Effects of Low-Quality Computation Time Estimates in Policed Schedulers

Justin M. Wozniak
University of Notre Dame

Yingxin Jiang
University of Notre Dame

Aaron Striegel
University of Notre Dame

Abstract

Researchers conducting computer simulations can often provide estimates of computation time for a given type of simulation, which may be used by the compute cluster to aid in resource allocation and scheduling. However, the low quality of these estimates can cause deadline misses and unpredictable behavior. This problem is exacerbated on complex compute resources, clusters, and grids. Past-deadline jobs may be killed to provide resources for others, but the effect on the throughput of whole batches not fully understood. In this paper, we examine models for simple job schedulers and examine the quality of the deadline guarantee given. Simulation results based on actual runtimes are provided and discussed.

1 Introduction

The emergence of commodity compute clusters and grids has provided researchers with an important tool for simulation. Batch systems such as PBS [12] and LSF [21] provide users with a cluster of compute hosts to which jobs may be submitted, while grid engines such as Condor-G [10] and Globus GRAM [8] create an access point for widely distributed compute systems. Although such resources often suffer from poor internal communication speed compared to a multiprocessor shared memory machine, they have been used with considerable success by researchers who need to submit a batch of independent jobs for processing. The vast majority of this work has focused on how to successfully share and complete computing tasks such as computation and storage to achieve a large scientific objective.

However, an often unaddressed aspect of grid computing is the notion of deadline-driven scheduling [18]. Unlike traditional deadline scheduling from the realm of real-time computing [15], the problem of deadline scheduling in the grid context is significantly more difficult. The critical difference that emerges in grid computing is the dynamic nature of the grid resources themselves.

Previous work in the area of grid deadline scheduling considered only the impact of low quality estimations on throughput with regards to deadline scheduling [18]. To the best of our knowledge, no previous work has examined the effects of *policing* when coupled with variable quality deadline estimations in the context of grid computing. In addition, previous work in grid scheduling has focused on each task being completely independent. For many types of scientific simulation such as parameter sweeps and parameter explorations, this is often not the case. In those cases, tasks can often be grouped into a batch of related tasks that while computationally independent, the utility of the final scientific result is dependent upon the completion of all tasks.

Thus, the motivation of our paper is to answer the following question: *Given an environment of related tasks with low quality information, how strictly should policing be enforced and what effects will result on throughput and deadline guarantees?* In our paper, we make several key contributions. First, we offer insight into the effects of policing when the system contains low quality run-time estimations. We offer a broad set of simulation studies incorporating both real world data as well as synthetic data. Second, we formalize a model whereby batches of jobs can be specified with differing deadlines, and applied this to real world scenarios. Finally, we offer insight into how to police. In short, when multiple users offer bad estimates, who should pay the price?

The remainder of our paper is organized as follows. Section 2 describes related work both in the grid computing community as well as related work from multi-processor real-time systems. Next, Section 3 motivates our work by providing cases that exemplify the tradeoffs directly considered in this work. Then, Section 4 formalizes the system model used in Section 5 for our simulation studies based on both synthetic and real world data. Finally, Section 6 offers several concluding remarks.

2 Related Work

This work is intended to explore the area between traditional real-time computing and grid computing. These two

research areas traditionally have very different objectives.

In a real-time system, jobs are marked with a worst-case computation time that is enforced by system policy. Any deviation over this worst-case estimate is expected to result in termination. The result is that real-time schedulers must be conservative when scheduling. An investigation into how conservative such schedulers must be when computing on the grid may be found in [18].

The real-time literature is rich with examples of schedulers for single and multi-processor systems. In uniprocessor systems, EDF (Earliest Deadline First) is optimal [14]. In the multiprocessor case, however, the worst-case achievable utilization on M processors for EDF, which is a job-level dynamic-priority algorithm, is only $(M+1)/2$ [6]. In [15], the authors showed that if all tasks have utilization limitations under a value α, the utilization bound for the system increases to $(\beta M+1)/(\beta+1)$ where $\beta = \lfloor 1/\alpha \rfloor$. Anderson et al. [3] proposed a EDF-based scheduler which ensures the bounded deadline tardiness for every task in the system. In [3], however, even though there is no restriction on the overall system utilization, the per-task utilization for any task must be capped: the lower the cap, the lower the tardiness threshold.

Efficient scheduling of a batch of independent tasks is a challenging problem in grid no matter what the performance metric is. Several scheduling strategies have been proposed to achieve the minimum overall running time [11, 13, 16], which do not take into account possible deadline requirements from users.

Simulation of computational grid resources is an important tool when testing scheduling algorithms and techniques. The Bricks system [19] has been used by several researchers to test algorithms and grid performance. For example, to effectively schedule parameter sweep jobs and associated file staging on an internetwork of compute hosts, Bricks was used [7] to evaluate various scheduling algorithms.

Real software aids actual users who desire efficient scheduling on the grid. The AppLeS system [4] and the AppLeS Parameter Sweep Template (APST) match jobs to a variety of grid resources. The Nimrod parameter sweep system also provides deadline scheduling with respect to resource costs [2].

3 Case Studies

3.1 Applications: SimpleScalar and NS-2

For our first example case, a CPU design was optimized through simulation with SimpleScalar [5], a tool for simulating the performance of real programs on a range of modern processors utilizing execution-driven simulation. Tasks varied in run-time from as shown in Table 1. Most importantly, the utility of the results themselves was dependent upon the completeness in that no points were missing from the results. In the second simulation case, network simulations were conducted using the NS-2 [1] simulator. NS-2 is a discrete event simulator that provides packet-level granularity for simulating networks. During execution of the above simulations, the run-time statistics were sampled. *No reasonable guesses about when the batches would actually complete could be made.*

3.2 Grid Middleware: GIPSE

Over the past few years, the GIPSE (Grid Interface for Parameter Sweeps and Exploration) toolkit has been developed [20]. Rather than exposing the task-centric nature of the grid, the tool allows the user to solve problems using a research-oriented interface. GIPSE manages the creation and monitoring of jobs on the compute grid, as well as maintaining a database of all the relevant data about previously completed jobs, including input parameters and output results, job metadata such as the executable and resources used, job computation time, and other useful information.

It is this requirement for timeliness the large body of work in real-time multi-processor scheduling can be brought to bear. In essence, the problem can be reduced to an admission control and scheduling problem. For the researcher, the question is simple: given a group of tasks, can they be finished on time? However, it is the nature of grid computing that makes this problem significantly more difficult than the traditional multi-processor scheduling problems faced in real-time.

4 A Model for Deadline-Driven Grids

As discussed above, there is a great need for *ad hoc* compute clusters and grids that can provide reasonable schedule guarantees. In this section, we lay out a model to meet such demands, as shown in Figure 1.

The object of the system is to complete several batches of jobs, where each batch has a deadline given by the user. For each batch $B_i \in B$, $B_i = \{J_{ijk}\}$, where each job J has three indices, the batch number, the task type, and a uniqifier. Each batch has an independent deadline, $B_i.deadline$. The system can identify a job J_{ijk} as an instance of task $T_j \in T$, belonging to batch B_i, and different from all other jobs in the system.

To obtain a response from the scheduler as to whether B_i is feasible given the current system state, each job is given a computation time estimate, $J_{ijk}.est$. The user also provides each job with a corresponding input set I_{ijk}, which is a set of input parameters valid for task T_j, which is the parameter

space for T_j, denoted $T_j.I$. Each parameter is indexed, so $I_{ijk}[0], I_{ijk}[1], ...I_{ijk}[m-1] \in I_{ijk}$. The actual computation time is obtained empirically by the grid; in this model, each task has a *device* that may be applied to an input set to determine computation time. Hence, once submitted to a compute resource, the computation time $J_{ijk}.c$ is obtained by evaluating $J_{ijk}.c = T_j.device(I_{ijk})$.

A compute grid G consists of N homogeneous processing hosts, that are each capable of executing any job in a non-preemptive, single-processing manner. The jobs currently running on the grid at time t are said to be in the set R_t. If a job J is added to R_t, then $J.start = t$. If more jobs are submitted to the grid than available hosts, they are queued in the FIFO wait queue W. When a job J_i completes execution, $R_{t+1} := R_t - \{J_i\}$, an event is triggered back to the user, and if W is not empty, it is popped to obtain job J_j, which is added to R_{t+1}.

The scheduler S accepts a batch B_i from the user at any time, and can determine whether the batch has a feasible deadline in several heuristic ways. In the FIFO model, which we use[1], S builds up a calendar into the future, stacking onto R all jobs in $W \cup \{B_i\}$. If each job meets the deadline, success is returned for the batch, but since the guarantee is based upon the accuracy of the task estimates provided, there is the possibility of deadline misses.

The value of the batch in a simulation research setting represents, for example, data points on a plot. This work assumes the researcher cannot accept partially complete datasets. Since the user penalty for having a rejected batch is less than that from a late or incomplete batch, it is better to force the user to negotiate: to adjust their submission to meet a deadline than to provide a less accurate guarantee, providing poor results.

This is based on the assumption that the user can supply per task estimates where the actual runtime is centered around the estimated computation time, but offset by a random value that is within a given percentage of the actual time. We call this value the Quality of Estimate (QoE) to avoid conflict with [9]. For example, if the simulated user can always provide the scheduler with a time estimate such that the actual runtime is within 40% of the estimate, and not biased higher or lower, we say the QoE is 40%. So we assert:

$$\left|\frac{J_{ijk}.est - J_{ijk}.c}{J_{ijk}.est}\right| \leq QoE. \quad (1)$$

For any set of batches B, after submission to the scheduler, we have an acceptance ratio that indicates the size of the subset of B that was accepted. Additionally, we use the definition of guarantee ratio as given in [17], on batches, so the ratio represents the number of batches in which all jobs met the deadline for that batch, divided by the number of batches accepted by the scheduler.

To help free up space for a new batch that arrives at a given scheduling event, the scheduler may be permitted to kill jobs that have been running in R for longer than they were allocated based on their estimate. So we say that J is eligible to be killed at time t if:

$$t - J.start > J.est. \quad (2)$$

To demonstrate an intermediate approach as motivated in Section 5, we can instruct the scheduler to kill jobs that have exceeded a certain threshold K, meaning we kill jobs when:

$$t - J.start > J.est \times (1 + K). \quad (3)$$

For example, if $K = 0.5$, jobs are allowed to exceed their estimate by 50% before becoming a possible victim.

Choosing K is difficult, because it is not easily possible to choose a value that provides the users with flexibility for errors and still promotes fairness and deadline guarantees. To help select jobs for termination, and ameliorate these issues, we may assign a probability p that a job will be killed at a scheduling event at time t, as:

$$p = 1 - \frac{J.est}{t - J.start}. \quad (4)$$

For example, a job that had exceeded its estimate by a factor of 2 at a given scheduling event would have a 50% chance of being killed.

[1]For simplicity and compatibility with our experimental results on a Condor system.

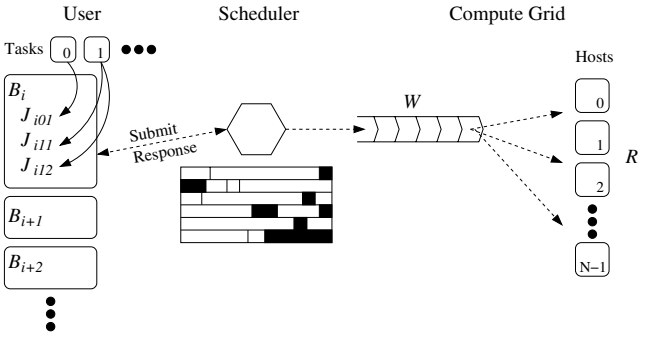

Figure 1. Deadline-Driven Computing Model

5 Simulation

To investigate the properties of the system presented in Section 4, a simulator was written that corresponds to that model. The results in this section demonstrate the performance of scheduler policy under idealized and experimental workloads.

The simulator, designed to allow the study of computation time *estimates* over *time* is named East[2]. The program allows the researcher to set up a virtual compute grid and scheduler combination. Virtual tasks may be defined, assigned input parameters, grouped into batches, and sent to a virtual compute cluster. A table of computation time devices is given in Table 1. The experimental run times were obtained from the Condor system as discussed in Section 3.

Each of the following tests is based on a varying QoE, as defined above in (1). For each QoE value, 10 tests were run, and the results were averaged. The standard deviation of the tests is shown by the bars around each data point. For each job, the estimate given to the scheduler was randomly selected from the set of estimates that satisfy (1). Batches of fixed size arrived at the scheduler between random, uniformly distributed intervals around a given mean value. The input set for each job was randomly, uniformly selected from the space of valid input for the appropriate device. The simulated compute resources comprise a simple N node cluster. The batch size was scaled up as the number of hosts increased to simulate more complex systems.

5.1 Low-Quality Estimates

In this test, the acceptance and guarantee ratios for the scheduler were measured agaist the QoE. As shown in Figure 2, increasing the error in the estimate has a significant effect on scheduler acceptance, especially in complex systems. This was observed in many cases to be due to a single large over-estimation of the runtime of a long job, which forces the schedule past the deadline, and results in rejection. Additionally, the guarantee ratio for whole batches is perfect when the estimates are exact, but even when they vary widely, the effect is that the guarantee ratio rarely drops. This is because the deadlines are somewhat permissive, but tight enough that not all batches are accepted. However, in the complex case with 128 hosts, a heavy job rate, and very bad estimates, almost no jobs are accepted, because nearly all batches will have at least one extremely long job with an overestimated runtime, forcing batch rejection. In these extreme cases, the Batch Guarantee Ratio is shown as 0 because no batches were considered.

Overall, while the guarantee ratio is nearly perfect, the acceptance ratio is not. This indicates that users could intentionally provide underestimates to increase their acceptance ratio, hogging the system.

In our next test, Figure 3, we attempt to police the system fairly and increase the scheduler acceptance rate by killing jobs that have exceeded their estimate. However, when the scheduler is instructed to kill jobs that have exceeded their estimate in order to accept future jobs, many fewer batches are able to complete. This is aggravated by the fact that we are measuring the Batch Guarantee Ratio, so killing one job that is slightly over time results in a loss of the whole batch. In addition, there is little to no gain in scheduler acceptance. This is because many of the jobs that were killed were close to completion at the time of kill, which means that very little schedule time was freed.

Similar results are obtained when using runtimes from real NS-2 runs as shown in Figures 4 and 5, in which we repeat the above experiments using experimentally observed NS-2 runtimes.

5.2 Grace Periods

The fact that so many jobs are killed near their completion time motivates the implementation of grace periods. The simple technique of granting jobs a threshold, or grace period, before killing them was also simulated for the Poly-Device tasks. For a range of values of K, the same simulation was performed, where jobs were killed according to the method described above. Although the results in Figure 6 show that larger grace periods result in more throughput as measured by the batch guarantee ratio, this is an unfair, easily manipulated system policy.

5.3 Probabilistic Enforcement

Applying the probabilistic policy, as describe in Equation (4) and shown in Figure 7, acceptance ratios match up with ratios as given in the previously shown hard enforcement and non-enforcement policies. The batch guarantee ratio is much improved. This is due to the policy, which rarely kills jobs that are near their estimate. However, throughput as measured by the batch guarantee ratio still does not approach that of the unpoliced system.

5.4 Protecting Users from Bad Estimates

The intent of a policed system is to provide better results for users overall, especially those that use the system properly. Users that intentionally abuse the system, for example, by providing low estimates to increase the probability of batch acceptance, should receive poor results, if this is necessary to continue to provide good results to other users.

In this experiment, we compare the results obtained by two groups of users: Group A, which submits NS-Device tasks with estimates centered on the correct value, and Group B, which always submits an underestimate. Estimates for Group B were generated by randomly selecting an estimate inside the given QoE range, until an underestimate was obtained. Acceptance and guarantee ratios are shown in Figure 8. These results show that Group A users obtain better results: more of their accepted batches complete on

[2]This work was supported in part by the National Science Foundation through the grant NSF DBI-0450067.

Type	Name	Device	Median	Mean	Min	Max
Ideal	SumDevice	$\sum I[i]$	22	22	0	45
Ideal	PolyDevice	$\sum k_i I[i]$	123	198	0	804
Experimental	SS-Device	Lookup I in table `SS`	218	237	137	615
Experimental	NS-Device	Lookup I in table `NS`	297	426	87	3442

Table 1. Computation Time Devices Used in the East Simulator

time, because very few of their jobs are killed, compared to the Group B users.

6 Conclusion

While real-time computing and grid computing both emphasize scheduling, their ultimate goals are often different and result in tradeoffs. Strictly enforcing computation time estimates on the grid can greatly reduce throughput in heavily loaded, complex systems. However, failure to enforce a schedule is unfair to the other users of a shared compute system, and necessitates policy to prevent users from cheating the system by providing misleading estimates.

Tests performed with a new grid scheduling simulator showed that killing jobs that exceed their estimates can greatly reduce throughput, especially in complex environments. A more balanced approach is required. We offered a probabilistic policy that reduces the throughput penalty by probabilistically forgiving users extra time for over-running jobs. The new policy is also difficult to manipulate, and offers the best results to users that provide the best estimates.

References

[1] NS-2 software package. Available at www.isi.edu/nsnam/ns/.

[2] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with Nimrod/G: Killer application for the global grid. *Proc. International Parallel and Distributed Processing Symposium*, 2000.

[3] J. Anderson, V. Bud, and U. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. *Proc. Euromicro Conference on Real-Time Systems*, 2005.

[4] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14, 2003.

[5] D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Technical Report 1308, University of Wisconsin, Madison, WI, 1996.

[6] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.

[7] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Proc. Heterogeneous Computing Workshop*, 2000.

[8] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. *Lecture Notes in Computer Science*, 1459, 1998.

[9] L. Dunning and S. Ramakrishnan. A heuristic cost estimation method for optimizing assignment of tasks to processors. In *Proc. Symposium on Applied Computing*, 1999.

[10] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3), 2002.

[11] T. Hagerup. Allocating independent tasks to parallel processors: An experimental study. *Journal of Parallel and Distributed Computing*, 47, 1997.

[12] R. L. Henderson and D. Tweten. Portable batch system: Requirement specification. Technical report, NAS Systems Division, NASA Ames Research Center, 1998.

[13] S. E. Hummel, E. Schonberg, and L. E. Flynn. Factoring: A practical and robust method for scheduling parallel loops. *Communication of ACM*, 35(8), Aug. 1992.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

[15] J. Lopez, M. Garcia, J. Diaz, and D. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. *Proc. Euromicro Conference on Real-Time Systems*, 2000.

[16] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2), Nov. 1999.

[17] C. S. R. Murthy and G. Manimaran. *Resource Management in Real-time Systems and Networks*. MIT Press, Apr. 2001.

[18] A. Takefusa, H. Casanova, S. Matsuoka, and F. Berman. A study of deadline scheduling for client-server systems on the computational grid. *Proc. High Performance Distributed Computing*, 2001.

[19] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a perfomance evaluation for global computing scheduling algorithms. In *Proc. High Performance Distributed Computing*, 1999.

[20] J. M. Wozniak, A. Striegel, D. Salyers, and J. A. Izaguirre. GIPSE: Streamlining the management of simulation on the grid. In *Proc. Annual Simulation Symposium*, 2005.

[21] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proc. Cluster Computing*, 1992.

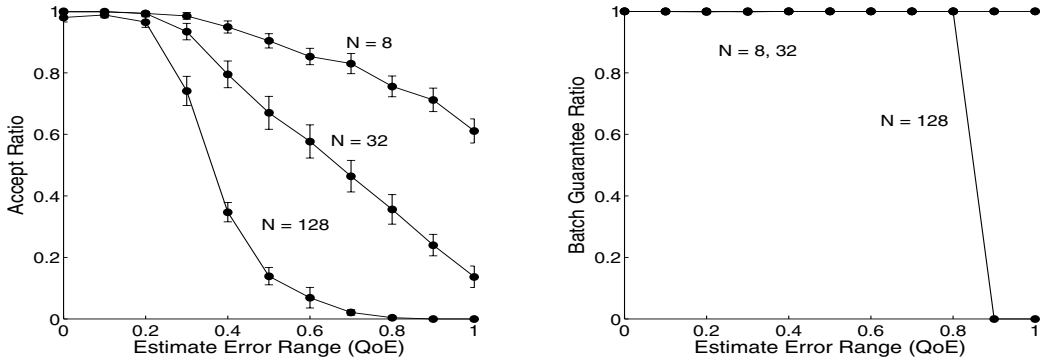

Figure 2. Acceptance and guarantee ratios for batches of jobs without enforcement. Batches of PolyDevice jobs.

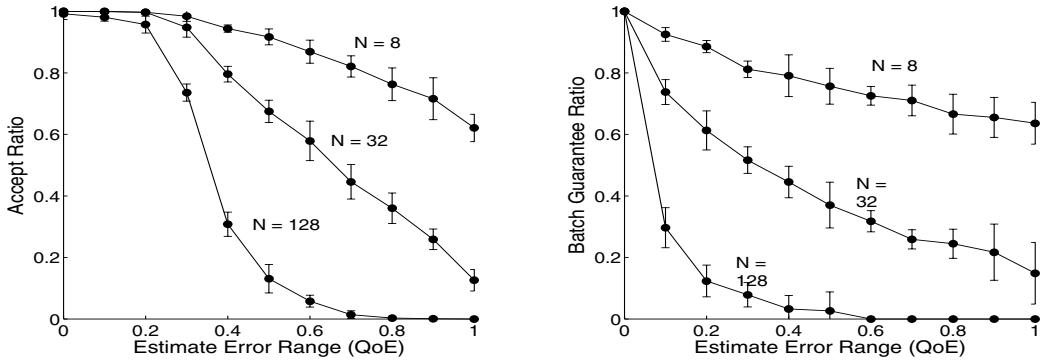

Figure 3. Acceptance and guarantee ratios for batches of jobs with hard enforcement. Batches of PolyDevice jobs.

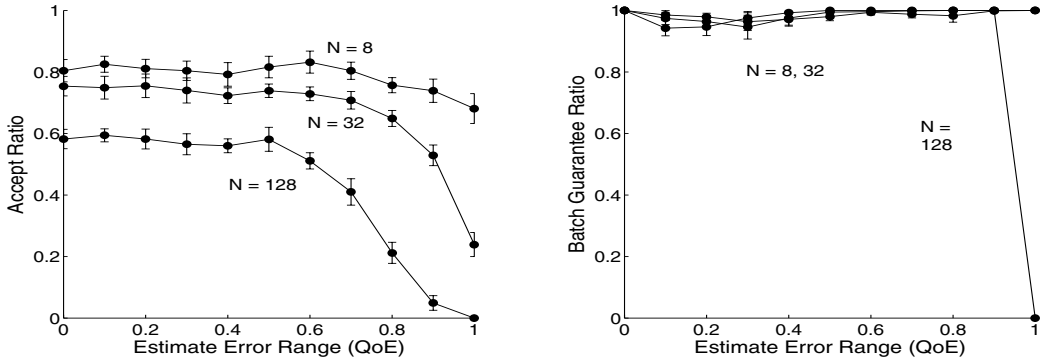

Figure 4. Acceptance and guarantee ratios for batches of jobs without enforcement. Batches of NS-Device jobs.

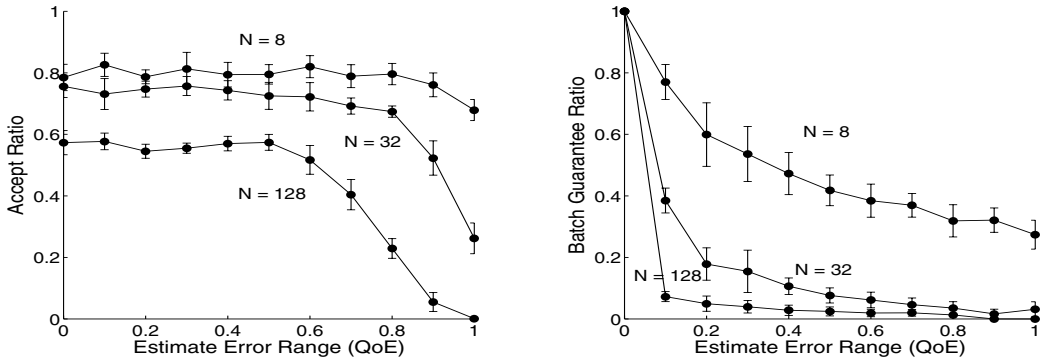

Figure 5. Acceptance and guarantee ratios for batches of jobs with hard enforcement. Batches of NS-Device jobs.

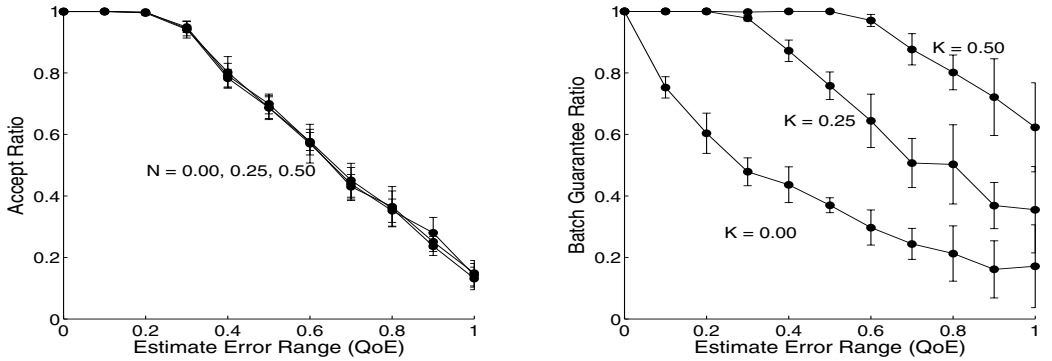

Figure 6. Acceptance and guarantee ratios for batches of jobs with enforcement level K. Batches of PolyDevice jobs.

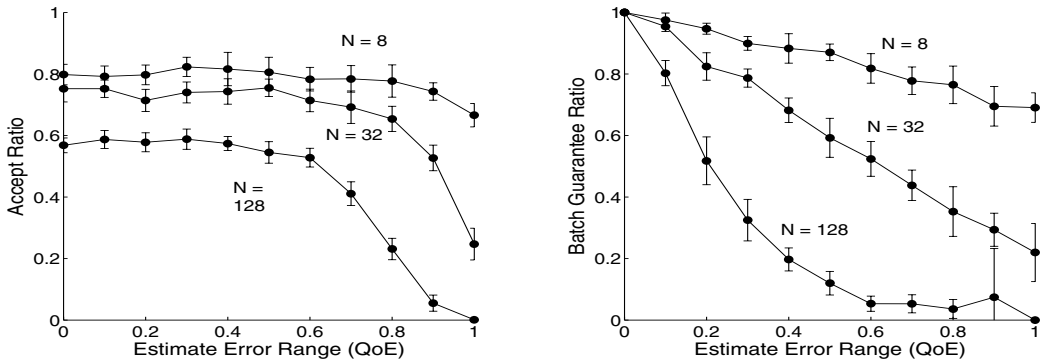

Figure 7. Acceptance and guarantee ratios for batches of jobs with probabilistic enforcement. Batches of NS-Device jobs.

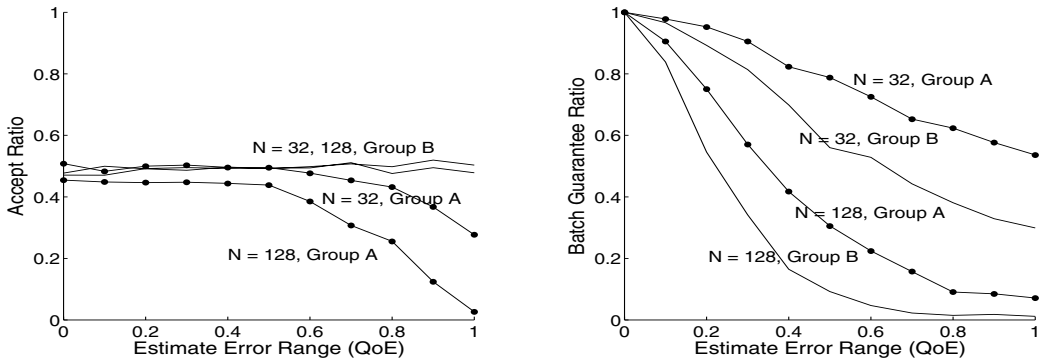

Figure 8. Acceptance and guarantee ratios for batches of jobs with probabilistic enforcement. Batches of NS-Device jobs. Data with dots indicates the Group A users, who provided estimates centered on the correct running time, undotted lines indicate Group B users, who provided consistent underestimates. This enforcement method demonstrates an intermediate approach between hard enforcement and no enforcement, with intermediate guarantee ratios as a result. As shown, if users can keep their running times within 50% of the estimate, they achieve similar acceptance ratios to users that deliberately underestimate, while obtaining much better guarantee ratios on their batches. (Standard deviation information removed for clarity.)

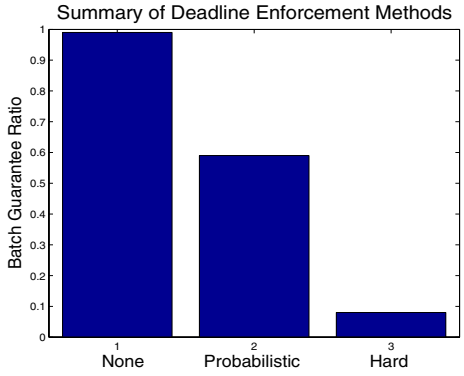

Figure 9. Guarantee ratios for batches of jobs with probabilistic enforcement. Batches of NS-Device jobs, N = 32, QoE = 50%. Data compiled from previous diagrams. All users Group A: unbiased estimates. In summary, no enforcement allows for most jobs to complete on time, because of necessary over-estimation for worst case input parameter sets. Since this policy is easily abused, this policy is compared against the probabilistic enforcement method and the hard enforcement method. This illustrates the trade-off between deadline enforcement and high throughput.