Investigating Deadline-Driven Scheduling Policy *via* Simulation with East

Justin M. Wozniak and Aaron Striegel
University of Notre Dame
May 10, 2008

Opportunistic techniques have been widely used to create economical computation infrastructures and have demonstrated an ability to deliver heterogeneous computing resources to large batch applications, however, batch turnaround performance is generally unpredictable, negatively impacting human experience with widely shared computing resources. Scheduler prioritization schemes can effectively boost the share of the system given to particular users, but to gain a relevant benefit to user experience, whole batches must complete on a predictable schedule, not just individual jobs. Additionally, batches may contain a dependency structure that must be considered when predicting or controlling the completion time of the whole workflow; the slowest or most volatile prerequisite job determines performance.

In this chapter, a probabilistic policy enforcement technique is used to protect deadline guarantees against grid resource unpredictability as well as bad estimates. Methods to allocate processors to a common workflow subcase, barrier scheduling, are also presented.

1 Introduction

Running complex applications on widely distributed resources is an unpredictable process, complicating the user experience with new systems. While opportunistic technologies and grid infrastructures dramatically increase the resources available to the application, they also increase the range and volatility of resulting behaviors. Job turnaround time, the span between the time a job is ready to run and the time results are returned, is of primary importance to users seeking larger, more powerful computing platforms, but is more difficult to measure on conglomerations of heterogeneous computation elements. The fact that users cannot easily achieve predictable turnaround results – even in the presence of increased available parallelism, and when average case performance is improved – *can cause frustration and reduce interest* in new, complex distributed computing systems.

There is a fundamental disconnect between short-term user objectives and long-term system design techniques that underlies many user frustrations with commodity computing systems. Users prefer responsive, fast turnarounds for specific workloads. Grid system designers and administrators take a long view, intending to maximize utilization, and thus the return on investment, for a given resource set, given a wider range of applications. These viewpoints translate into the various technologies available. For example, opportunistic systems (Thain, 2004) excel at improving system utilization start by locating idle resources and employing them to perform useful work. Real-time systems (Murthy, 2001), contrarily, start by admitting acceptable workloads and ensuring that the results will be returned on schedule. Ordinary workstations are a middle ground, providing a necessarily available resource that offers moderate predictability through system simplicity and isolation from external forces.

1.1 Sources of Unpredictability

We start with an examination of the underlying causes of unpredictable performance in opportunistic systems.

- *Processor heterogeneity:* Since users of opportunistic systems often employ resources owned and managed by diverse organizations, heterogeneity is an ever-present challenge. CPU heterogeneity takes two forms: "hard" heterogeneity, meaning architectural differences, and "soft" heterogeneity, meaning performance differences within a class of compatible architectures.

  While users can benefit from existing matchmaking techniques to select certain processors, opportunistic systems have been designed for "compute-hungry" users. These users are capable of consuming an ever-growing number of processors. Thus, they are expected to overcome hard heterogeneity obstacles by compiling for multiple architectures or using portable interpreted languages. Consequently, the impact of heterogeneous systems can be modelled by simply considering performance differences.

- *Contention:* Opportunistic systems attempt to harness the aggregate computing ability of large numbers of processors for large numbers of users. The requested workload for such a system can thus be quite variable, particularly in small scale, experimental settings. For example, in a typical university-sized Condor installation, a user may initially have access to all of the available machines, but then unexpectedly be forced to split the resources with another user.

Heterogeneous, opportunistic systems thus pose two significant performance predictability problems for users. First, performance analysis prediction for a given task on a range of potential architectures is a labor-intensive process that is not standard practice in distributed computing, nor is it appealing to potential new users of complex systems. Second, since micromanagement of job distribution is also a complex project that must take into account the contention for resources among multiple users, the actual execution site of a task in an opportunistic system is often left to the metascheduler. Consequently, while the benefits of tight runtime estimates are clear, modern systems must recognize that typical cases will rely on rough estimates given by users, and are not particularly trustworthy due to the unpredictable allocation of imperfectly understood processors.

1.2 Real-time Computing Approaches

The difficulties experienced by users of opportunistic systems can be ameliorated in a variety of ways.

- *The deadline computing model:* In a typical single workstation or symmetric multiprocessor environment, deadlines are a natural but often unstated aspect of computing. Users are aware that jobs must be completed within a certain known time frame- when anomalous runtime performance appears, given predictable resources and low contention, they simply turn to software defect investigations. However, more complex computing systems built upon an opportunistic fabric require autonomic schedulers to maintain the expected level of predictability by managing the progress of jobs and workflows. These schedulers require an explicit user-supplied expected runtimes and deadlines with which to operate. Then, given a set of processors and contending users and jobs they can optimize system utilization to minimize unpredictability in the sense of reducing deadline misses.
- *Admission control:* Users of a deadline-aware scheduler gain the additional benefit of certain fail-fast behavior in the form of deadline rejection. For example, in a simple case in which the user provided estimate exceeds the user provided deadline, simple job rejection can force the user to reconsider the attempted workload. In the more complex case of multiple existing jobs with

deadlines, if the new workload cannot be scheduled, the system can quickly inform the user that the work cannot be scheduled. However, once a job has been admitted to the system, the acceptance constitutes a guarantee that the deadline will be met. The automatic services provided by the scheduler must attempt to protect the deadline by allocating resources and managing contention as necessary.

A scheduler that implements these real-time computing concepts may be called a *deadline-driven system*. While users of such a system can be expected to be able to offer reasonable statistics and requirements in the form of estimates and deadlines, a significant *semantic shift* in job submission must be considered from both a human and a technical perspective. Real-world users are simply not used to having to provide these figures along with job submission. Thus, certain systems designers have begun to reduce the required user work in this area by building historical or analytic runtime estimators. Technically, these figures are not part of typical job submission semantics in workstation systems or opportunistic systems.

A feature that is present in typical computing systems is the concept of job priority. Processing resources are expected to be fairly distributed, possibly with the additional ability to boost performance for users considered more important by the scheduler or resource involved. However, in a deadline-driven system, the outstanding guarantees must be considered, creating a second semantic shift. Fair-share systems are thus augmented by the ability to consider and prioritize resource allocation with respect to the schedule that must be met.

While these features are intended to improve user experience with unpredictable resource fabrics, the pervasive unreliability of the resources themselves is capable of defeating the best efforts of a deadline-aware job manager. Additionally, erroneous estimates may disrupt a feasible schedule, pushing many jobs past their deadlines. First, it must be recognized that such a system must be considered a soft real-time system, in that deadline misses that are small from a user perspective are undesirable but usually acceptable. Second, it must be recognized that bad estimates will be offered even by well-meaning users because of the difficulty predicting job runtime on complex resources, and that this should be tolerated and corrected by the system-penalizing users for rare, small errors will again damage the user experience. Consequently, the scheduler must be capable of managing the system in difficult circumstances, in a way that achieves several objectives, including good predictability, performance, and utilization.

1.3 Trade-offs

These two computing paradigms – opportunistic and real-time computing – result in trade-offs that must be made by system designers. Increasing the predictability of opportunistic systems through real-time techniques increases the set of responsibilities given to the scheduler and poses constraints on fair-share processor allocation. Additionally, deadline guarantee strategies such as over-provisioning reduce the utilization of volunteered resources, constituting a direct ideological challenge to these systems.

Improving the predictability of user jobs on opportunistic resources requires a careful balance between these disciplines. Schedulers intended to straddle this design space must be capable of reducing the impact on the users and systems that it connects. Studying the effects of such schedulers before implementation is difficult because the required experimental testbed would need to consist of a large number of machines comparable to those used in the desired setting. Additionally, the effects of heterogeneous, unreliable hardware is difficult to mimic. Therefore, simulation is typically used as a low-cost first step in the trial of new scheduling strategies. These tests are tied to reality by the ability of the software to incorporate real-world trace data from existing computational systems. Then, new strategies may be applied to real-world workload cases to obtain predicted performance characteristics.

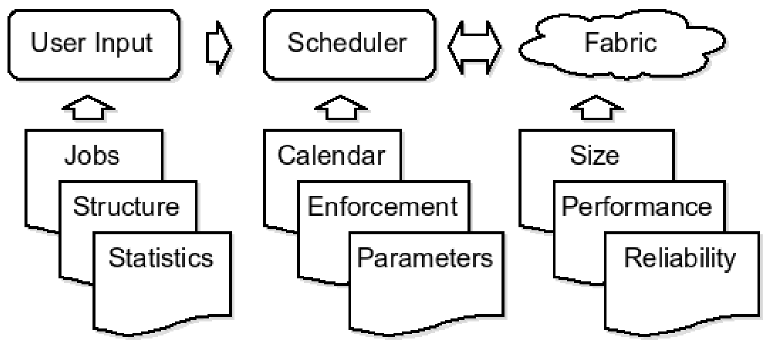


Figure 1. Components of a scheduler simulator.

The remainder of this chapter is organized as follows. Section 2 describes policy questions to be investigated in the grid computing community. Next, Section 3 frames a simulation-based approach to exploring the intersection of these areas by describing a simulator framework that provides insight into the effects of novel scheduler behaviors. Then, Section 4 presents an investigation of probabilistic policy enforcement methods, and Section 5 describes barrier scheduling. Finally, Section 6 offers several concluding remarks.

2 Investigations in Scheduler Policy

East enables scheduler architects and policy makers to bridge the gap between real-time computing and grid computing- areas which traditionally have very different objectives. For example, in a hard real-time system, jobs are scheduled with respect to their worst-case computation time, and any deviation exceeding schedule limits is expected to result in termination. Therefore, some previous work has investigated how conservative schedulers must be when computing on the grid. This may be combined with heuristic methods to approximate optimal workflow task placement to pack batches onto the grid, improving performance and predictability. Our work, however, intends to satisfy the grid objective of high utilization and prevent job termination due to *relatively small* runtime fluctuations while providing a high guarantee ratio.

While a variety of grid-enabled schedulers have been proposed such as Globus GRAM (Czajkowski, 1998), GridBus (Venugopal, 2004) , GrADS (Dail, 2002), and others, quality requirements are often managed through strict business-flavored structures such as service level agreements, which have seen rapid recent entry into grid computing (Yarmolenko, 2006). The scheduler model here, however, intends to gain the functional benefit of timely computing while maintaining the benevolence of opportunistic computing by the construction of a lightweight, autonomic front-end scheduler. This component is the object of the policy study presented here.

3 Simulator Architecture

To approach these policy investigations, an appropriate simulator structure must be developed. Scheduler simulators must incorporate three major aspects of the problem as shown in Figure 1:

- *User input:* User input to the system takes the form of submitted jobs, estimates, and deadlines. These jobs may be presented as individual executions, batches of interdependent jobs, or as structured workflows of dependent tasks. Such workloads may be obtained through idealized models or through traces of real-world workloads that include execution information and the resulting statistics.
- *Scheduler behavior:* The simulated

scheduler is the object of study and thus the heart of the simulator model. However, as the policy tester attempts to improve scheduling results, it must be possible to *plug in* new schedulers for rapid evaluation, through parameter tuning as well as algorithmic overhaul.

- *Resource fabric:* Finally, the resource fabric must also be modelled by the simulator. These models may be quite simple for initial tests, but the validity of the returned results depends on the similarity of the model to typical or specific real-world infrastructures, including the effects of heterogeneity and unreliability. Additionally, resource model inputs to the system must be correlated with any trace data used as user input to the system.

System assembly is shown in Figure 1 as vertical arrows that incorporate subcomponents into the structure. Internal simulator runtime interactions are shown as horizontal arrows that transmit time-dependent events among components. Once the three-component system has been assembled, the simulation may be executed over time by modelling the underlying events as they occur. Since most events relevant to typical experiments are discrete – such as job start and stop events or machine availability changes – discrete event simulation is a viable choice, operating at a relatively high level on relatively large entities such as whole jobs.

As an example implementation of this simulation model, a simulator called East (Wozniak, 2007) was constructed to allow the rapid construction of each of the three components by a flexible software architecture that allows subcomponents to be quickly plugged into the assembled system. The system performs a discrete time experimental evaluation of idealized or trace-based workloads.

East simulates the distributed computing case in which a deadline-aware front-end scheduler accepts client requests and services them by employing pre-existing simple batch queues. This model is proposed as an alternative to a stovepipe solution because it simplifies the construction of the new system and reduces the risk involved in the deployment of the hypothetical new metascheduler. By controlling pre-existing resources we intend to obtain any required scheduling properties. East is ultimately a first step towards the construction of the real system, and may be used to examine the hypothetical behavior of future schedulers.

4 Characteristics of Scheduler Policy

In this section, we present two examples of deadline-driven scheduler policy that may be examined by simulation. The utilization trade-off is presented, a property which makes it difficult to provide services meeting multiple performance demands. Secondly, the quality of estimates is considered as policing strategies may be used to promote user behavior that enhances system characteristics.

4.1 The Utilization Trade-off

As described in the introduction, a trade-off exists between high utilization systems and predictable, timely systems capable of delivering high guarantee ratios. Utilization is a metric that measures the effectiveness of the shared resource arrangement as negotiated by the cooperating parties. Commodity grid computing and storage systems intend to make the most of the available resources primarily by increasing their utilization. Once this is accomplished, other more advanced grid techniques are used to provide other desired qualities. However, high utilization systems are simply *too busy* to provide the timely service that users may require in time-sensitive applications. An example of this is shown in Figure 2. In this simulated experiment, a range of workloads are presented to an admission-regulated scheduler that provides guarantees based on user-provided estimates. For each input workload, a resulting utilization and guarantee ratio may be observed from the resulting simulator output. In this case, a simple cluster of 10 homogeneous computers was presented with single job submissions, each paired with an estimate good within 50% of the actual runtime. Over-provisioning was applied at

25% above the estimated required computation time. As shown in Figure 2, the resulting behavior forms a band of likely characteristics that decreases as the workloads degrade the quality of the guarantees available from the scheduler.

4.2 Quality of Estimates and Contention

While it is assumed that users of the opportunistic systems are generally benevolent and desire to improve the user experience for all stakeholders, the system presented thus far is easy to manipulate if estimate enforcement is not applied. Since the admission control system takes user estimates and uses them to schedule jobs with respect to deadlines, erroneous or malicious under-estimates can defeat the whole schedule, causing swaths of deadline misses and penalizing all users.

Thus, the motivation of the following experiment 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 short, when multiple users offer bad estimates, who should pay the price?

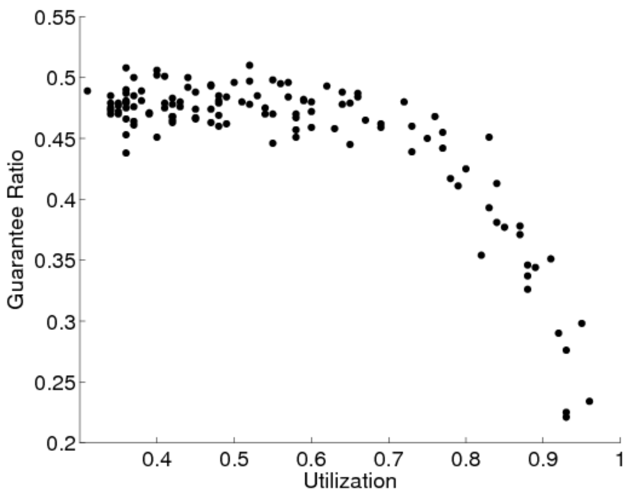


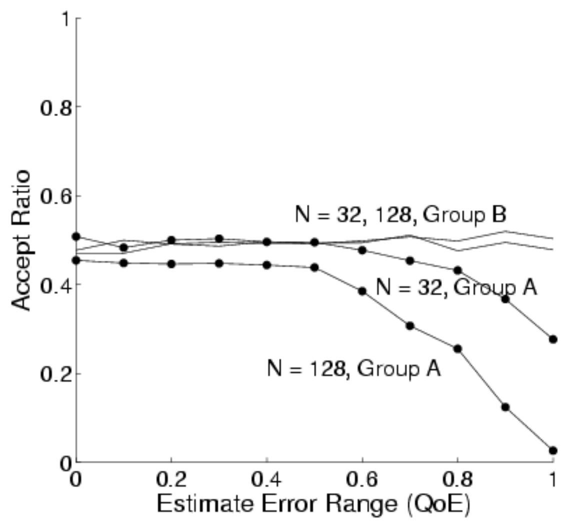
Figure 2. Guarantee ratio under increasing utilization stresses.

An estimate enforcer may be applied in the scheduler layer to simply terminate all user jobs that exceed their estimates, but given the known unpredictability of grid resources and the sensitivity of large interdependent batches of jobs or workflows, killing jobs that only exceed their estimate by a small amount would result in poor results even for well-intended users. Additionally, known system behavior such as simple over-provisioning may be manipulated just as easily as no enforcement, by providing careful underestimates intended to slip into a highly utilized schedule but abuse the system by intentionally overrunning the alloted time.

A probabilistic enforcer is thus applied, which embraces the unpredictability of grid resources while preventing wild unreliability due to schedule problems. This component terminates jobs that have exceeded their estimate with a probability proportional to the estimate violation. Thus, users that stay close to their estimates are likely to receive good performance, but users that attempt to manipulate the system are unlikely to receive desired results.

To demonstrate the success of this technique, two contending user groups were simulated with the East simulator. *Group A* provided job estimates centered on their true runtimes, while *Group B* provided consistent underestimates. Both groups submitted similar jobs to a heterogeneous cluster of *N* hosts.

As shown in Figure 3, both user groups are simulated as they provide estimates of varying quality (QoE). As the error in the given estimates increases, indicating more unpredictable grid conditions, Group B is able to maintain a high acceptance ratio by simply misrepresenting expected job performance. However, a correction is applied by the enforcer, resulting in better guarantee ratios for the Group A users. This guarantee ratio advantage is maintained even in highly unpredictable settings.



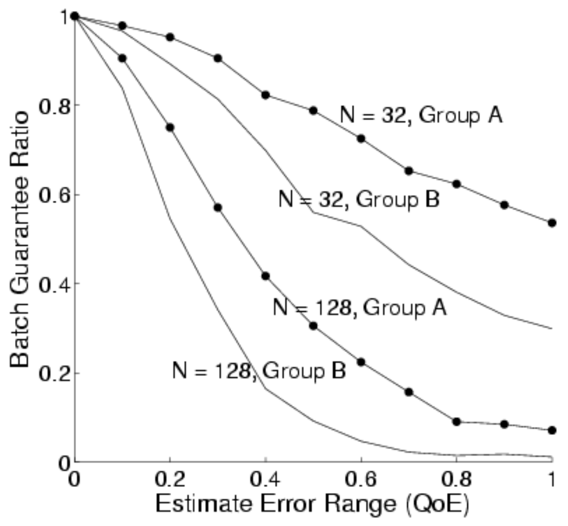


Figure 3. Schedule results for contending users.

The intended consequence of simulating proposed advanced scheduling and enforcement methods is to moderate the effects of the utilization trade-off. In the case of probabilistic enforcement, this is attempted by promoting good user input, resulting in better system performance as measured by multiple metrics.

5 Barrier Scheduling

A more complex scheduling example is exemplified by barrier-dependent computations. A multiprocessor barrier operation is a programmatic step which must be passed by all processes within a barrier group at the same time. This method may used to begin a synchronized all-to-all communication or, in a Monte Carlo setting, to allow intermediate processing to interleave rounds of parallel computation. This operation may be phrased in deadline-driven terms by specifying that the deadline for a batch is the time that the fastest job hits the barrier. Executing barriers on the grid is challenging for a variety of reasons, including 1) heterogeneity of computation resources, 2) resource unreliability, and 3) the potentially high cost of job migration. As a motivating example, previous work scheduling a barrier-dependent molecular dynamics computation in an opportunistic computing system used dedicated clusters of faster processors to advance jobs that were lagging behind or encountered resource problems (Woods, 2005). In this section, we will describe how this concept may be simulated with respect to arbitrary system parameters.

The barrier scheduling problem may be generalized for an arbitrary case in which multiple competing users schedule these workloads. In these complex, unpredictable cases, certain user jobs will lag behind their peers, resulting in potential lost utilization. The simulated model for barrier dependent jobs in East, consists of a batch of jobs. Within each batch, all jobs must pass a certain number of barrier in a synchronized manner, thus all jobs in the batch are part of the same barrier group. Jobs are further divided - as if by a checkpoint technique – into segments that may be quickly examined to determine which jobs are lagging behind.

The first technique that may be applied in such a case is for a job to block upon reaching the barrier until all other jobs in the group have reached the barrier, at which point all proceed. However, this is highly deadlock-prone: as each job reaches the barrier, the number of available processors decreases, increasing the probability of the system running out of free processors.

Assuming a small number of high

performance processors are available for use, these may be isolated into a cluster reserved for jobs that lag behind in an attempt to provide timely performance near the barrier. However, this architecture removes nodes from general use, thus trading utilization for timeliness in an attempt to improve overall throughput. This setting was modelled in East by in a homogeneous cluster of 24 hosts augmented by a high performance cluster of 8 hosts that run 4 times as fast as the normal hosts. A varying number of the high performance hosts were reserved for specialized usage.

As shown in Figure 4, however, this architectural method can fail in seemingly useful cases. In this case, batches of jobs varying in runtime by up to a factor of 10 were submitted to the cluster. Jobs that were found to be delaying a barrier transition were promoted to the reserved cluster. However, this strategy did not succeed in enhancing overall batch completion time. Any gains that were made by promoting the timeliness of lagging jobs were lost to the underutilization of the fastest hosts in the cluster. It should be noted that the model used here did not take into account job migration time, which could additionally impact scheduling strategies.

6 Conclusion

Overall, the matter of integrating concepts from grid computing and real-time computing involves trade-offs. Common real-time strategies such as reservations and overprovisioning result in low resource utilization. Grid computing performance strategies such as global job-data locality improve system throughput but do not benefit individual jobs directly or take a schedule into account; additionally, while the grid may function under partial failure and resource heterogeneity it does not protect the schedule from these effects. New methods to be developed through simulation must combine the ability to work with high-utilization average case estimates and probabilistic policies to promote schedule predictability in competitive settings.

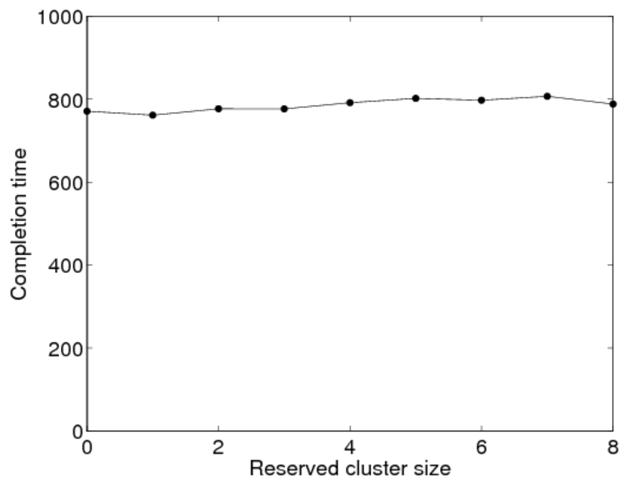


Figure 4. Barrier scheduling performance

References

- (Czajkowski, 1998) Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke, “A Resource Management Architecture for Metacomputing Systems,” Lecture Notes in Computer Science 1459, Pages 62-82, 1998.
- (Dail, 2002) Holly Dail, Henri Casanova, and Fran Berman, “A Decoupled Scheduling Approach for the GrADS Program Development Environment,” Proc. Supercomputing, Page 55, November 2002.
- (Murthy, 2001) C. Siva Ram Murthy and G. Manimaran, "Resource Management in Real-time Systems and Networks," MIT Press, USA, April 2001.
- (Thain, 2004) Douglas Thain, Todd Tannenbaum, and Miron Livny, “Distributed Computing in Practice: The Condor Experience,” Concurrency and Computation: Practice and Experience, Volume 17, Issue 2-4, Pages 323-356, February-April 2005.
- (Venugopal, 2004) Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton, “A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids,” Proc. Workshop on Middleware in Grid Computing, Pages 75-80, October 2004.
- (Woods, 2005) Christopher J. Woods et. al., “Grid Computing and Biomolecular Simulation,” Philosophical Transactions of the Royal Society A, Volume 363, Number 1833, Pages 2017-2035, August 2005.
- (Wozniak, 2007) Justin M. Wozniak, Yingxin Jiang, and Aaron Striegel, “Effects of Low-Quality Computation Time Estimates in Policed

Schedulers," Proc. Annual Simulation Symposium, Pages 283-292, March 2007.

- (Yarmolenko, 2006) Viktor Yarmolenko, Rizos Sakellariou, "An Evaluation of Heuristics for SLA Based Parallel Job Scheduling," Proc. High Performance Grid Computing Workshop, April 2006.