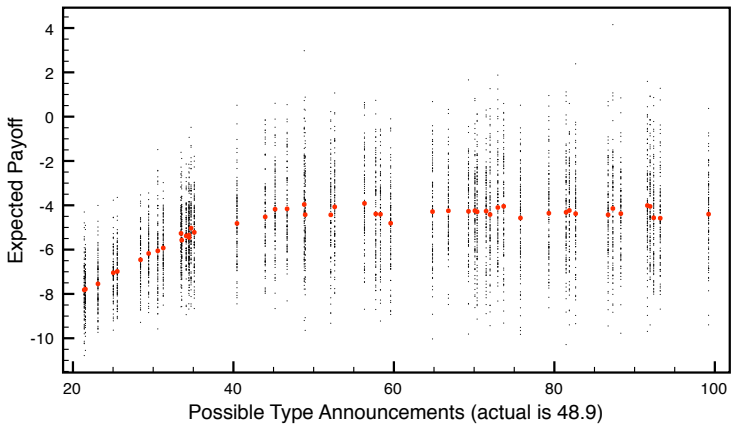

Figure 4.2: Expected payoff for a user waiting deeper in the queue.

4.2.2 Performance

As mentioned above, we expect to see an $O(n^3)$ cubic relationship between the execution time of the Expected Externality Scheduler and the number of jobs currently waiting in the batch queue. Figure 4.3 depicts the results of our experiments to test this. The solid line connects the data points that we measured as we increased the number of jobs in the queue. The dashed line is an overlaid cubic function ($y = 0.5947 * x^3 + 3424.42$) that tightly fits the data with a Root Mean Square Error of 4549.13. From the figure, as we increase the number of jobs waiting in the queue, the running time of the algorithm scales in a manner consistent with that predicted by theory.

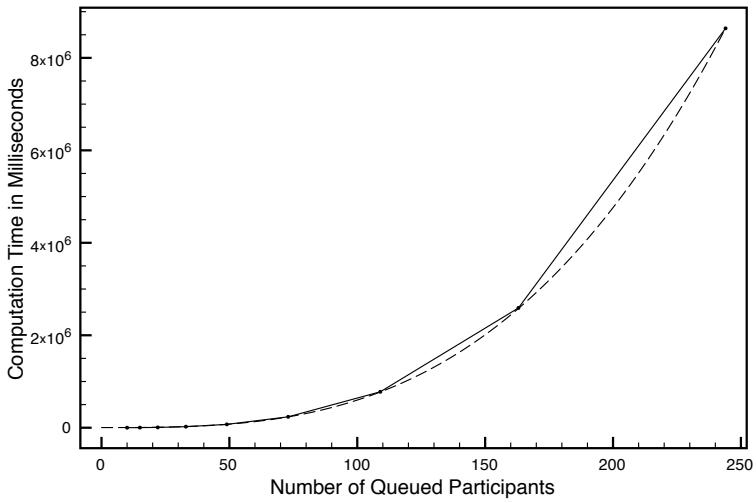

Figure 4.3: Demonstration of the $O(n^3)$ running time of the mechanism.

Some readers may notice that, while the results scale well, the absolute amount of execution time required is quite high (to compute the payments with 244 queued jobs, the simulator ran for over two hours). It is important to note that this absolute execution time depends on the granularity of the expected value computations being performed. The tests in Figure 4.3 were done with very fine-grained expected value computation and performed 1000 draws from the associated distribution for each of such computations.

In Figure 4.4, we demonstrate that this execution time can be reduced by choosing a more coarse-grained level of expected value computation. We tested the amount of time required by the mechanism on a simulated queue of 32 jobs as we scaled the degree of granularity. Each data point in the graph represents

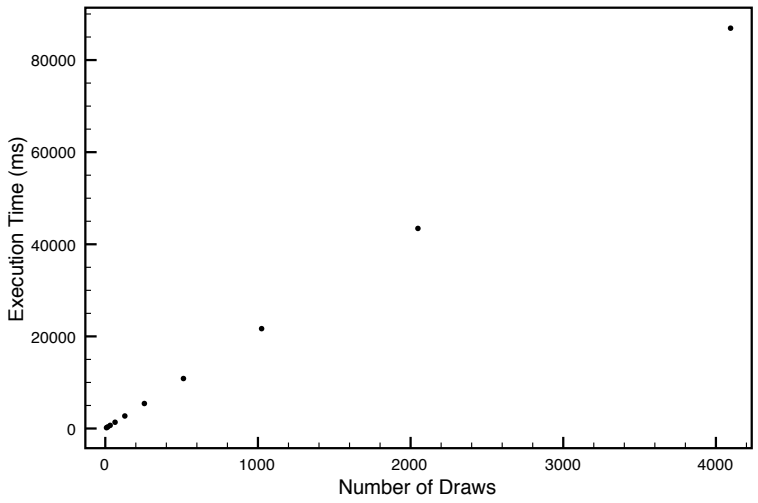

Figure 4.4: Execution time with respect to the number of expected value draws

the execution time for each granularity level of expected value computation. The figure clearly shows a linear relationship between the number of draws and the resulting execution time.

4.2.3 Budget Balance

This reduced granularity does not come without consequences, however. In Figure 4.5, we show the relationship between the expected value granularity level and the degree to which the payments in the system balance out. Each dot in the graph is the budget discrepancy divided by the total magnitude of payments in the system. As described in Section 3.4.1, one of the advantages of the Expected Externality Mechanism is that the sum of the payments in the system is zero. As

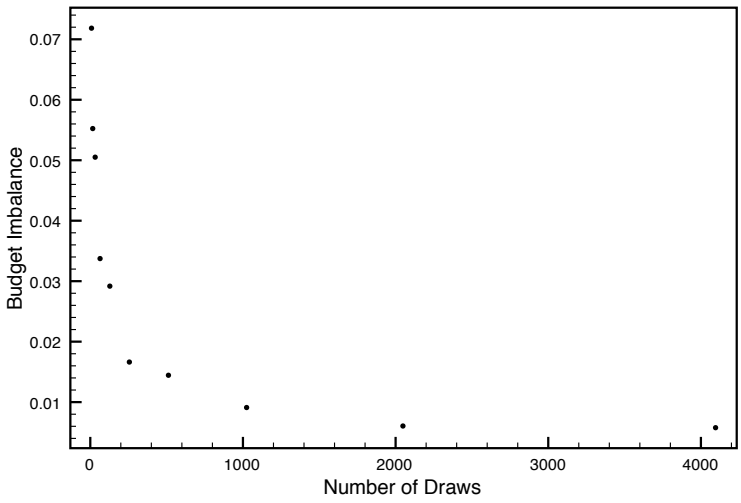

Figure 4.5: Budget imbalance with respect to the number of expected value draws.

we reduce the granularity, and as a result the accuracy, of these computations, the consequence is a greater degree of budget imbalance. In this example, we found a budget discrepancy of 7% when we used a very coarse-grained number of 8 draws. The discrepancy continued to drop to below 1% for 1024 draws and appears to asymptotically approach 0% as we increase this granularity further.

We see this performance tradeoff as a strength of the system. The expected value granularity level provides a tunable parameter that allows us to trade payment accuracy for increased performance.

Chapter 5

Eliciting Honesty in Reservation Systems

The best-effort batch queues discussed in the previous chapter are undoubtedly the most common mechanism used today to allocate grid resources. Reservations systems are a different approach, and while currently used to a lesser degree, their use is increasing. As a result of their newness, some reservations made today at supercomputing centers require direct administrator action to implement, although this is likely to change in the future as these systems mature.

Like best-effort batch queues, reservation systems offer users a simple usage model. Users make a request for resources to the system, and get back a guaranteed time slot. This runtime certainty is a convenience for users that batch queues don't provide. Resource administrators have historically been averse to reservation systems, however, because giving users guaranteed time slots can lead to gaps between jobs and, as a result, lower resource utilization statistics.

As we did with batch queues in the previous chapter, we would like to be able to have greater certainty in the accuracy of the metadata submitted with jobs. Accurate metadata would allow us to schedule more accurately, as well as make scheduling decisions based on previously-unavailable information such as job importance.

From a mechanism design perspective, however, reservation systems are different than batch queues. Because of the strongly negative externality that consumption incurs on others (as discussed in the previous chapter), the preferences of users in a Batch Queue have externalities that do not exist in a reservation system. Different uses of a resource at time t_i has no effect on the happiness of a different consumer of the resource at other times. The preferences in reservations systems do exhibit more complex complementary and substitutive properties, however. If we discretize the schedule of each resource, adjacent time units tend to be desired by the same user (a complementary relationship), and many different sequences of time blocks can satisfy the same job (a substitutive relationship).

A simple approach to this problem of applying a truth-revealing pricing mechanism to the allocation of grid reservations, would be to discretize the time schedule of each resource and to apply a combinatorial GVA to allocate it. For example, user who wanted 3 hours of time before noon would place multiple, mutually exclusive bids of ($1AM$ and $2AM$ and $3AM$ for \$5), ($2AM$ and $3AM$ and $4AM$

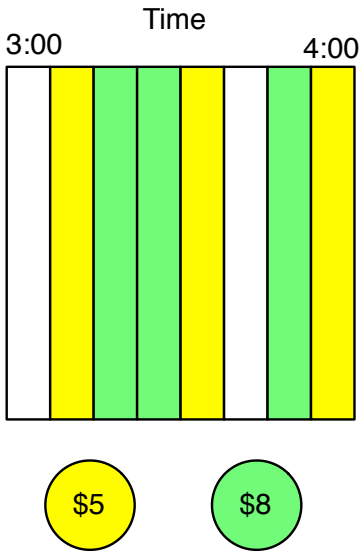

Figure 5.1: A depiction of the type of bids that can be expressed by a user in a combinatorial auction.

for $5)..., etc. This technique loses no expressiveness and would achieve efficient allocations, but would be NP-Hard to solve.

To take this approach would be to unnecessarily reduce the problem to one with excessive expressiveness. In the DPGVA mechanism, bids can only request contiguous segments of time and are indifferent between when they are (subject to the deadline constraint). A combinatorial auction does not have these constraints, and is a stronger tool than is required. Instead of going this route, we have developed a special-purpose dynamic programming algorithm for optimally solving this scheduling problem in pseudo-polynomial time.

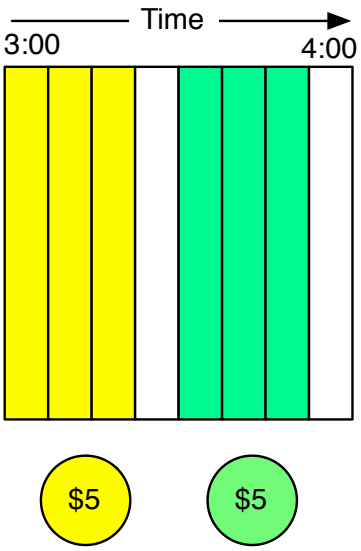

Figure 5.2: A depiction of the type of bids that can be expressed by a user in the DPGVA.

For example, compare Figures 5.1 and 5.2. In Figure 5.1, we see that the Combinatorial Auction reduction allows us to place bids for any collection of time slots, allowing non-contiguous bundles of time slots and allowing bundles to have different values. Figure 5.2, on the other hand, shows how the DPGVA system allows only contiguous time slots to be bundled, and we are indifferent to multiple time slots (subject to deadline constraints). This stepping-back from a combinatorial GVA, and reducing the expressive power of the bids to something more suiting grid reservations, allows us to avoid the intractability of a combinatorial GVA.

In addition to allowing only bids that suit grid reservations, the DPGVA reservation system makes two simplifications that allow us to deliver a tractable, yet truth-revealing solution: we mandate that the number of nodes requested by a user must be equal to the number of nodes we are scheduling over, and we restrict the granularity with which users specify time-units. The reason for both of these restrictions will be discussed at greater length below.

5.1 Our Reservation System

5.1.1 What users see

In the DPGVA reservation system, users place bids for computational time. In these bids, users specify three values: the length of time needed (l), the deadline by which the job must be completed (d), and the value of the work being performed (v). Users are guaranteed one of two outcomes: either they will get no computational time and will be charged nothing, or they will get exactly l amount of time by the deadline d and will pay at most v. These bids are sealed, and must be submitted before the scheduling period begins (*i.e.* if each day is scheduled at midnight, the bids must be submitted the previous day).

In our current implementation, users place bids today for reservations scheduled tomorrow. In our simple bidding program, users simply indicate to the

scheduler the amount of time they need, when they need it by, and the most they are willing to pay for it. For example, a user that wished to request three hours (180 minutes) of computing time, ending at 8am on November 21st, 2008, and was willing to pay at most 500 credits for the time would submit:

```
$placebid 180 "2008.11.21 at 8:00:00" 500
```

One limitation of our scheme that will be explained at length below is that we enforce some maximum granularity on the length of jobs and the deadlines that can be specified. This granularity can be modified by administrators. For example, if the maximum granularity were set to 15 minutes, allowable job lengths would be 15, 30, 45, 60... minutes and allowable deadlines would be 1:00AM, 1:15AM, 1:30AM, 1:45AM,... etc.

The scheduler collects many such bids throughout the day, and at midnight it computes the next day's schedule. This reservation schedule is communicated to PBS using the *setres* command. Users can then use the PBS command *showres* to find when their job's reservation has been scheduled.

5.1.2 Schedule Determination

Our dynamic programming algorithm builds a table of intermediate results, starting on simpler instances of the problem and combining them to ultimately solve the whole problem. As can be seen in Figure 5.3, this table has two axes, one

being time and the other jobs. The time axis represents every time unit *granule* during the discretized scheduling period. The job axis has one entry for each job request submitted to the system. Each cell in the table (J_i, T_k) holds the answer to the question: what is the optimal scheduling of jobs from time T_k to time T_n if we consider only jobs J_1 through J_i (jobs are sorted by deadline). As such, when the algorithm has completed, the answer to the entire optimal scheduling problem will be held at (J_m, T_1).

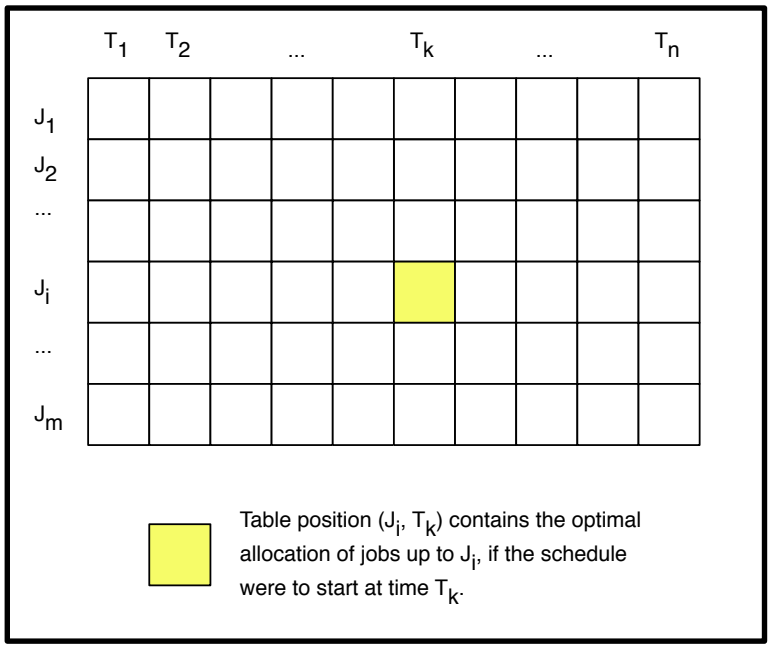

Figure 5.3: The structure of the table that stores intermediate results.

We build this table by starting at (J_1, T_n), and go from right to left, and then top to bottom. The recursion relation (as depicted in Figure 5.4) is as follows:

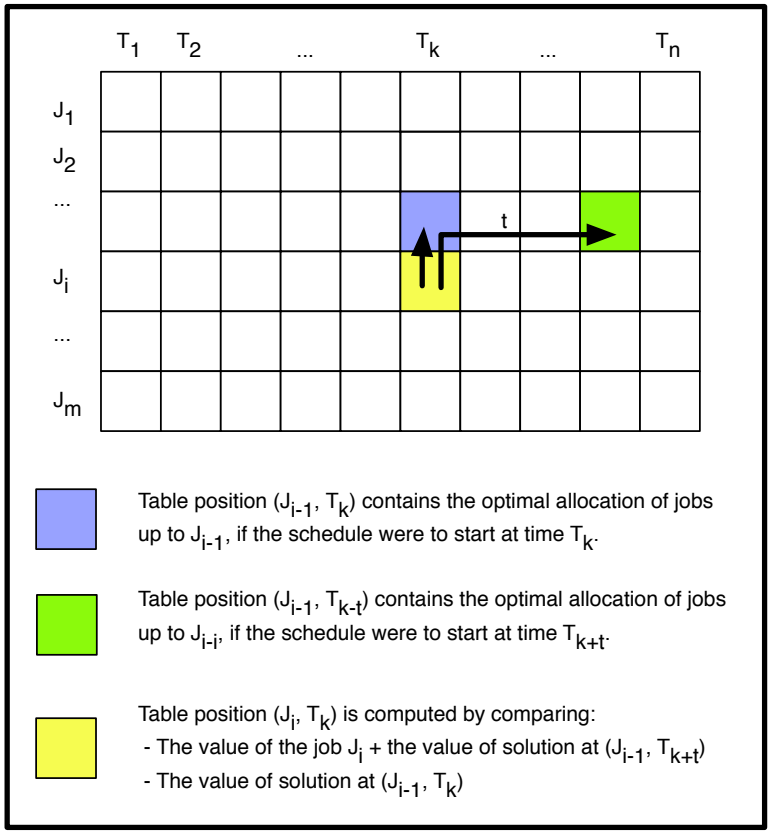

Figure 5.4: The recursion relation. Each cell in the table is computed based on the results computed in other cells.

```
compute(t, j){
if t=n, return empty;
if j=0, return empty;

a = value(j) + value(table(t + length(j), j-1));
b = value(table(t, j-1));

if (t+length(j)) > deadline(j)
return table(t, j-1);

if a > b,
return j + table(t + length(j), j-1);
```

```
else return table(t, j-1);
}
```

Each cell contains the optimal solution to the sub-problem characterized by its cell location (i.e. a list of jobs). The recursion relationship is simple. To determine if j_i is in the optimal solution of the sub-problem (J_i, T_k), we compare two different sub-problems: (J_{i-1}, T_k) and (J_{i-1}, T_{k+t}) (where t is the running time of job j_i). If the total value of sub-problem (J_{i-1}, T_k) is greater than the total value of (J_{i-1}, T_{k+t}) plus the value of j_{i-1}, then j_{i-1} is not included in the solution of (J_i, T_k) and we use the result from (J_{i-1}, T_k). Otherwise, we take the result from (J_{i-1}, T_{k+t}) and add j_{i-1} to it.

As a result of the order in which we build the table, when computing each cell, the cells it relies upon have already been computed. Consequently, the amount of time needed to compute each cell is proportional to the size of the result stored, and since this can be (worst-case) the number of jobs submitted, this time scales $O(m)$ linearly with respect to the number of jobs submitted.

So, because the table is of size $n * m$, and each step can take a maximum of $O(m)$ time, the complexity of building the table (and solving the problem) is $O(n * m^2)$.

One limitation of our mechanism as it currently exists is its inability to handle jobs that require different numbers of machines. So the mechanism works perfectly

for allocating, say, 8-node jobs on 8-node resources, but does not currently handle jobs of arbitrary size. This limitation is due to us currently lacking a dynamic programming algorithm that allows arbitrarily-sized jobs.

As with other pseudo-polynomial algorithms, the actual magnitude of inputs (e.g. job deadline, schedule size) has an impact on the running time of the algorithm. As a result, one parameter that must be set when using the DPGVA algorithm is the maximum specifiable granularity. This allows the mechanism to schedule k discrete time units in the same amount of time, regardless of if those units are sized in seconds or in days.

5.1.3 Payment Computation

As discussed in Chapter 3, the GVA payment expression can be represented as (for more on this expression, please see Section 3.4.2):

$$\mu_i(s_i, s_{-i}) = \sum_{j \neq i} U(s_j, \kappa^*(0, s_{-i})) - \sum_{j \neq i} U(s_j, \kappa^*(s_i, s_{-i}))$$

In this expression, κ^* is the scheduling algorithm described in the previous section.

U is the codification of the user's preferences. As described above, we consider a user to get v value if his job runs before its deadline, d, and gets l amount of time to run. So we can express U as:

$$U(s_i, \kappa^*(t_i, s_{-i})) = \begin{cases} v_i, & \kappa^*(t_i, s_{-i}) = runs \\ 0, & \kappa^*(t_i, s_{-i}) = discards \end{cases} \tag{5.1}$$

Once we have computed the schedule, κ^*, this utility function takes constant time to compute. For m bids in the system, we need to compute $m + 1$ different κ^* allocations, one actual schedule, and m hypothetical schedules, each assuming a different user had not participated. So, since we need to compute κ^* a total of m times, and (as described above) each κ^* computation takes $O(n * m^2)$ time, we expect the GVA payment computation to scale $O(n * m^3)$.

It should be noted that the schedule is already known after the actual κ^* is computed, and it's not necessary to wait until the payments have been computed in their entirety to start jobs running. This allows us to start the schedule after a relatively quick $O(n * m^2)$ schedule computation, and then perform the slower $O(n * m^3)$ payment computation in parallel (or even at a later date).

5.2 Results

5.2.1 Incentives

Without exhaustively testing the complete space of possible user behavior, we can't experimentally *prove* that a mechanism always incentivizes honesty. But while we can't *prove* this experimentally, we *can* use simulated experiments to repeatedly verify that this is indeed the case in practice. In this section, we'll present the results of such simulations.

Figures 5.5, 5.6 and 5.7 show the results of one such simulation. In these tests, we've created a pool of competing jobs and watched what happens to an individual's satisfaction with the allocation as we vary what he reports to the mechanism. A user's payoff is defined as his value of the amount of compute time he received (in currency), minus the amount he had to pay into the system to get it.

In all of these tests, there were 20 competing jobs. The number of units of time that were being scheduled was 100. The deadline for these randomly created jobs was selected from a uniform distribution of 0 to 100. The length of these jobs was selected from a uniform distribution of 0 to 100. The value parameter for these jobs was selected uniformly from 1 to 1000.

Figure 5.5 shows the change in user satisfaction from varying the declaration of the deadline of the job in question. This figure answers the question, "what will happen to my satisfaction level if I were to lie about my deadline?" The deadline of the job in question is in fact 90, and as this figure shows, reporting a deadline other than 90 leads to the same or lower payoff. We don't explore the payoff of declaring deadlines greater than 90, because the user would potentially obtain allocations that were useless (allocations that did not satisfy the user's deadline).

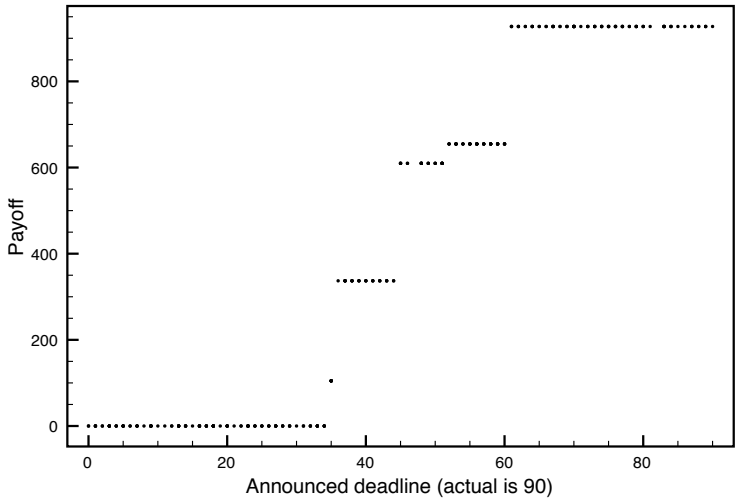

Figure 5.5: The payoff seen by the simulated user over a range of possible deadline declarations.

Figure 5.6 shows the change in user satisfaction from varying the declaration of the needed running time of the job in question. This figure answers the question, "Do I see any benefit if I exaggerate the amount of time I need?" The actual length

of the job is 5, and as this figure shows, reporting a longer running time only leads to a less desirable outcome. We don't explore declarations smaller than 5, due to the fact that jobs are allocated exactly their stated running time. Declarations smaller than the actual running time would be useless to the job owner.

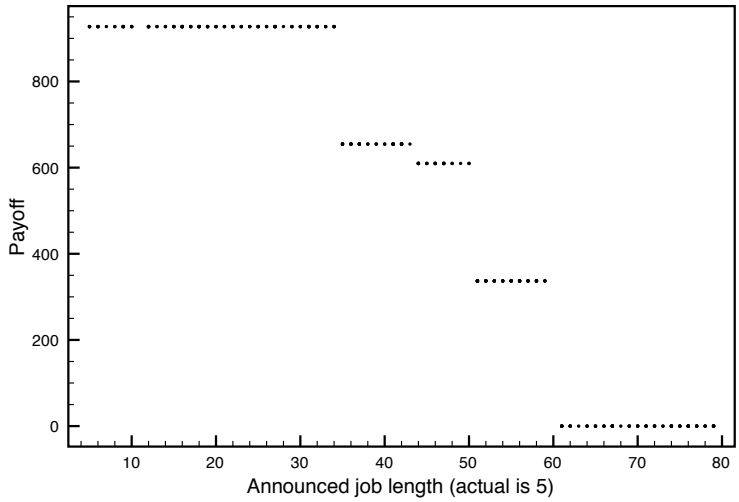

Figure 5.6: The payoff seen by the simulated user over a range of possible job length declarations.

Figure 5.7 shows the change in user satisfaction from varying the value declared when submitting the job in question. This figure answers the question, "Will I benefit from being dishonest about the true value of my job?" The actual value of the job is 950, and as this figure shows, reporting either a higher or lower value does not improve the outcome for the job.

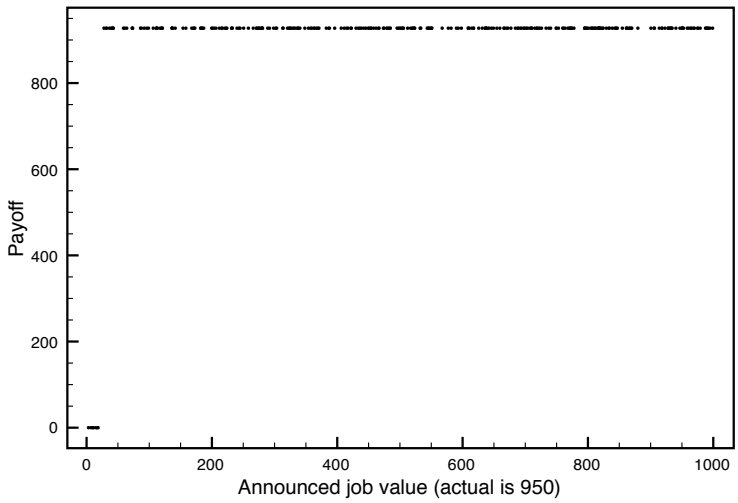

Figure 5.7: The payoff seen by the simulated user over a range of possible job value declarations.

5.2.2 Performance

In Section 5.1.3, we discussed how the DPGVA scheduling algorithm should provide good scaling properties. In this subsection, we will present the results of our tests on live servers to verify this. All of the following results were measured on a quad-processor Intel Xeon running at 3.2 ghz with 1GB of RAM. The DPGVA mechanism was implemented in Java, and executed on the Java HotSpot Client, version 1.6.0.

As currently implemented, the DPGVA mechanism runs in a separate address space from the rest of PBS. It exists as two components: a continuously running server and briefly running clients that submit bids. The server component listens

on a TCP port for client bid submissions throughout the day, and at the end of the day executes the DPGVA algorithm to compute the next day's schedule and the corresponding user payments. The server then constructs a series of PBS commands using the "setres" command to communicate the next day's reservations to the PBS scheduler. Users can then user the PBS command "showres" to find out when they have been granted access to the resource. For our tests, the DPGVA software was run on the same hardware as the rest of the PBS software. This is not necessary, however, and the DPGVA computations can be executed on separate hardware.

Theory predicts that as we increase the number of bids submitted to the mechanism, the amount of time necessary to compute the schedule scales $O(n^2)$. In Figure 5.8, we present the measured results of what happens to computation time as we increase the number of bids. In these tests, we schedule a six hour period using a maximum specifiable time granularity of 15 minutes.

In addition to computing the schedule, the mechanism must also compute the payments seen by each user. As discussed above, this computation involves scheduling n different instances of the problem, each one supposing a certain bid had not been present. As a result, computing the payments takes n times longer than computing the schedule, yielding scaling complexity $O(n^3)$ with respect to the number of bids. In Figure 5.9, the performance measurements for this com-

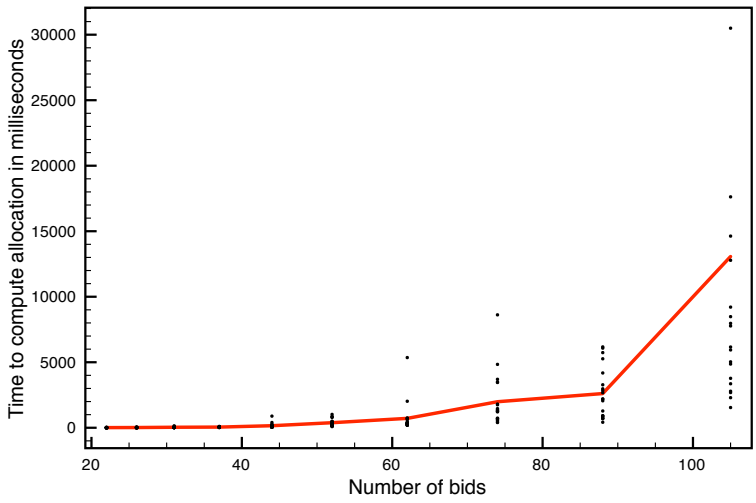

Figure 5.8: The amount of time required to find the allocation of jobs, as we vary the number of bids submitted.

putation can be seen. It should be noted that while these absolute numbers are somewhat high, the job schedule does not depend this computation. So the jobs can immediately begin running after computing the schedule, and the payments can be computed separately in parallel.

As a demonstration of how positive these results are, we have recorded the amount of time needed to do the GVA computations using a traditional brute-force technique, rather than our dynamic programming solution. This brute-force algorithm simply considers all possible orderings of jobs (subject to deadline constraints), and selects the one with the highest aggregate job-value. As can be seen in Figure 5.10, the amount of time needed for an exhaustive brute-force

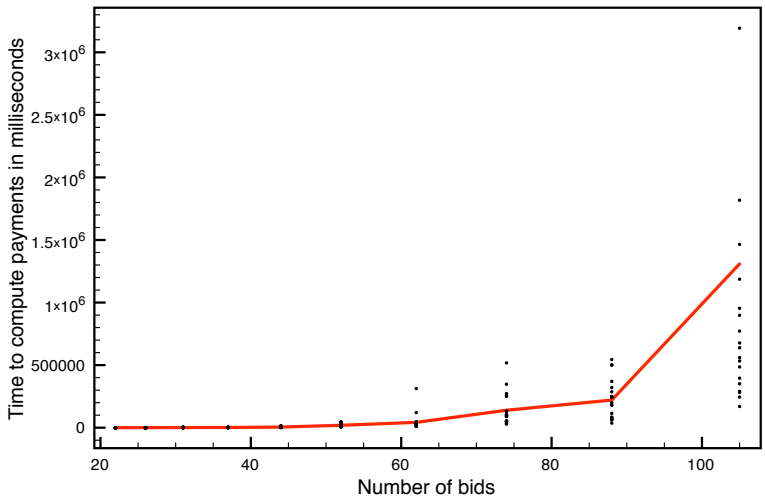

Figure 5.9: The amount of time required to find the payments of each user, as we vary the number of bids submitted.

computation of the GVA quickly explodes as we increase the number of bids in the system. Even at only 44 bids in the system, the brute-force algorithm requires almost 8 minutes to compute the same set of payments that our DPGVA algorithm finds in under 6 seconds.

One parameter that system administrators would likely want to adjust is the time period over which the scheduling occurs. Depending on the target environment, they may want to be scheduling one day at a time, or one month at a time. The analysis presented in Section 5.1.3 indicated that the running time of the algorithm should scale $O(n)$ linearly as we increase the time period over which we are scheduling. As you can see in Figures 5.11 and 5.12, we see this linear increase

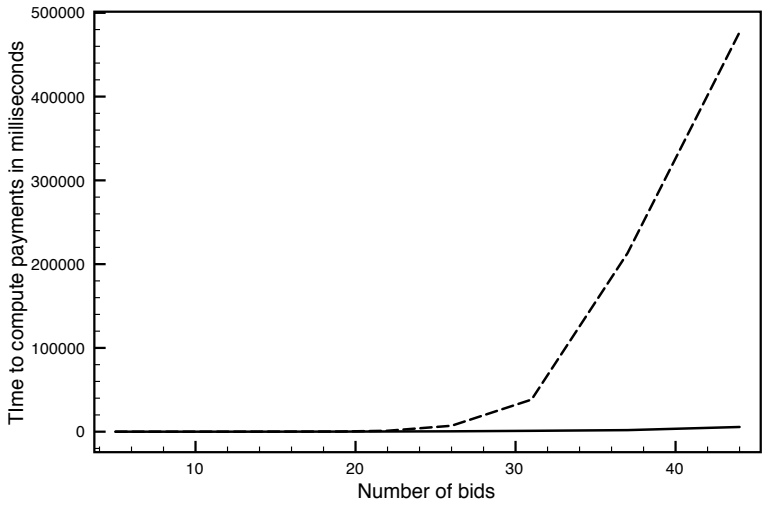

Figure 5.10: Comparing the performance of a brute-force GVA implementation to the DPGVA algorithm, while computing the payments. The dashed line is the brute force algorithm and the solid line is the DPGVA results.

in practice. In these tests, we schedule 20 jobs using a maximum specifiable time granularity of 15 minutes. Figure 5.11 presents the running time measured to schedule the jobs as we increase the time period over which we are scheduling. Figure 5.12 presents the running time measured to compute each users payemts as we increase the time period over which we are scheduling. As you can see from the results, both increase roughly linearly.

Another parameter that system administrators would likely be modifying is the maximum time granularity used when users submit bids. Increased granularity allows users with very small or very predictable jobs to get no more resource than they need, while decreased granularity allows faster schedule computation.

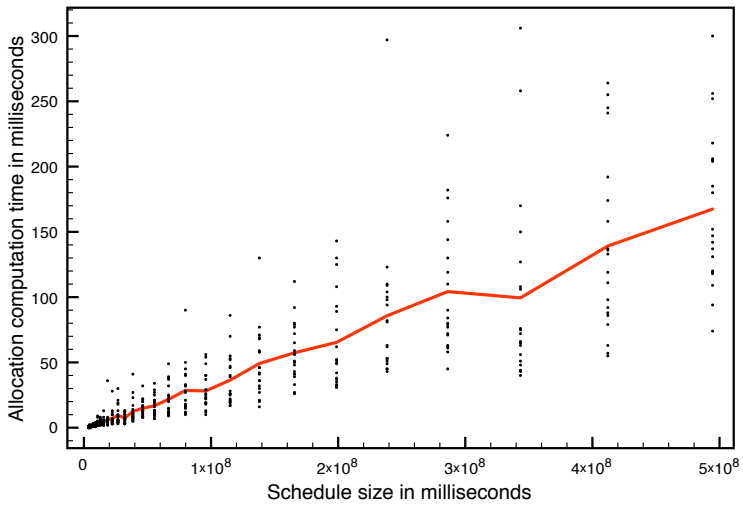

Figure 5.11: The amount of time needed to compute the schedule, as we increase the length of time over which we are scheduling.

Figures 5.13 and 5.14 depict what happens as we increase the size of the smallest schedulable unit. In these tests, we are scheduling 20 jobs over a six hour period. The smallest schedulable unit is varied from 60 seconds to 900 seconds (one minute to 15 minutes). Because the size of the smallest unit is inversely proportional to the number of distinct schedulable time units, we would expect this data to resemble the reciprocal of the data presented in Figures 5.11 and 5.12. As you can see in the data, this is indeed the case, as both figures seem to scale $O(1/n)$ with respect to the size of the smallest unit.

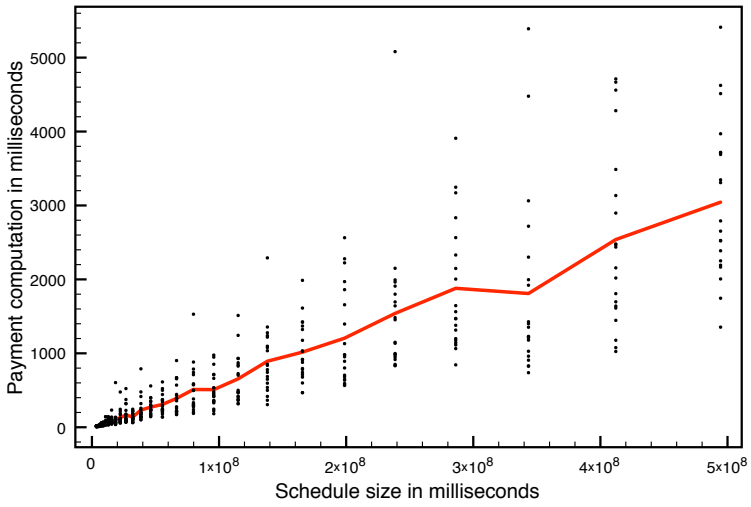

Figure 5.12: The amount of time needed to compute the payments, as we increase the length of time over which we are scheduling.

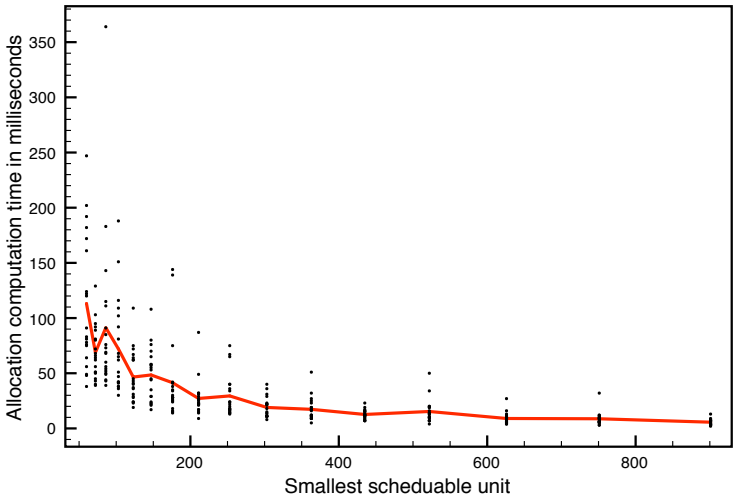

Figure 5.13: The amount of time required to find the allocation of jobs, as we vary the size of the smallest scheduable unit.

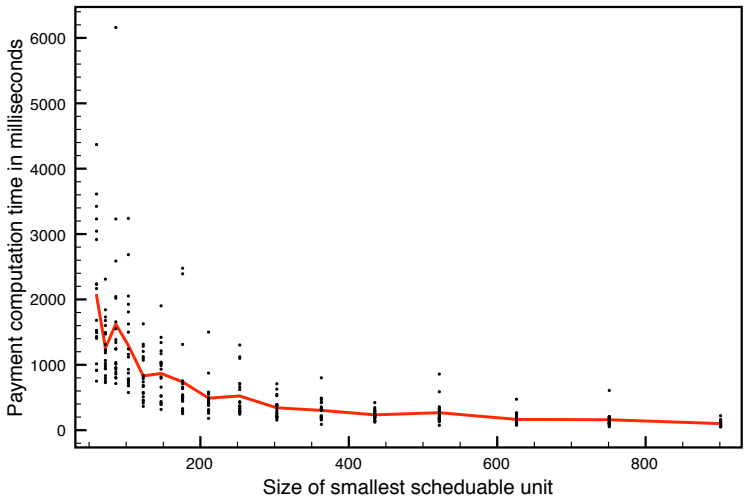

Figure 5.14: The amount of time required to find the payments of each user, as we vary the size of the smallest scheduable unit.

Chapter 6

Relaxing Theoretical Constraints: Batch Queued Systems

As discussed in Chapter 3, both the Expected Externality Mechanism and the Generalized Vickrey Auction assume that the allocation function in use is completely optimal. In chapter 4, we presented the results of our application of the Expected Externality Mechanism to allocation in Batch Queues. In that chapter, we remained completely faithful to that requirement by restricting allowable jobs to only those that allowed us to tractably solve the allocation function κ^* optimally. Specifically, we mandated that all jobs in the queue require exactly the number of nodes available on the resource.

That restriction however, makes the scheme impractical for use on today's computational grids. Current batch queues weave together the execution of jobs of many sizes, while using backfilling techniques to fit small jobs between the gaps

of other larger jobs. Resource providers today expect a resource management scheme to multiplex jobs of many different sizes on to one set of resources.

In this chapter, we examine what happens when we knowingly break the constraints of the EES mechanism in order to deliver this increased functionality. We do this by replacing the optimal allocation function κ^* with a heuristic, κ', which provides results that are feasible but not optimal. For a diagram that graphically depicts the relationship of this chapter to the chapters that surround it, see Figure 1.2. The heuristic that we have chosen is not an approximation algorithm, as it makes no claims about how bad the worse-case results it provides are. We do not see this as a serious weakness, however, because being able to bound the accuracy of κ' would still violate assumptions made by the theoretical proofs of the mechanism.

Adding the ability for users to indicate different job "widths" is not the only feature the EES lacks that administrators would want. Administrators currently have the ability to multiplex multiple queues on to one set of resources, which is a feature the EES lacks. We believe this is somewhat mitigated, however, by the fact that multiple queues are frequently used as a priority mechanism (i.e. a high- and a low-priority queue). Since the EES has value and urgency semantics built in, multi-queue features may not be necessary for some administrators.

6.1 Relaxed-Constraint EES

6.1.1 What Users See

The relaxed-constraint EES has very similar semantics to the non-relaxed version presented in chapter 4. Users submit four parameters along with every job: called v, d, r, and n. The scheduler considers the jobs in First-in-First-out (*FIFO*) fashion, and based on the submitted parameters, decides to either run the job or to discard it. Like the traditional EES, a job accrues currency while waiting in the queue and pays out currency if it runs when it reaches the front of the queue, the net effect being that jobs which are eventually discarded by the mechanism gain currency overall, and those that are run by the mechanism lose currency.

The v, d and r parameters are identical to the traditional EES. The first two parameters are currency-based and indicate job priority: the value of the work to be performed. v, and the tolerance of the user towards delay in total turnaround time, d. Both v and d are specified in terms of a cluster-specific currency that has been allocated by the administrators of the resource. The parameter r is the expected running time of the job, in seconds. The parameter n, not present in the traditional EES, is the number of nodes required by the job. Unlike the traditional EES, The relaxed EES makes no restrictions regarding the allowable

nodes requested by a job (other than being no greater than the number of available nodes). For more information on the parameters v, d and r, see Section 4.1.1

6.1.2 Schedule Determination

When each job reaches the front of the queue, it is considered for running. This decision is driven by the parameters associated with that job and the parameters associated with the other jobs waiting in the queue.

Intuititvely, this decision function compares the value of the job in question with the collective inconvenience borne by the other jobs waiting in the queue. Symbolically, the decision to run job i can be represented as such:

$$a = v_i \tag{6.1}$$

$$b = \sum_{j \neq i} d_j * delay_j(i) \tag{6.2}$$

$$\kappa^{'}(a,b) = \begin{cases} discards, & a < b \\ runs, & a \geq b \end{cases} \tag{6.3}$$

Where, $delay_j(i)$ is the difference between the starting time of job j in the best-effort schedule (described below) including i and the starting time if i is

discarded. So intuitively, $d_j * delay_j(i)$ is a measure of how inconvenienced the owner of job j is by the running of job i. If $a \geq b$, job i is run by the scheduler. Otherwise, job i is discarded and the next job in the queue is considered.

6.1.3 Computing Delay

In order to determine the inconvenience borne by other participants in the queue, $delay_k(i)$, we need to estimate the increase in turnaround time that some job k will see, due to the running of job i. Determining this delay with complete accuracy is a difficult problem. To demonstrate why, consider jobs i, j and k, with job i at the front of the queue, and jobs j and k further back, with j before k. To decide if job i will run, we need to know the impact it will have on the turnaround time of job k. To compute this, however, we'd have to know the schedule of jobs that would result if i alternately does and does not run. Knowing that schedule would require us to know whether or not job j (before k) runs. Determining that, however, requires us to know the impact j would have on k if it alternately did and did not run. As a result, the brute force approach to computing this requires considering sub-problems numbering in proportion to the power set of the set of jobs in the queue. Since considering $|2^I|$ subproblems would not be tractable, and since we are unaware of a more sophisticated algorithm that can solve this problem more quickly, we will estimate this value.

In order to estimate the delay effect that job i has on job k, we make the simplifying assumption that all jobs in the queue between i and k will run. As a result, our estimation becomes the difference between two schedules: one schedule being the greedy insertion of all jobs into the soonest available spot (in queued order), and the other being the same insertion technique, but with job i removed.

To do this, we use a data structure we refer to as the SegmentedFunction data structure. This data structure allows us to answer the question: "what is the soonest window where n nodes are available for t seconds?" It is a linked-list, where each node represents a segment of the positive real domain of a function (in our use, the function returns the number of occupied nodes at a given time). This list keeps the segments in sorted order, starting at 0. To add a segment to this data structure, we traverse the list to find the affected nodes, and insert a new node segment (thus taking linear time). To lookup the node availability at a given moment, or to find the earliest window of n available nodes, we do a similar list traversal, also taking linear time. Segment removal is not a feature we need to support for our uses.

As a result, the time required to construct the schedule of a set of jobs I in this manner (finding the first available n nodes, and inserting a new segment) takes $O(|I|^2)$ time. Because the two schedules (containing and not containing i) can be

reused when estimating the delay, computing b for use in equation 6.3 requires only $O(|I|^2)$ time.

6.1.4 Payment Computation

As discussed in Section 4.1.3, we are using the Expected Externality Mechanism to provide payments that will incentivize truth-telling in batch-queued systems. The theoretical properties and history of this mechanism are discussed in Section 3.4.1.

The Expected Externality Mechanism is budget balanced. In our scheduler, running jobs pay currency to jobs that are waiting in the queue. The net effect of this is that a job that waits in the queue and does not end up eventually running leaves with more currency than it entered with. A job that eventually runs leaves with less currency than it entered with.

Symbolically, the payment made by user i is (following notation from [9]):

$$\left(\frac{1}{I-1}\right)\sum_{j\neq i} E_{s_{-j}}\left[\sum_{l\neq j} U(s_l, \kappa'(t_j, s_{-j}))\right] - E_{s_{-i}}\left[\sum_{j\neq i} U(s_j, \kappa'(t_i, s_{-i}))\right] \quad (6.4)$$

where for a job l at the front of the queue,

$$U(s_l, \kappa^{'}(t_j, s_{-j})) = \begin{cases} v_l, & \kappa^{'}(t_j, s_{-j}) = runs \\ 0, & \kappa^{'}(t_j, s_{-j}) = discards \end{cases} \tag{6.5}$$

and for any job l not at the front of the queue,

$$U(s_l, \kappa^{'}(t_j, s_{-j})) = \begin{cases} -d_l * delay_l(i), & \kappa^{'}(t_j, s_{-j}) = runs \\ 0, & \kappa^{'}(t_j, s_{-j}) = discard \end{cases} \tag{6.6}$$

In this expression, $\kappa^{'}$ is the decision function described in Formula 6.3. t_m is the parameter stated by job m (in the case of the frontmost job, this is the v, for all other jobs this is d). In this expression, $E_{s_{-j}}(x)$ refers to taking the expected value of x over the possible parameter announcements of all jobs other than j. In this sense, t_m and s_m are variables of similar type, the difference being that t_m is the actual announced parameter of user m, whereas s_m is the parameter corresponding to job m that is drawn during the expected value computation. I is the number of jobs in the queue. The function u quantifies the amount that a user is affected by the allocation decision $\kappa^{'}$ in terms of currency.

We expect the computation of the payment function to scale $O(n^4)$ cubically as we increase the number of jobs in the queue. In Formula 6.3, computing $\kappa^{'}(t_j, s_{-j})$

takes $O(n^2)$ time and it is nested inside two loops that each scale linearly with n. In fact, the allocation $\kappa'(t_j, s_{-j})$ is the same for each of those innermost loops, and is reused each time, so the scaling has the potential to be $O(n^3)$. Unfortunately, the data structure we have used to store these intermediate schedules requires linear-time lookup, so we see $O(n^4)$ scaling. This is discussed further in Chapter 9.

If κ' was making scheduling decisions on perfectly accurate information, rather than on estimations, the Expected Externality Mechanism would be expected to incentivize both truth-revelation and budget balance. In the following section we will investigate to what degree these features remain, despite our breaking the requirement of optimality.

6.2 Results

In order to verify that our scheduler exhibited the characteristics predicted by theory, we developed a simulator to test it. In simulation, we have the ability to repeatedly run identical scenarios and compare the payoff seen by a user over a range of different declarations. This allows us to experimentally determine if honesty is really the optimal strategy when interacting with these payment mechanisms, and to examine how the performance scales as we increase the number of jobs in the system.

The following experiments were conducted on a quad-processor Intel Xeon running at 3.2 ghz with 1GB of RAM. The operating system was Debian Linux version 1:3.3.5-13. The simulator was written in the Java programming language and was compiled and run with the Java HotSpot Client, version 1.6.0.

6.2.1 Performance

In Section 6.1.4, we discussed why we expect to see the EES payment computation scale $O(n^4)$ as we increase the number of queued jobs. Figures 6.1 and 6.2 depict the results of our performance tests. In Figure 6.2, we present the amount of time needed to compute the payments in the system, as we increase the number of jobs in the queue. As you can see, it takes more than 30 minutes to compute the payments for a job queue of length 64. While this is a somewhat onerous burden, one thing to keep in mind is that the scheduling decision is computed almost instantly, and it is the payments that take a long time. As a result, there is no delay needed between jobs running, and these payments can be computed in parallel, and (if needed) on separate hardware.

In order to elucidate the scaling differences between the original EES and the relaxed version presented in this chapter, in Figure 6.2 we've taken the data from Figure 6.1 and divided the execution time by the number of $jobs^3$. As you can see, the original EES flattens out at about 0.5, whereas the relaxed EES

continues to grow. This indicates that the relaxed EES scales faster than $O(n^3)$ while the traditional EES does not. In addition, this growth appears linear, which is consistent with our expectation of $O(n^4)$ scaling.

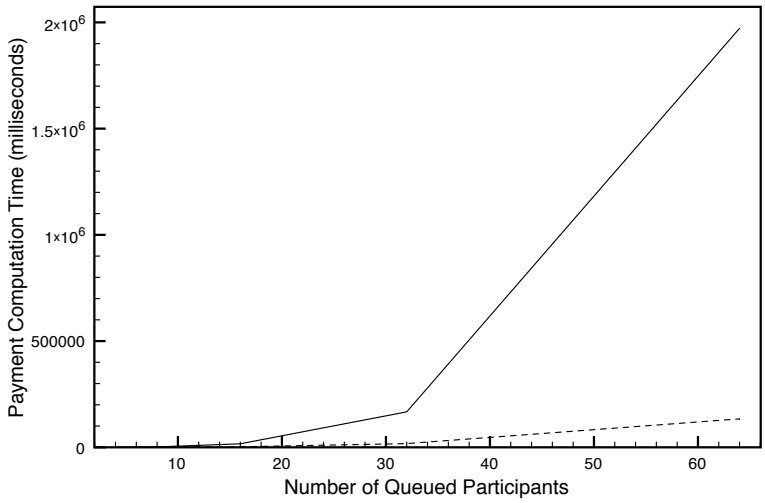

Figure 6.1: The execution time necessary to compute payments, as we increase number of bids. The dashed line is the original, unmodified EEMS.

6.2.2 Incentives

In Chapter 3, we described how the Expected Externality Mechanism provides theoretical guarantees that honest preference revelation is a Nash equilibrium, meaning that it is an optimal strategy if all other participants are being honest. In Chapter 4, we presented results demonstrating that the EES batch scheduler retained these guarantees in practice. In this section, we will present our explo-

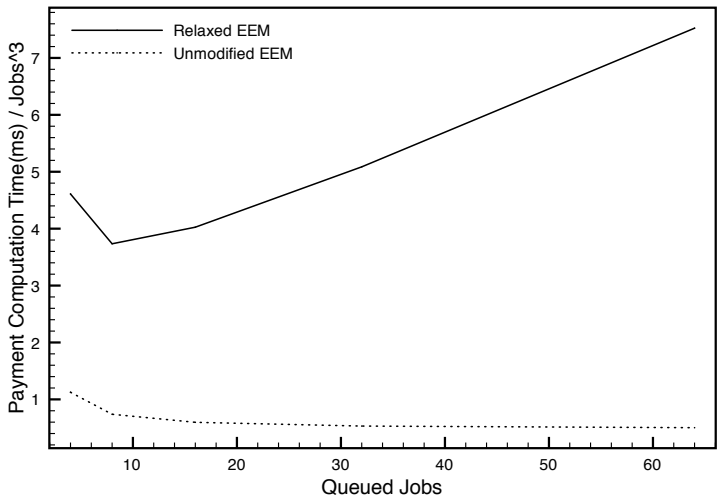

Figure 6.2: The execution time necessary to compute payments, divided by $bids^3$, as we increase number of bids. The dashed line is the original, unmodified EEMS.

ration of the incentives faced by users in the relaxed-constraint EES presented in this chapter.

In order to demonstrate empirically that our mechanism has this property, we created simulated queues and explored the expected payoff for many different preference declarations. It should again be noted here that we cannot experimentally *prove* that these mechanisms always incentivize honesty; that can only be done theoretically. We *can*, however, repeatedly examine whether these preferences hold in practice. In each case we have tested, our results show that expected payoff was indeed maximized when users honestly declared their preferences.

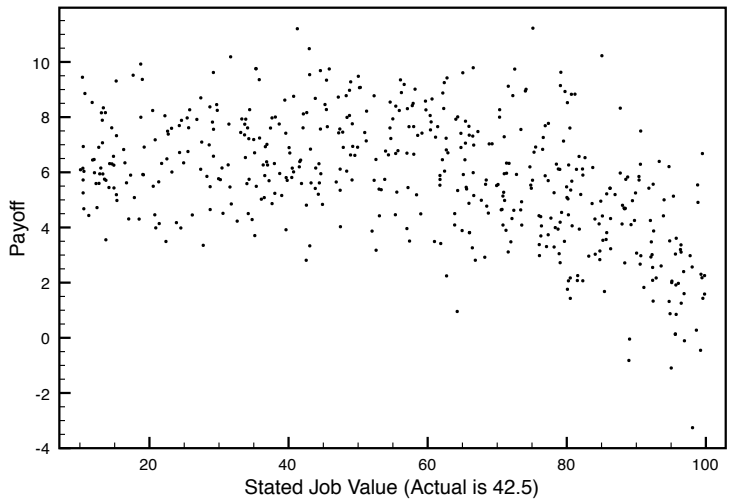

Figure 6.3: The expected payoff for a user over different stated job values.

Figure 6.3 depicts the expected payoff seen by a participant as we explore a range of different possible declarations of the v parameter. If our system were properly enforcing truth-revelation, Figure 6.3 would see the data clearly peak at the actual job value, 42.5. As you can see in Figure 6.3, the data loosely has a regime of indifference up to about 60, after which payoff drops. The participant doesn't have a reason to deviate from the true value of 42.5, but the incentives tat provide this are somewhat weak and noisy. This graph indicates that the estimation technique has introduced a fair amount of noise, but that honesty still roughly corresponds to maximized payoff. Also, outliers can be seen in the graph that occur at non-honest values. These outliers may or may not present tempting

targets to motivate users to be dishonest. This is discussed at greater length in Chapter 8.

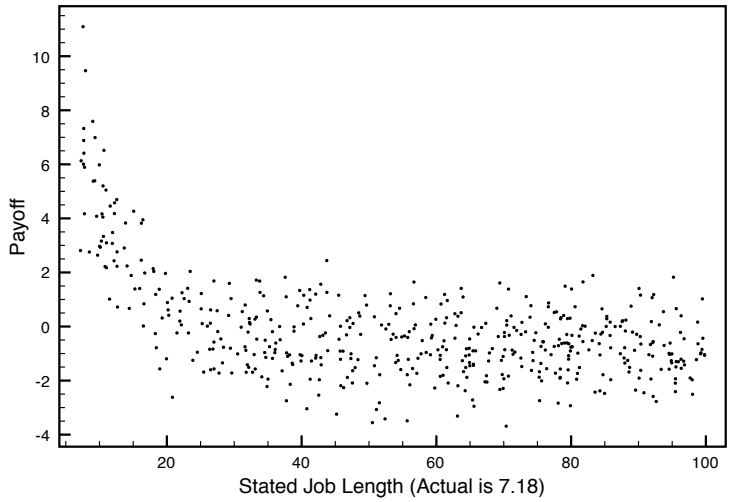

Figure 6.4: The expected payoff for a user over different stated job lengths.

Figure 6.4 is a similar graph, this time exploring the payoff seen over many different stated job lengths. Because the EES will allocate exactly the amount of time requested, we explore only potential length declarations greater than the actual job length. In this case, the actual job length is 7.18 and ideally the maximum would occur at this value. This graph shows much cleaner incentives, as the user indeed sees his total payoff drop off quickly as he exaggerates the amount of time he needs from the system.

Figure 6.5 is a similar graph, this time exploring the payoff seen over many different stated number of nodes required. Similar to the job length parameter, we

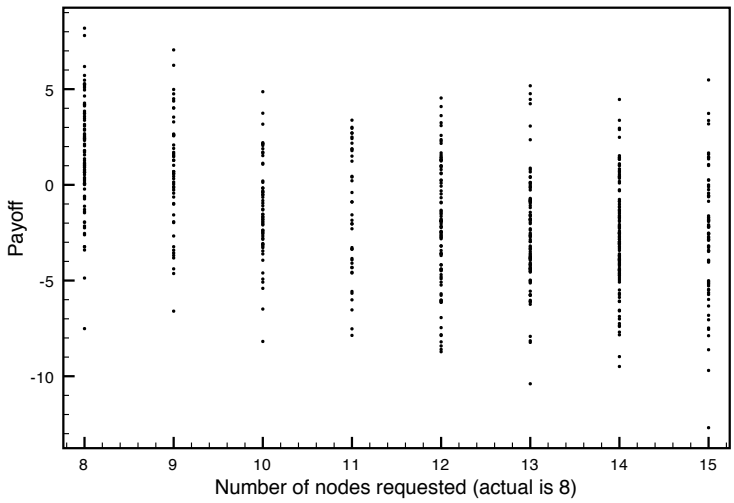

Figure 6.5: The expected payoff for a user over different stated Nodes needed.

only explore the declaration needing nodes greater than the actual nodes needed; in our tests we assume that an 8-node job will not be able to run on 7-nodes. Figure 6.5 depicts the payoff that an 8-node job sees from the declaration of needing increasing numbers of nodes. As can be seen, the highest payoffs result from declaring the honest parameter (in this case, 8) to the system.

Figure 6.6 is a similar graph, this time exploring the payoff seen over many different declarations of the d parameter, the user's marginal dislike of waiting. The incentives exhibited in the unmodified EES in chapter 4 were always weak. As can be seen in Figure 4.2, the graph is dominated by a large regime of indifference. This characteristic is present to a greater degree in the relaxed EES. In all of our tests, the d parameter has had very little impact on user satisfaction, and in

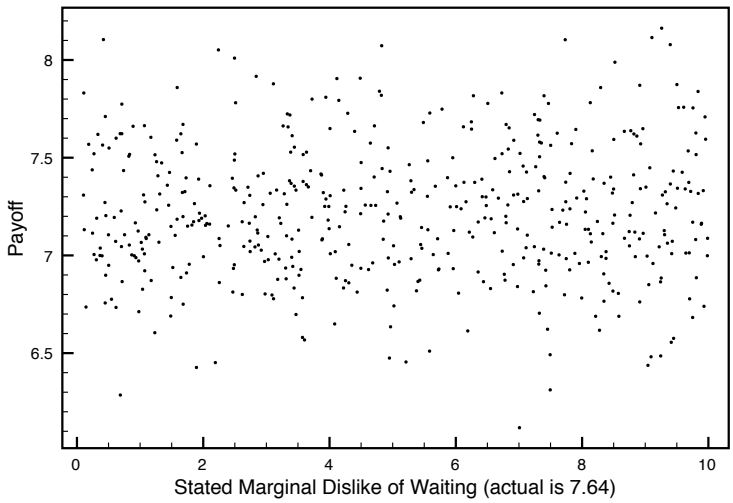

Figure 6.6: The expected payoff for a user over different stated marginal dislike of waiting values.

Figure 6.6 it is completely flat. This is not surprising, as each individual's d parameter has only a small effect on the aggregate dislike of waiting, which is what is used when making scheduling decisions. A parameter's complete lack of influence on the payoff seen by a user is referred to as being weakly truth-revealing. This means that while there are no strong forces incentivizing honesty, there is also no motivation to be dishonest.

6.2.3 Budget Balance

Figure 6.7 shows the relationship between budget imbalance (the degree to which the payments don't sum to zero), and the number of draws we use when

computing the expected values in the payment function. As is similar to original EES in Figure 4.5, increasing the thoroughness of the expected value computations greatly improves the budget balance characteristic. Because the graph rapidly approaches zero, the budget balance characteristic has survived the constraint-relaxation process fairly well.

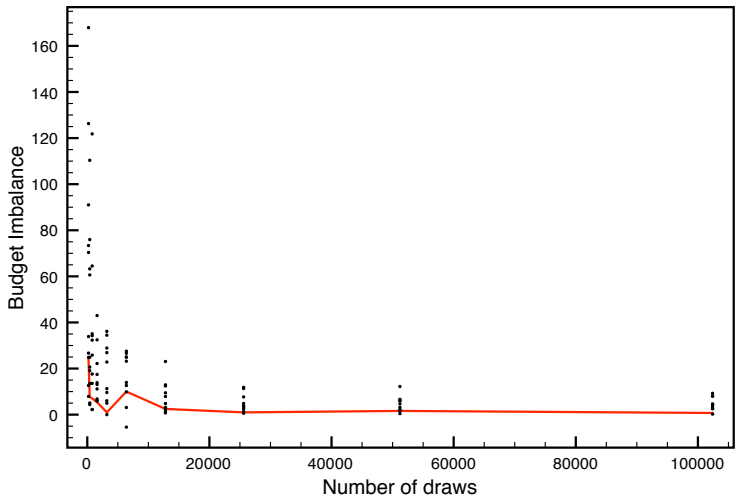

Figure 6.7: The budget imbalance, as we increase the number of draws used in computing the expected values.

Chapter 7

Relaxing Theoretical Constraints: Reservation Systems

As discussed in Chapter 3, both the Expected Externality Mechanism and the Generalized Vickrey Auction assume that the allocation function in use is completely optimal. In chapter 5, we presented the results of our application of the Generalized Vickrey Auction to allocation in Reservation Systems. In that chapter, we remained completely faithful to that requirement by restricting allowable jobs to only those that allowed us to tractably solve the allocation function κ^* optimally. Specifically, we mandated that all jobs submitted require exactly the number of nodes available on the resource. That restriction however, makes the scheme impractical for use on today's computational grids. Resource providers today expect a resource management scheme to multiplex jobs of many different sizes on to one set of resources.

As we did in chapter 6 for batch queues, in this chapter we examine what happens when we knowingly break the constraints of the GVA in order to deliver this increased functionality for Reservation Systems. Similar to chapter 6, we do this by replacing the optimal allocation function κ^* with a heuristic, κ', which provides results that are feasible but not optimal. For a diagram that graphically depicts the relationship of this chapter to the chapters that surround it, see Figure 1.2.

Whereas in chapter 6 we used a single-shot, best-effort scheduling function to estimate effects on turnaround time, in this chapter we use an iterative search function with an administrator-settable search depth. Like chapter 6, however, this search function provides no bounds on the quality of worst-case results, and as such is not an approximation algorithm.

7.1 Relaxed-Constraint DPGVA

7.1.1 What Users See

The relaxed-constraint DPGVA has similar semantics to the non-relaxed version presented in Chapter 5. In the relaxed-constraint DPGVA reservation system, users place bids for computational time. In these bids, users specify four values: the length of time needed (l), the deadline by which the job must be completed (d), the value of the work being performed (v), and the number of nodes needed

(n). Users are guaranteed one of two outcomes: either they will get no computational time and will be charged nothing, or they will get exactly l amount of time on n nodes by the deadline d and will pay at most v. These bids are sealed, and must be submitted before the scheduling period begins (*i.e.* if each day is scheduled at midnight, the bids must be submitted the previous day).

Aside from the ability to specify the number of nodes needed in the job, users interact with the system in an identical manner to the original DPGVA. For more information, see Section 5.1.1.

7.1.2 Schedule Determination

As discussed above, in this chapter we are replacing the optimal κ^* used in chapter 5 with a non-optimal $\kappa^{'}$ in order to tractably schedule jobs of non-uniform node count. We use a simple scheduling heuristic to implement $\kappa^{'}$, described as follows.

In Section 6.1.3, we introduced the SegmentedFunction data structure and described how we use it to greedily schedule jobs. The data structure allows us to keep track of a schedule of jobs, and to find new windows of availability and to insert new jobs in time scaling $O(n)$ as we increase the number of jobs in the structure.

For scheduling in the relaxed DPGVA, we repeatedly randomize the list of submitted jobs and greedily insert them into an instance of the SegmentedFunction data structure, limiting the schedule to our 24 hour scheduling time horizon and discarding all jobs that cannot be scheduled before their submitted deadline. As we do this, we keep track of the schedule that has the highest aggregate value (v) parameter. Each time we do this, we compare the aggregate value of the currently considered schedule (the schedule resulting from a greedy insertion of randomly ordered jobs) to the highest found thus far. If the considered schedule delivers higher aggregate value than the previously-found maximal schedule, we replace the maximal with the current. As a result, since each considered ordering of n jobs takes $O(n^2)$ time (n linear-time insertions), we would expect the search algorithm operating at search-depth m to scale $O(m * n^2)$.

As in Chapter 6, this κ' search heuristic provides no guarantees as to the quality of the schedules produced, and as such is not considered an approximation algorithm. We do not see this as a serious weakness, however, because being able to bound the accuracy of κ' would still violate assumptions made by the theoretical proofs of the mechanism.

7.1.3 Payment Computation

In the relaxed DPGVA, payments are computed in a manner virtually identical to the traditional DPGVA (and as presented in Section 5.1.3). Instead of the optimal dynamic programming method (called κ^*) used in Chapter 5, we are using the $\kappa^{'}$ heuristic presented in the previous section. This yields the following payment expression:

$$\mu_i(s_i, s_{-i}) = \sum_{j \neq i} U(s_j, \kappa^{'}(0, s_{-i})) - \sum_{j \neq i} U(s_j, \kappa^{'}(s_i, s_{-i}))$$

U is the codification of the user's preferences. As described above, we consider a user to get v value if his job runs before its deadline, d, and gets l amount of time to run on n nodes. So we can express U as:

$$U(s_i, \kappa^{'}(t_i, s_{-i})) = \begin{cases} v_i, & \kappa^{'}(t_i, s_{-i}) = runs \\ 0, & \kappa^{'}(t_i, s_{-i}) = discards \end{cases} \tag{7.1}$$

7.2 Results

In order to examine the extent to which these theoretical relaxations have affected the properties of the mechanism, we've developed a simulator. Since we

can know everything about simulated participants, and we can repeatedly examine the outcome of slightly different scenarios, we can examine to what degree the mechanism has retained desirable properties and we can measure the performance of the system.

The following experiments were conducted on a quad-processor Intel Xeon running at 3.2 ghz with 1GB of RAM. The operating system was Debian Linux version 1:3.3.5-13. The simulator was written in the Java programming language and was compiled and run with Sun Microsystem's java distribution version 1.4.2-03.

7.2.1 Incentives

As we've discussed, using the κ' heuristic invalidates the theoretical guarantees of proper honesty incentivization. In this subsection, we'll present data from our tests to determine to what degree these incentives are still present. In each of the incentives graphs, if the data peak near the actual value of the parameter, the mechanism is incentivizing honesty.

Figure 7.1 depicts the expected payoff seen by a user over a range of possible declarations of the v value parameter. The actual value of the job in question was 470. As you can see from the graph, there is roughly a regime of indifference between approximately 400 and 1000, and declarations lower than 400 tend to

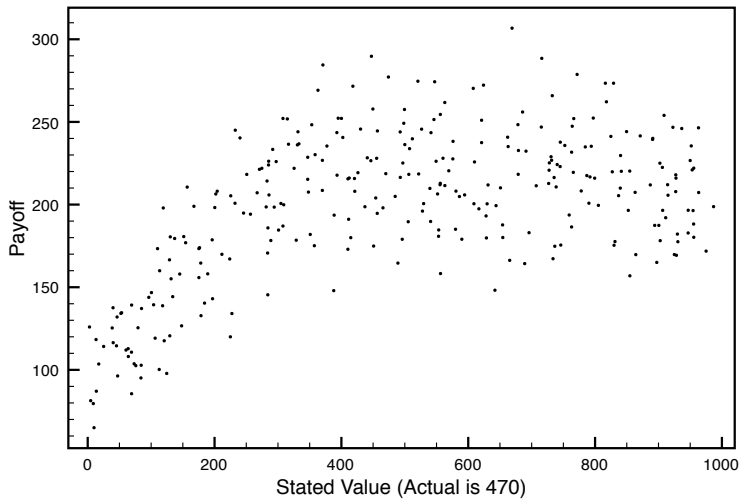

Figure 7.1: The expected payoff seen by a user, over many possible stated job values.

produce less desirable outcomes. This indicates that, while there is a fair amount of noise, there is no incentive for this participant to report his v parameter dishonestly.

Figure 7.2 depicts the same test, this time varying the the d deadline parameter. The actual deadline of the job in question was 82. Because we assume these deadlines are hard constraints and a user would see no value from a job that was completed after its deadline, we only explore declarations lower than 82. As you can see from the graph, declarations of deadlines lower than the actual value tend to produce less desirable outcomes for the participant in question. So, this shows

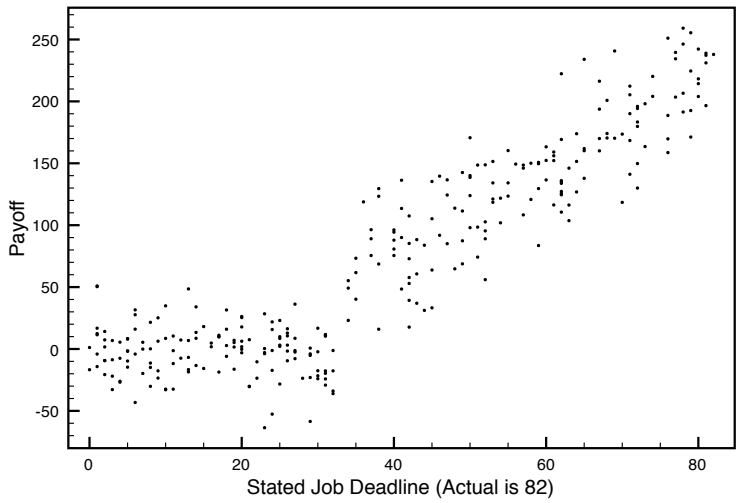

Figure 7.2: The expected payoff seen by a user, over many possible stated job deadlines.

us that despite our changes to the mechanism, we still see fairly good (although somewhat noisy) incentives for reporting the d parameter honestly.

Figure 7.3 depicts the same test, this time varying the the l job length parameter. The test needs to be redone, so this paragraph is placeholder until that time.

Figure 7.4 depicts the same test, this time varying the the n node count parameter. The actual number of nodes needed was 6. Because we assume that the node count is a hard constraints and a user would see no value from a job that was given fewer nodes than it requested, we only explore n declarations greater than 6. As you can see from the graph, declarations of n greater than the ac-

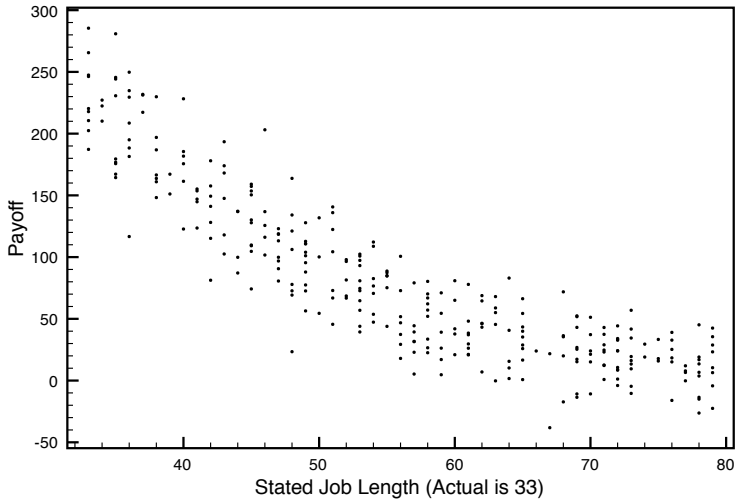

Figure 7.3: The expected payoff seen by a user, over many possible stated job lengths.

tual value tend to produce less desirable outcomes for the participant in question. This shows us that our changes to the mechanism to enable this n parameter have retained their truth-revealing properties for this new parameter.

7.2.2 Performance

In Section 7.1.2, we presented the search heuristic and our expectation that its exection time would scale $O(m * n^2)$, as we increase the search depth, m, and the number of jobs, n. Figure 7.5 presents how our scheduling algorithm scales as compared to the non-relaxed (optimal) DPGVA.

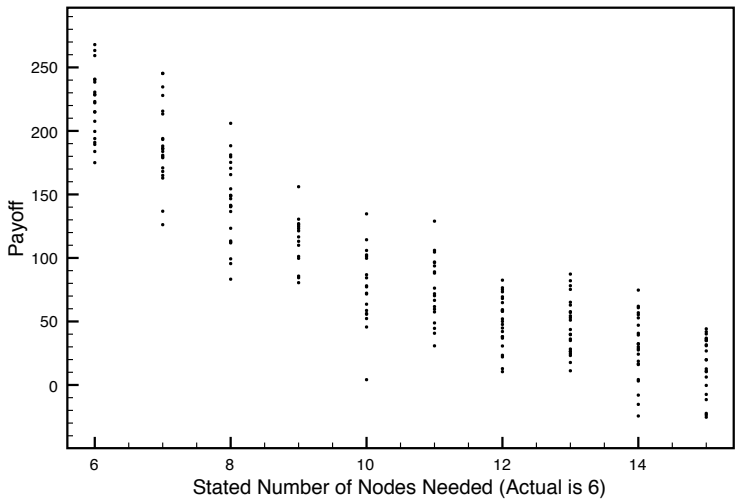

Figure 7.4: The expected payoff seen by a user, over many possible stated job nodes needed.

In Chapter 5, we discussed how the traditional DPGVA scheduling execution time scales $O(m^2)$ as we increase the number of jobs. From Section 7.1.2, we would expect the performance of the traditional and the relaxed systems to scale similarly. In practice, as can be seen in Figures 7.5 and 7.6, the execution time of the relaxed DPGVA seems to scale more slowly than the traditional.

The slower scaling in the relaxed system is likely due to the SegmentedFunction data structure completely filling its 24 hour time window, and as a result its linked-list stops growing. So, while the number of jobs that we are attempting to insert into the schedule keeps growing, the actual number of jobs in the data structure

stops growing, and the m number of linear-time accesses become m number of constant-time accesses.

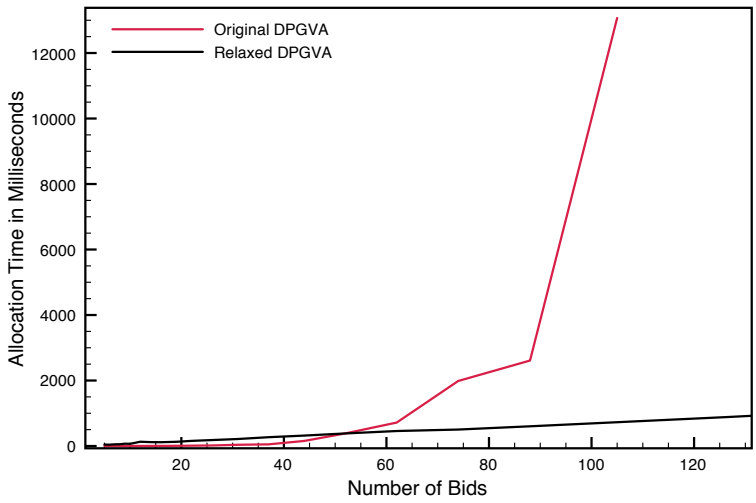

Figure 7.5: A performance comparison of the relaxed DPGVA to the original.

7.2.3 Effects of Search Depth

Search depth is a parameter in this system that can be set by the system administrators. The greater the search depth, the longer the heuristic searches for higher quality schedules.

Figure 7.6 depicts the effect that this parameter has on execution time. For search depths 100, 1000, and 10,000, the execution time increases slightly in excess of linear (as discussed in the previous section). At even search depth level 10,000,

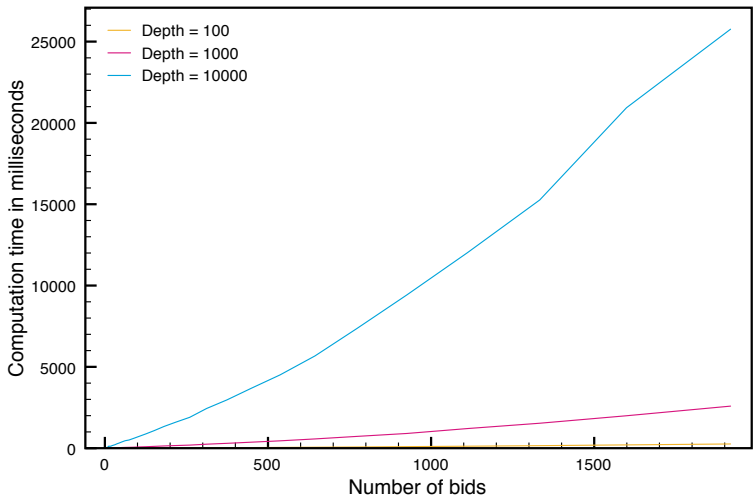

Figure 7.6: A depiction of how the search alogirthm's performance changes with respect to search depth

however, the system schedules 2000 bids in roughly 25 seconds, which is far better performance than the traditional DPGVA exhibited in Chapter 5.

Figure 7.7 depicts the same performance numbers, but includes a performance comparison to our brute-force solver. The brute-force solver operates in a similar manner to the heuristic, but rather than randomly searching for quality schedules, it exhaustively checks every possible ordering of jobs. As you can see in Figure 7.7, the execution time needed for the brute-force technique rapidly explodes, as the heuristics scale much more slowly (note that the y-axis is logarithmic).

The tradeoff for this improved performance, of course, is accuracy. Figures 7.8 and 7.9 show how the quality of the schedules produced degrades as we decrease

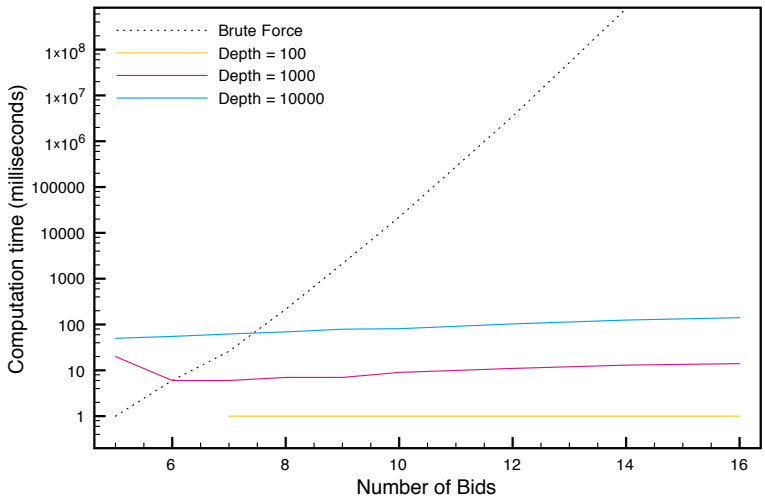

Figure 7.7: A depiction of how the search alogirthm's performance changes with respect to search depth, including the brute force data.

the search depth. As you can see in Figure 7.8, for small numbers of jobs, the quality of schedule produced is the same, regardless of search depth. As we increase the number of jobs, however, increased search time yields better schedules. It should be noted that for 14 jobs or fewer, a search depth of 10,000 produced the optimal schedule in all but one test (the 12 jobs test).

Comparison with the brute-force technique is only possible for a low number of jobs, as the brute-force solver scales very poorly ($O(n!)$). Figure 7.9 compares the different search depths to each other for larger values. Each location on the x-axis displays the amount of total wealth found by the heuristic for three different search depths: 100, 1000 and 10000. A search depth of 100 tends to find

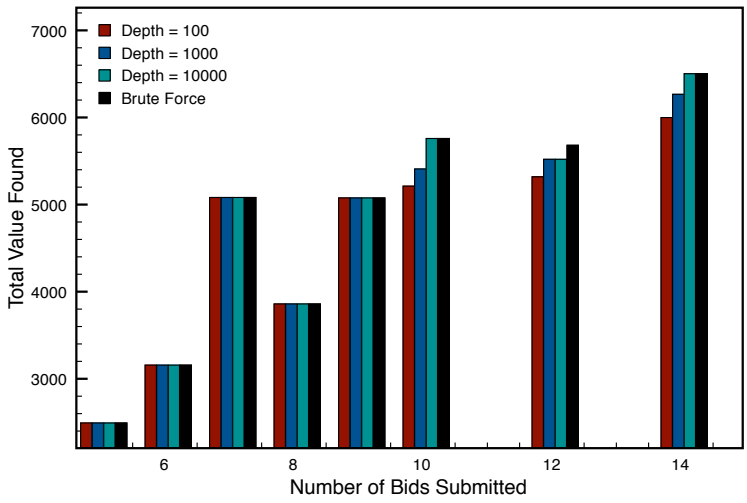

Figure 7.8: A depiction of how the search alogirthm's accuracy changes with respect to search depth, including data from a brute force search mechanism

respectable answers, but increasing the search depth to 1000 and 10000 does yield increased aggregate value in each case.

Increased schedule quality, of course, also matters to us in the effect it has on the truth-revealing incentives faced by the participants. Figure 7.10 depicts this relationship. As we increase the search depth (and as a result, the quality of the heuristic), the incentives become increasingly powerful, with less noise and a clearer maximum at value 470. Compare the blue ($depth = 10000$) dots to the yellow ($depth = 100$) dots. The blue dots are more tightly distributed in a regime of indifference surrounding the true value of 470, while the yellow dots seem to peak near 300, and provide the participant payoffs both positive and negative.

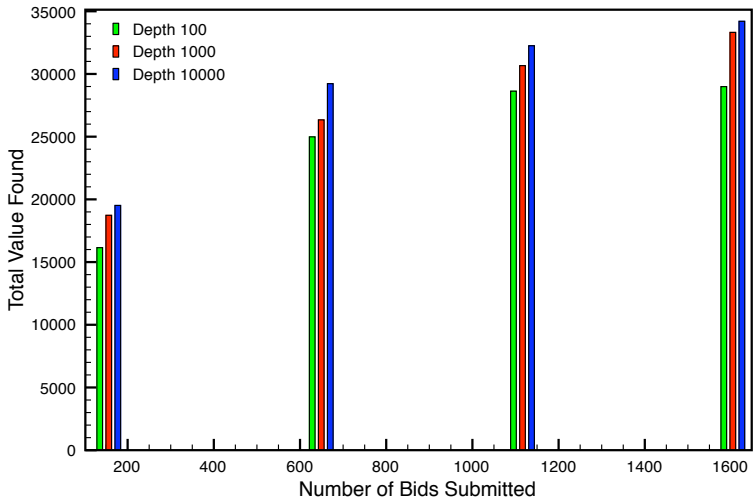

Figure 7.9: A depiction of how the search alogirthm's accuracy changes with respect to search depth

The reason for this can be seen in Figure 7.11. In this Figure, we display the likelihood that the user's job runs, over a range of different value declarations. For all search depths, increasing the declared value of your job increases the likelihood it runs. For low search depths, however, there is much more randomness in the outcome. The blue ($depth = 10000$) dots show a reliable jump from almost never being scheduled to almost always being scheduled at around $v = 100$. The yellow dots, on the other hand, exhibit a great deal of unpredictability, as even if you declare your job to have very low value it will still run roughly 30% of the time.

Figure 7.10: The effect that search depth has on the incentives faced by participants

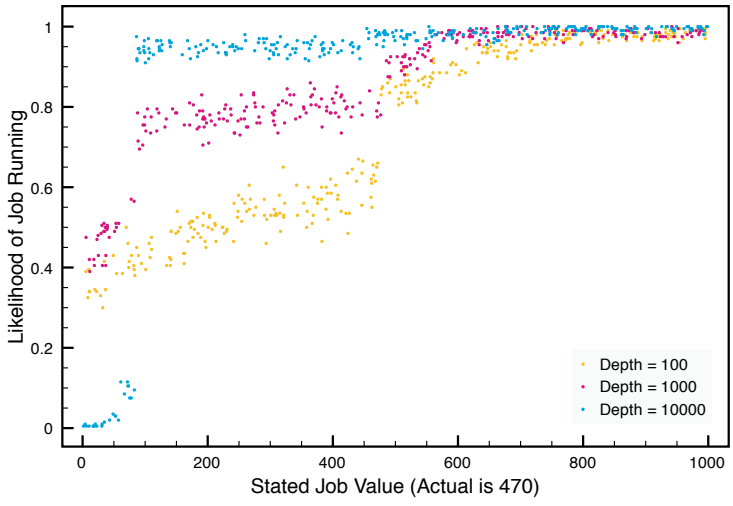

Figure 7.11: The effect that search depth has on the likelihood of a job running.

Chapter 8

Discussion

The preceeding four chapters have presented the results of four different areas of the design space that we have explored. In this chapter, we step back for a broad discussion of issues that span these chapters.

8.1 Nash-Optimality and Honesty

As discussed in Chapter 3, the Expected Externality Mechanism does properly incentivize truth-revelation, but only for a Nash-optimal solution concept. Nash-optimality is a weaker solution concept than the dominant strategy solution concept. While dominant truth-revelation would mean a participant was incentivized to be honest, regardless of the behavior of others, the Nash-optimal truth revelation in the Expected Externality Mechanism is predicated on other participants also honestly revealing their true valuations. If everyone is honest, then honestly is optimal.

Is this a realistic solution concept for our target environment? First, it should be noticed that this is self-reinforcing. If everyone in the system is behaving honestly, then it's in everyone's best interest to continue behaving honestly. Also, if we have a pool of n participants who are behaving honestly, then after introducing participant $n + 1$ it's still in everyone's best interest to behave honestly.

One approach to deploying the EES would be to build the user community slowly, starting with a small group of trusted individuals who can be expected not to game the system. Additional participants could be added one-at-a-time, while maintaining the truth-revealing equilibrium.

In the EES, it's important that the participants believe the properties of the system. Because the mechanism is only truth-revealing if everyone believes it's truth revealing (and acts accordingly), administrators must communicate this to the users in a convincing manner. From this perspective, it is advantageous that our targeted user community is composed of scientists. These users are likely to have faith in a game-theoretic result, and may have the ability to verify the math that underlies the mechanism if they so desire.

8.2 Distribution of Preferences

Our work in Chapters 4 and 6 with the Expected Externality Mechanism assumes that we have some knowledge of the distribution of participant preferences. The mechanism does not require that the preferences conform to any particular distribution, but does require that we know what the distribution is.

Determining this in practice would require harvesting large amounts of log data about users. A statistical model would need to be built from this data, predicting the likelihood of a job's length, value, urgency, and (for Chapter 6) the number of nodes needed. The correlation between these parameters would also need to be understood.

This task is not trivial, but is feasible given that the scheduling software would be at a central location and able to record this data. Additionally, the mechanism's truth-revealing properties would potentially produce log data that closely matched the actual parameters.

This problem is connected to the issues discussed in Section 8.1. If we initially deploy the mechanism using a semi-accurate preference distribution, and users believe the mechanism to be truth-revealing, then the information we gather about participant preferences will lead to a more-accurate preference distribution.

This self-reinforcing characteristic is common with nash-equilibria in general, and means that a pool of users and the mechanism that governs them can reach a point of dysfunction, where users lack faith in other users or the distribution of preferences used by the mechanism. To be clear about this self-reinforcing characteristic, it is not an *attractor*, in the sense that the system will not gravitate towards truth revelation if it is elsewhere, but if the system is at the nash-equilibrium and each participant expects the others to behave in a truth-revealing way, then that state is self-reinforcing. Techniques discussed in Section 8.1 can be used to rebuild dysfunctional communities.

One big advantage of the GVA mechanism presented in Chapters 5 and 7 is that these issues are completely avoided. Those mechanisms use a dominant strategy solution concept, and as such the strategies of each player do not depend on the strategies employed by his competitors.

8.3 Preference Distribution in Simulation

In Chapters 4 through 7, we've used simulation to test the properties of these mechanisms. In all of these tests, we've had to produce random participants with random preferences. Throughout these simulations, we've relied on a uniform

distribution to simulate these random preferences. Real participants would almost certainly have non-uniform preference distributions.

This assumption affects our results in a variety of ways. The results that are affected the least, of course, are the performance numbers presented in all sections. The parameter distributions would have a small effect on the size of some data structures, and as a result we can't say the performance is completely unrelated to this assumption, but the effect would be minimal.

The incentivization graphs, on the other hand, are affected to a greater degree. When computing the payments for the Expected Externality Mechanism, the distribution of participant preferences directly affects the payment calculations. As a result, our assumption of uniformness would be most likely to affect the incentivization graphs for our work in batch queues. Theory predicts that the incentives should hold for all distributions, so we don't believe that the mechanism would fail these tests if we had assumed different preference distributions. The incentives may have different characteristics, however (indifference regimes may be larger or smaller, for example).

The incentivization graphs for reservation systems would also be affected, but to a lesser degree. The preference distributions are not considered when computing the payments in the GVA, and as a result wouldn't affect an individual run very much. When testing the expected satisfaction of a declaration (the expected

difference between a user's satisfaction with the allocation and the amount he had to pay), the distribution of competitors' preferences will have an affect, however.

As a result, it's likely that our assumptions of uniform parameter distributions, while unrealistic, won't have a major impact on our results.

8.4 Collusion

It's important to note that neither the Expected Externality Mechanism, nor the Generalized Vickrey Auction prevent users for colluding to game the system. The truth-revealing results assume participants are acting alone.

As an example of how the Generalized Vickrey Auction can be susceptible to collusion, consider two particpants who both value the same outcome with maximal value. Since both of them value that outcome with very high value, when the mechanism compares the effect of each of them entering the system, it finds that effect is zero (both value it so highly, that it becomes the chosen outcome every time). As a result, both participants would pay zero and would get the desired outcome.

8.5 Outliers in Incentives Tests

In Chapter 6, we presented results indicating that users still face truth-revealing incentives, even after we've violated some of the mechanisms' constraints. These incentives were somewhat weaker, however, as the incentives graphs had a great deal of noisiness. In generally flat regions we described as "indifference," there would be occasional high-end outliers at non-honest declarations. Strictly speaking, any high payoffs at non-honest values don't bode well for truth-revealing incentives.

But in practice, we believe such values would not encourage dishonest behavior in users. In order to determine exactly where any outliers would be, a strategizing user would need to have accurate knowledge of the participants statistical parameter distributions. This information is gathered by administrators and used in the Expected Externality payment computation. If administrators keep this information private, this greatly reduces all users abilities to predict their competitors' actions. Additionally, these outliers are surrounded on each side by non-outlier values, so even if it were possible to identify points that our heuristics reliably overcompensated, it would need to be done with a degree of accuracy that the secret nature of the parameter distribution would prevent.

Chapter 9

Future Work

The work presented in this dissertation could be expanded upon in many ways, but there are two broad areas of greatest interest. The first is real-world testing involving actual users, and the second is more research to understand the relationship between κ' non-optimal allocation functions, and user preferences.

9.1 Real-World Testing

One significant difference between the Expected Externality Mechanism and the Generalized Vickrey Auction is the solution concept used. As discussed in Chapter 3, the Expected Externality Mechanism uses a *Nash-optimal* solution concept, meaning that truth-revelation is only an optimal strategy if it is also employed by the other participants. The Generalized Vickrey Auction, on the other hand, uses a *dominant strategy* solution concept, where truth-revelation is always the best strategy, regardless of the actions of others. These solution

concepts are predictors of real world behavior, but they give no guarantees that real people will act according to their predictions.

In the GVA-based systems in Chapters 5 and 7, will grid users act rationally and divulge job metadata accurately? Or will they mistakenly believe they need to underbid or overbid in some manner in order to get their fair share of resources? Will users in the EEM-based systems in Chapters 4 and 6 have faith that their competitors are divulging metadata honestly and act accordingly? Or will cynicism prevail, and users will be left without a clear optimal strategy? To what degree do the weaker incentives presented by Chapters 6 and 7 influence the behavior of real participants? Are they more likely to attempt to game the system?

The answers to these questions will ultimately affect the success of truth-revealing grid scheduling mechanisms. In order to answer them, tests with real users must be conducted. This is one natural and interesting area for future research.

9.2 Non-Optimal Allocation Functions and Incentives

Another natural direction to further this research would be to better understand the relationship between the κ' non-optimal allocation function and the resulting incentives faced by users. In this work, we've seen the effects that two different non-optimal allocation functions can have on incentives in Chapters 6 and 7. The non-optimal schemes we've tried are far from exhaustive, and it would be interesting experiment with other such functions to understand the effects they have on user incentives.

Do allocation functions with a greater degree of randomization make it harder for participants to strategize and game the system? Do some allocation functions have predictable quirks that produce exploitable allocations? Which allocation functions do the best job of incentivizing honesty, and at what computational cost do they come?

These questions would further the goal of delivering production quality, truth-revealing grid scheduling systems, and their answers would be found by further research.

Chapter 10

Conclusions

In the introduction, the following thesis question was asked:

> Can we design grid scheduling systems that reward users for being honest when submitting jobs, and can we do this while both retaining much of the semantics of existing schedulers and adhering to the computational tractability requirements of high-performance computing centers?

To conclude this dissertation, we would say the answer to this question is "Yes, but with caveats." We've presented four "sweet-spots" in this design space that deliver tractable, truth-revealing job scheduling while working within the traditional batch queue and reservation system frameworks. Our solutions in Chapters 4 and 5 tilted towards systems with stronger incentives while sacrificing feature-set, for batch queues and reservation systems, respectively. The solutions in Chapters 6 and 7 brought an expanded feature set (by eliminating the restriction that the number of nodes for each job and the number of nodes on the hardware

be the same) at the cost of the strength and clarity of the incentives presented to the user.

Ultimately, the strengths of the four schemes presented in this dissertation depend on the target user community. For a user community whose jobs were primarily of node-count equal to that of the hardware, or whose jobs were very flexible on node count, the schemes presented in Chatpers 4 and 5 are likely preferable to those presented in Chapters 6 and 7. Since these users would not benefit from the node-count relaxations, they would prefer to avoid the weaker incentives that accompany those solutions. On the other hand, user communities that had a great deal of heterogeneity with respect to job size would almost certainly prefer the solutions in Chapters 6 and 7, as dealing with weaker incentives is likely preferable to asking users to rewrite software to adhere to the fixed node-count of the local hardware.

Similarly, a user community's attitude towards best-effort batch queues and reservation systems would likely determine a preference between the chapters. User communities who have a great deal of hard deadline constraints, and who greatly value predictability as to when jobs run, would likely prefer the reservation-based solutions in Chapters 5 and 7. Chapters 4 and 6, on the other hand, would be best for communities who lacked these hard constraints, and whose

users preferred the more traditional and well-known semantics of best-effort batch queues.

Bibliography

[1] K. J. Arrow. *Economics and Human Welfare*, pages 23–39. Academic Press, 1979.

[2] A. Auyoung, B. Chun, A. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *Proceedings of the first workshop on operating system and architectural support for the on-demand IT infrastructure*, 2004.

[3] B. N. Chun, P. Buonadonna, A. AuYoung, C. Ng, A. C. S. David C. Parkes, Jeffrey Shneidman, and A. Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. 2nd IEEE Workshop on Embedded Networked Sensors, May 2005.

[4] B. N. Chun and D. E. Culler. Market-based proportional resource sharing for clusters. Technical Report CSD-1092, Computer Science Division, University of California at Berkeley, January 2000.

[5] B. N. Chun and D. E. Culler. User-centric performance analysis of market-based cluster batch schedulers. In *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid*, Berlin, Germany, May 2002.

[6] E. Clarke. Multipart pricing of public goods. In *Public Choice*, volume 2, pages 19–33, 1971.

[7] C. d'Aspremont and L.-A. Gerard-Varet. *Aggregation and Revelation of Preferences*, chapter On Bayesian Incentive Compatible Mechanisms, pages 269–288. North Holland Publishing Company, 1979.

[8] T. Groves. Incentives in teams. In *Econometrica*, volume 41, pages 617–631, 1973.

[9] V. Krishna and M. Perry. Efficient mechanism design. Working paper, 1998.

[10] J. F. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers*, 38(5):705–717, 1989.

[11] D. Lehmann. Truth revelation in approximately efficient combinatorial auctions. In *Proceedings of ACM Conference on Electronic Commerce*, pages 96–102, Denver, 1999.

[12] R. B. Myerson and M. A. Satterthwaite. Efficient mechanisms for bilateral trading. In *Journal of Economic Theory*, volume 29, pages 265–281. 1983.

[13] C. Ng, P. Buonadonna, B. N. Chun, A. C. Snoeren, and A. Vahdat. Addressing strategic behavior in a deployed microeconomic resource allocator. 3rd Workshop on the Economics of Peer to Peer Systems, August 2005.

[14] C. Ng, D. Parkes, and M. Seltzer. Virtual worlds: Fast and strategyproof auctions for dynamic resource allocation. Fourth ACM Conf. on Elec. Commerce (EC'03), 2003.

[15] D. C. Parkes. An iterative generalized vickrey auction: Strategy-proofness without complete revelation. In *Proc. AAAI Spring Symposium on Game Theoretic and Decision Theoretic Agents*, pages 78–87. AAAI Press, March 2001.

[16] O. Regev and N. Nisan. The popcorn market – an online market for computational resource. In *First International Conference On Information and Computation Economies (ICE98)*, Charleston, SC, USA, 1998.

[17] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Winner determination in combinatorial auction generalizations. In *Proceedings of Autonomous Agents and Multiagent Systems (AAMAS) 2002*, Bologna,Italy, July 2002.

[18] B. Schnizler, D. Neumann, D. Veit, and C. Weinhardt. Trading grid services - a multi-attribute combinatorial approach. In *European Journal of Operational Research*, volume 3, pages 943–961, 2008.

[19] S. Smale. Dynamics in general equilibrium theory. *The American Economic Review*, 66(2):288–294, May 1976.

[20] J. Stoesser and D. Neumann. Greedex: a scalable clearing mechanism for utility computing. In *Electronic Commerce Research*, volume 8, pages 235–253, 2008.

[21] I. Stoica, H. Abdel-Wahab, and A. Pothen. A microeconomic scheduler for parallel computers. In *Proceedings of the International Parallel Processing Symposium (IPPS) '95 Workshop on Job Scheduling Strategies for Parallel Processing*, Santa Barbara, CA, USA, 1994.

[22] I. E. Sutherland. A futures market in computer time. *Communications of the ACM*, 11(6):449–451, June 1968.

[23] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. In *Journal of Finance*, volume 16, pages 8–37, 1961.

[24] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[25] C. A. Waldspurger, T. Hogg, B. Huberman, J. O. Kephart, and S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, February 1992.

[26] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational grid. *International Journal of High Performance Computing Applications*, 15:258–281.

Appendices