Efficient Auction-Based Grid Reservations using Dynamic Programming

Andrew Mutz [#1], Rich Wolski [#2]

[#]*Department of Computer Science, University of California Santa Barbara*
Santa Barbara, CA 93106
U.S.A.
[1]amutz@cs.ucsb.edu
[2]rich@cs.ucsb.edu

***Abstract*— Auction mechanisms have been proposed as a means to efficiently and fairly schedule jobs in high-performance computing environments. The Generalized Vickrey Auction has long been known to produce efficient allocations while exposing users to truth-revealing incentives, but the algorithms used to compute its payments can be computationally intractable. In this paper we present a novel implementation of the Generalized Vickrey Auction that uses dynamic programming to schedule jobs and compute payments in pseudo-polynomial time. Additionally, we have built a version of the PBS scheduler that uses this algorithm to schedule jobs, and in this paper we present the results of our tests using this scheduler.**

I. Introduction

Current high performance computing centers rely primarily on batch queues to allocate resources. These centers use complex scheduling algorithms to multiplex multiple queues of varying priorities onto computing resources. Reservations are also used in these environments, although less frequently, and can involve direct contact with administrators. The goal of both of these types of allocation schemes is to balance notions of fairness, prioritization and maximization of system utilization.

In order to accomplish these goals, these mechanisms use information about the type of work being performed to make scheduling decisions. This information can be implicit, such as a job being submitted to batch queue of high priority, or explicit, such as stating the requested running time of a job. Implicit or explicit, this information affects scheduling decisions. Because schedulers are using this type of user-specified information in scheduling, it is important that this information is accurate. We have previously looked at getting accurate information from users in a batch queued environment [1]. In this paper we present a solution for eliciting honest information from users in a reservation-based environment.

Eliciting honest information from users is not simple or straightforward. If users are simply asked, for example, to gauge how important the job they are submitting is, many users will exaggerate the importance of their work in order to influence scheduling decisions in a manner that benefits them. Previous publications have described the extensive lengths to which users will go to increase their allocative satisfaction by distorting the preferences they report to an allocation mechanism [2].

In order to make sound scheduling decisions based on user-submitted information, one approach is to employ a pricing mechanism that is *incentive compatible*. Incentive compatible pricing mechanisms use game-theory to ensure that users' self-interests are maximized when they honestly reveal their preferences to the allocation mechanism. These mechanisms can provide powerful guarantees about optimal user behavior and can deliver highly efficient allocations, but come at a cost. A Generalized Vickrey Auction with combinatorial bidding, for example is very efficient and elicits honest information from participants, but computing the payments involved can require the solution of NP-Complete problems [3]. At the other end of the design spectrum, approximately-strategyproof mechanisms have been proposed, which provide computational tractability but are only able to provide approximate incentive compatibility (i.e. the gains seen by a user from distorting his preferences can be bounded) [4].

These mechanisms use combinatorial bidding which, while powerful, is perhaps more heavyweight a mechanism than is necessary for scheduling jobs on grid resources. It is possible to step back slightly from these powerful combinatorial bids in order to deliver much of the same desirable properties, but in a computationally tractable manner. In this paper, we present our exploration of this sweet-spot in the design space of incentive compatible grid scheduling mechanisms. We use this auction mechanism to deliver a tractable, incentive-compatible reservation system, which we have integrated with the Portable Batch System (PBS). We have named our allocation mechanism the Dynamic Programming Generalized Vickrey Auction mechanism or *DPGVA mechanism.*

A. Why is incentive compatibility important?

The DPGVA mechanism gives us guarantees that users are properly incentivized to honestly reveal their private information to the scheduling mechanism. These guarantees are useful in many ways.

First, as mentioned above, scheduling decisions that are made based on this information can be improved with more accurate information. Batch schedulers use information reported

by users to implement utilization maximizing algorithms (such as backfilling). If users are distorting the information they report to schedulers, these decisions can become less and less reliable. In short, if we cant trust the information that is being fed to the scheduler, then we cant trust the desirability of the schedules that are being produced.

Second, usability increases when users don't have to worry about gaming the system. Users who are presented with a system that can be gamed may feel like they *need* to distort their preferences in order to get their fair share of resources. For example, if we simply ask users how important the work they are performing is, otherwise altruistically-minded users may feel compelled to lie to the system, assuming that the rest of the participants were lying. In a more complex system, such strategization may require more complex thinking, thus wasting the time of all users.

Finally, having a reliable way to measure how much useful work is being accomplished on a resource can help administrators make planning and purchasing decisions. It may be the case that two different hardware installations both have a very high level of system utilization, but are being used for very different applications. System administrators planning future purchases can use aggregate "job value" throughput as another metric with which to plan. Without a reliable way to gather information about "job value", this is much harder.

II. Related Work

The observation that the problem of allocating time on computational grids shares much with the field of economics is not new, and much work has been published in this area. A variety of markets and auctions (two broad and overlapping classes of economic mechanisms) have been proposed to determine which jobs will have access to which grid resources, and for how long [5], [6], [4], [7], [8]. The advantage normally associated with using economic mechanisms is that by providing currency-based consequences for resource consumption, users are motivated to avoid inefficient resource usage. Usage that can be discouraged by such mechanisms includes, for example, low-priority jobs being run during periods of high-demand, or processes being run on hardware for which they are not well-suited.

Previous work has documented the gains in user satisfaction when job value is accounted for in scheduling decisions. In [9], Chun and Culler simulate different scheduling systems that account for job value when making decisions and find that aggregate user satisfaction can be increased by a factor of between 2 and 14 over value-agnostic schedulers. Additionally, in [10], Lee and Snavely show that users are capable of expressing complex preferences when asked. The authors surveyed users at the San Diego Supercomputer Center about expected running times and about the value of different turnaround times and found that while users generally have a poor ability to predict running time, users are able to express in great detail the value they see as a function of turnaround time.

As mentioned in the previous section, previous work by Ng, et al. [2] has shown that users will go to great lengths to increase their share of computational resources. In their tests of the Mirage system for allocating time on Sensor network testbeds, the found users gaming the system in a variety of ways: shading their bids to lower the amount they pay and cutting large jobs up into smaller jobs, for example. The Mirage system uses an iterative combinatorial auction that is not incentive compatible.

The application of pricing mechanisms to batch queues has been pursued. Previous work [1] by the authors of this paper has attempted to use the Expected Externality Payment mechanism to get honest information from users in a batch-queued environment. In [9], Chun *et al.* compare traditional, value-agnostic queuing systems to a scheme that reorders jobs in the queue based on bids. In [7], Stoica *et al.* use a first-price auction to determine which job in a *ready list* will run, and then use a combination of those bids and price prediction to determine how much participants pay.

Many scheduling systems for computational clusters eschew reservations and instead distribute resources in a spot market where jobs compete for access to computational resources. The Generalized Vickrey Auction is commonly used in such spot markets: both Schnizler et al in [4] and Das *et al.* in [11] use an approximated version of it for spot market distribution. Although they use different approximation techniques, approximation of the GVA payment computation in this fashion breaks the GVA's theoretical guarantees of *incentive compatibility*. A similar mechanism is pursued by Lai *et al.* in [6], although instead of the Generalized form they use non-combinatorial second-price auctions. The distribution of grid resources by commodities markets are explored for this application by Wolski *et al.* in [8], where both the *Tatonnement* and *Smale* price adjustment mechanisms are compared to auctions. An original auction mechanism is presented by Chun *et al.* in [5], where rather than bidding for complete control of a resource, jobs are allocated to resources in proportion to the quantites of their bids.

Some scheduling systems have focused, as this paper does, on pricing resource reservations. In [12], Bubendorfer *et al.* propose a reservation system that uses the Generalized Vickrey Auction to price resource reservations on computational grids. Computational tractability of the scheme is not discussed. In [3], Wellman et al discuss the strengths and weaknesses of different auction design choices as they affect pricing generic resource reservation slots, including in their analysis variants of Ascending Auctions and the Generalized Vickrey Auction. The onerous computational requirements of the GVA are noted, but no solution is proposed.

Our work, in contrast, focuses on delivering computational tractability without breaking the theoretical constraints that ensure *incentive compatability*. The scientific contribution of this paper is the dynamic programming algorithm that allows us to accomplish this in our target environment. We present this mechanism in detail and present the results of our evaluation of the mechanism in both simulation and on live hardware.

III. High-Level Description

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).

A. User Interaction

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 6pm on November 21st, 2008, and was willing to pay at most 500 credits for the time would submit:

```
$placebid 180 "2008.11.21 at 18:00:00" 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 of course 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.

B. Computing Payments

Each user that is allocated time on the resource pays some amount of virtual currency for that time. The payment mechanism we use is the well-known Generalized Vickrey Auction (GVA). The GVA has many good properties that make it desirable in our target environment. First, it is flexible enough to handle bids specifying the value of a range of allocative outcomes, rather than just one. This is required in our target environment, since users can be satisfied with a few different running times for their job. Second, it is a *dominant strategy* to be honest in your bidding in a GVA. This means that, regardless of the bidding strategies used by other participants, an individual advances his own interests the most when being truthful. This allows us to completely sidestep the challenges posed by using user inputs for scheduling decisions when those inputs don't honestly reflect the preferences of the users.

The GVA accomplishes this by computing a user's impact on the aggregate welfare of other users and charging him exactly the amount that his presence reduces others' allocative satisfaction. As a result, a user is motivated to report a bid that maximizes the sum of his allocative satisfaction and the negative value representing his impact on others' satisfaction. If the allocation function that reconciles these bids produces perfectly optimal schedules, the bid that maximizes this is an honest bid.

Despite these strengths, the GVA does have a weakness. In order to deliver these properties, the GVA requires that we can optimally solve the allocation problem at hand. Algorithms that produce approximately-optimal solutions are unacceptable, as the mathematics that make honesty a *dominant strategy* break in the face of approximations. The DPGVA system works around this issue by employing a pseudo-polynomial dynamic programming algorithm. This way, we can deliver computational tractability without resorting to approximation.

The specific manner in which the GVA payments are computed will be covered in detail in section IV.

IV. Mechanism Details

As mentioned above, the DPGVA reservation system uses the Generalized Vickrey Auction for determining how much each user pays for access to the resource. This auction scheme is well known for its powerful predictions about optimal user behavior. A common approach to this problem is to implement the GVA as a simple combinatorial auction scheme, which typically means that solving becomes computationally intractable. For example, this way would reduce a bid of the form (4hours, by noon, \$5) to multiple, mutually exclusive bids of $(1AM - 5AM for \$5)$, $(2AM - 6AM for \$5)$..., 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.

A. Optimally Allocating using Dynamic Programming

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 1, 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

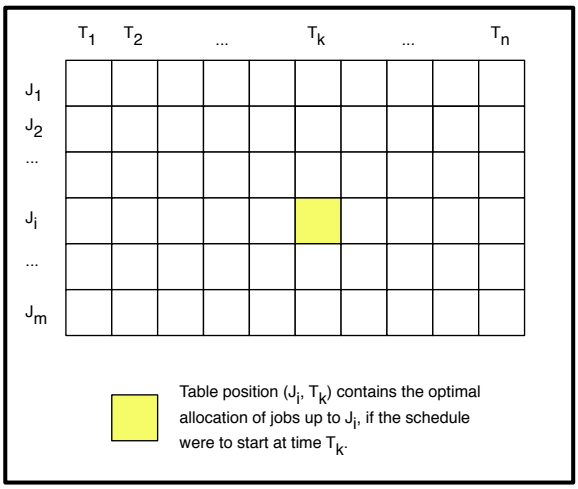

Fig. 1. The structure of the table that stores intermediate results.

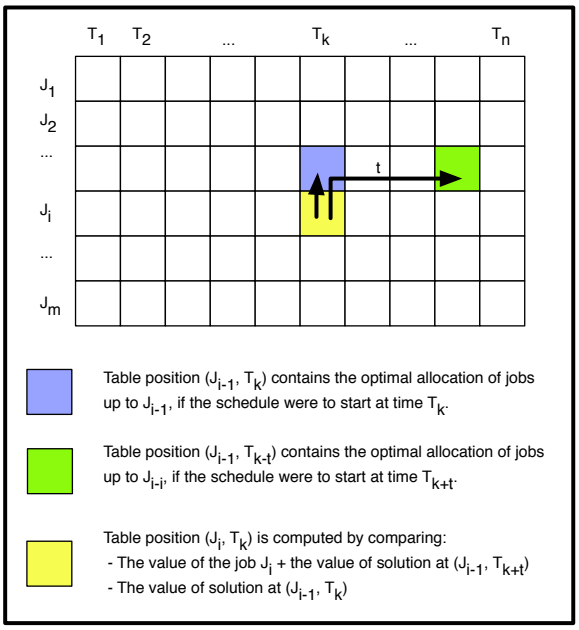

Fig. 2. The recursion relation. Each cell in the table is computed based on the results computed in other cells.

answer to the entire optimal scheduling problem will be held at (J_m, T_1).

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 2) is as follows:

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

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

V. Evaluation

In Section IV we the theoretical characteristics of the DPGVA mechanism. In this section we will present the results of our tests to experimentally verify these characteristics. These experiments take two forms: the interaction of simulated participants to verify the claims of *truth-revelation*, and load testing on live servers to verify the claims of computational tractability.

A. Truth Revelation

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.

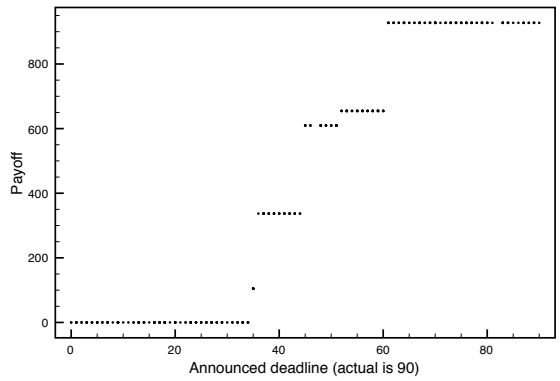

Fig. 3. The payoff seen by the simulated user over a range of possible deadline declarations.

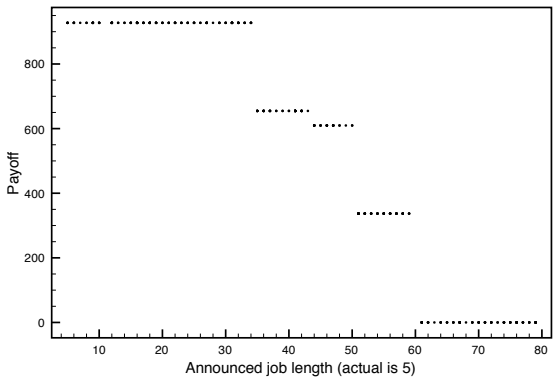

Fig. 4. The payoff seen by the simulated user over a range of possible job length declarations.

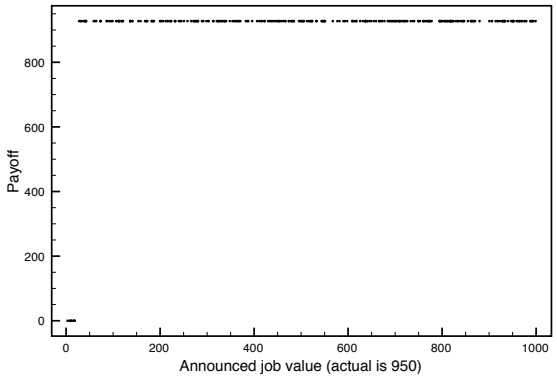

Fig. 5. The payoff seen by the simulated user over a range of possible job valus declarations.

Figures 3, 4 and 5 show the results of one such simulation. In these tests, we've created a large 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 3 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 value other than 90 leads to the same or lower payoff.

Figure 4 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 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.

These results indicate to us that the DPGVA mechanism exhibits the strong truth incentivization properties that theory predicts it should.

B. Performance

In Section IV, theory was used to predict good scaling properties of the DPGVA mechanism. 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 Pentium III (Xeon) running at 2.2 GHz, with 512MB of RAM, running Debian Linux. 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

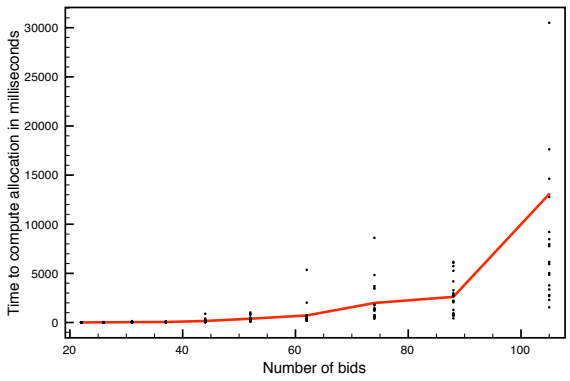

Fig. 6. The amount of time required to find the allocation of jobs, as we vary the number of bids submitted.

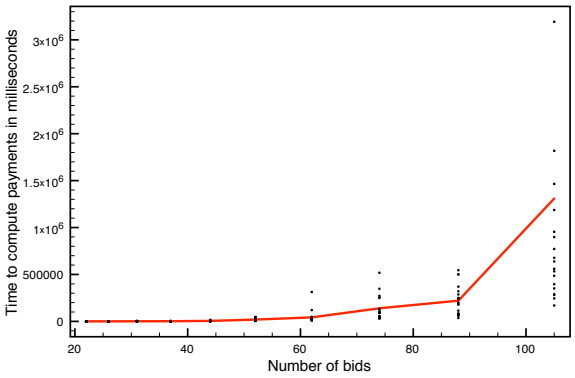

Fig. 7. The amount of time required to find the payments of each user, as we vary the number of bids submitted.

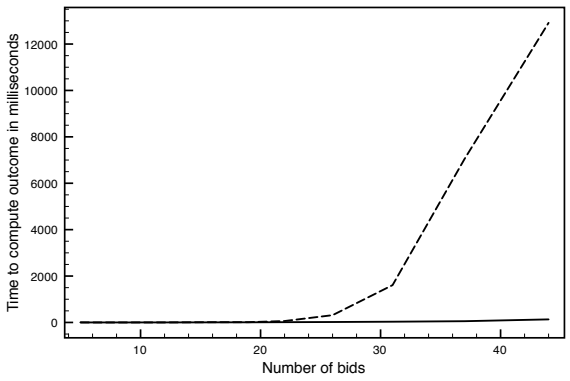

Fig. 8. Comparing the performance of a brute-force GVA implementation to the DPGVA algorithm, while computing the outcome. The dashed line is the brute force algorithm, while the solid line is the DPGVA results.

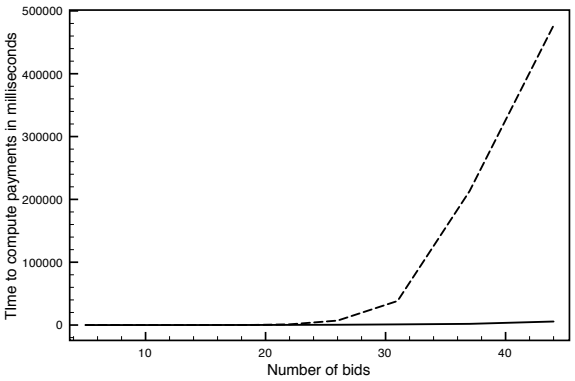

Fig. 9. Comparing the performance of a brute-force GVA implementation to the DPGVA algorithm, while computing the payments. The dashed line is the brute force algorithm, while the solid line is the DPGVA results.

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 6, 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 7, the performance measurements for this computation 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 Figures 8 and 9, the amount of time needed for an exhaustive brute-force 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 theory presented in Section IV 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 10 and 11, we see this linear increase in practice. In these tests, we schedule 20 jobs using a maximum specifiable

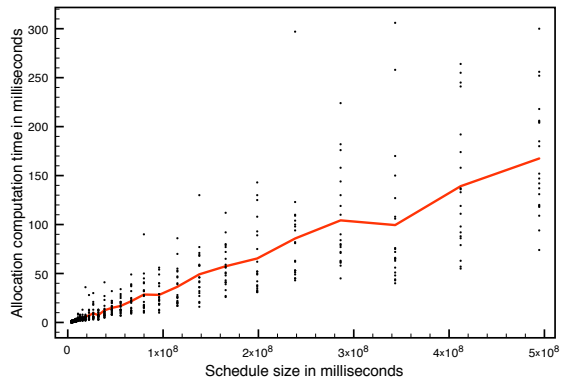

Fig. 10. The amount of time required to find the allocation of jobs, as we vary the size of the schedule.

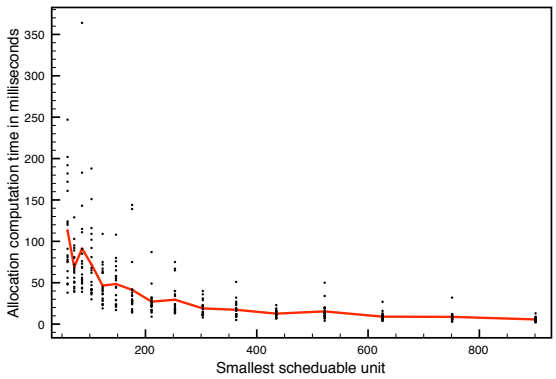

Fig. 12. The amount of time required to find the allocation of jobs, as we vary the size of the smallest scheduable unit.

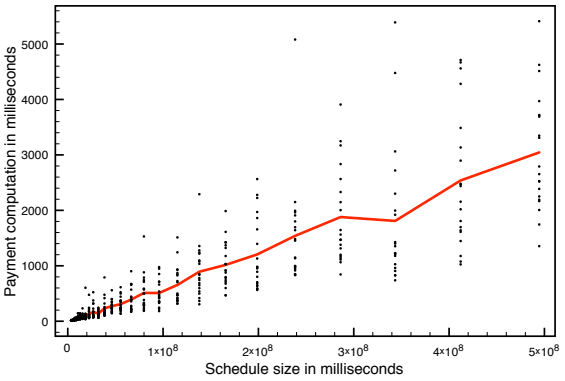

Fig. 11. The amount of time required to find the payments of each user, as we vary the size of the schedule.

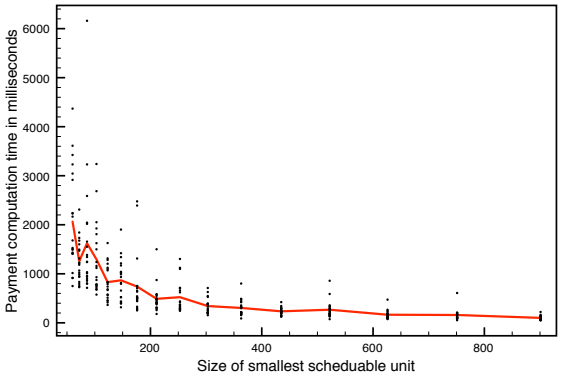

Fig. 13. The amount of time required to find the payments of each user, as we vary the size of the smallest scheduable unit.

time granularity of 15 minutes. Figure 10 presents the running time measured to schedule the jobs as we increase the time period over which we are scheduling. Figure 11 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. Figures 12 and 13 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 10 and 11. 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.

VI. Conclusions and Future Work

According to our examinations, the DPGVA mechanism delivers on its design goal: providing a computationally tractable and incentive compatible pricing mechanism for reservations on a computational grid. We have demonstrated through simulation that the powerful truth revealing guarantees provided by the GVA have been retained in practice. We have demonstrated through live performance measurements that our dynamic programming algorithm exhibits an acceptable degree of computational tractability. The DPGVA algorithm has exhibited, in our tests, a significant performance improvement over our brute-force GVA solver. Our DPGVA mechanism is certainly not perfect, however. The constraint that all submitted jobs must be of the same ”width” (i.e. number of nodes), is one that prohibits its use for many target environments.

We intend to focus on this deficiency in future work. One solution would be to identify another pseudo-polynomial algorithm that can solve the problem tractably without the same-size constraint. This solution would be ideal, but difficult to achieve. We have invested research into extending our current algorithm in this manner, but have found it a very hard problem. Searching previously published research on the

solution of related problems has proved similarly fruitless. We intend to address this shortcoming in future work, however, by attempting to measure the degree to which incentive compatibility breaks when we replace our optimal, pseudo-polynomial algorithm with one that is approximately optimal.

REFERENCES

[1] A. Mutz, R. Wolski, and J. Brevik, "Eliciting honest value information in a batch-queue environment," in *Proceedings of 8th IEEE/ACM International Conference on Grid Computing*, 2007.

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

[3] M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason, "Auction protocols for decentralized scheduling," *Games and Economic Behavior*, vol. 35, no. 1-2, pp. 271–303, 2001. [Online]. Available: http://citeseer.ist.psu.edu/383290.html

[4] B. Schnizler, D. Neumann, D. Veit, and C. Weinhardt, "A multiattribute combinatorial exchange for trading grid resources," in *Proceedings of the Research Symposium on Emerging Electronic*, 2005.

[5] B. N. Chun and D. E. Culler, "Market-based proportional resource sharing for clusters," Computer Science Division, University of California at Berkeley, Tech. Rep. CSD-1092, January 2000. [Online]. Available: http://citeseer.ist.psu.edu/chun99marketbased.html

[6] K. Lai, B. A. Huberman, and L. Fine, "Tycoon: A distributed market-based resource allocation system," 2004.

[7] 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. [Online]. Available: http://citeseer.ist.psu.edu/stoica94microeconomic.html

[8] 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*, vol. 15, pp. 258–281. [Online]. Available: http://citeseer.ist.psu.edu/wolski00analyzing.html

[9] 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. [Online]. Available: http://berkeley.intel-research.net/bnc/papers/ccgrid02.pdf

[10] C. B. Lee and A. Snavely, "On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions," vol. 20, pp. 495–506.

[11] A. Das and D. Grosu, "Combinatorial auction-based protocols for resource allocation in grids," in *Proceedings. 19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[12] K. Bubendorfer, K. Chard, P. Komisarczuk, and A. Desai, "Fine grained resource reservation and management in grid economies," in *Proceedings of The 2005 International Conference on Grid Computing and Applications*, 2005.