Decentralized Scheduling of Bursty Workload on Computing Grids

Juemin Zhang, Ningfang Mi, Jianzhe Tai and Waleed Meleis
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA 02115
{jzhang, ningfang, jtai, meleis}@ece.neu.edu

***Abstract*—Bursty workloads are often observed in a variety of systems such as grid services, multi-tier architectures, and large storage systems. Studies have shown that such burstiness can dramatically degrade system performance because of overloading, increased response time, and unavailable service. Computing grids, which often use distributed, autonomous resource management, are particularly susceptible to load imbalances caused by bursty workloads. In this paper, we use a simulation environment to investigate the performance of decentralized schedulers under various intensity levels of burstiness. We first demonstrate a significant performance degradation in the presence of strong and moderate bursty workloads. Then, we describe two new hybrid schedulers, based on duplication-invalidation, and assess the effectiveness of these schedulers under different intensities of burstiness. Our simulation results show that compared to the conventional decentralized methods, the proposed schedulers achieve a 40% performance improvement under the bursty condition while obtaining similar performance in non-bursty conditions.**

I. Introduction

Grid computing [4] is an important way to transparently harness distributed computing power to meet the demands of computation-intensive applications. A computing grid consists multiple parallel computing systems such as clusters and mainframe computers. These systems, referred to as computing sites, are commonly owned by different institutions, separated geographically, distributed on a wide area network and managed by individual resource management systems. A computing grid allows participating computing sites to be highly autonomous [5]. In contrast to traditional parallel and distributed computing, which manages resources within a single administrative domain, grid computing may span different resource management systems (RMSs). Individual RMSs on a grid may not use a single centralized RMS, or share an unified scheduling policy. Furthermore, a computing grid can inter-connect massive computing resources, aggregated from a number of computing sites and distributed over a wide area network such as the Internet. A centralized RMS or a grid-level scheduler may not be efficient or scalable.

The problem of allocating resources in this distributed grid environment is challenging and unsolved. The key questions are how to quickly identify and reserve computing resources specified by users, and how to balance workloads among all computing sites. Without using a centralized scheduler, or a grid-level job queue, we approach this problem by developing a decentralized scheduling framework [18]. Our decentralized scheduler, which runs at the user/application level, submits jobs to the site with the shortest estimated queue time.

Burstiness in workloads is often observed in systems including grid services [10], multi-tier architectures [12], and large storage systems [15]. Studies [7], [11], [12], [3] have shown that due to creating periods of continuous peak arrival rate, burstiness dramatically degrades system performance with system overloading, uncontrolled increase of response time and service unavailability. Thus, negative performance impacts of burstiness cannot be ignored in system design. How to counteract burstiness to maintain high system performance and availability becomes imminently important but challenging as well.

Many scientific applications which target computing grids involve a large number of concurrent and dependent jobs, which can be executed either in parallel or sequentially. Simultaneously launching jobs from different applications during a short time period can immediately cause a significant arrival peak, which further aggravates resource competitions and load unbalancing among computing sites. As a result, it is critical to investigate performance impacts of burstiness and take into account its deleterious effects in grid and cloud computing. However, conventional methods *only* consider exponentially distributed interarrival times between jobs, which unfortunately neglect cases of bursty arrivals and cannot capture the impacts of burstiness on system performance.

Motivated by this problem, in this paper we define a new methodology for effective resource allocation under bursty workload conditions. We use trace-driven simulations running in a Scheduling-Grouping-Relay (SGR) system [19], which implements a new dynamic resource allocation model using decentralized scheduling, to investigate how bursty workloads affect the performance of decentralized schedulers for computing grids. We show that burstiness in arrival flows does cause scheduler performance to deteriorate. After illustrating that performance can be further degraded if schedulers do not address burstiness in arrivals, we give evidence that performance improvement can be achieved by using the duplication-invalidation method. To address the load unbalancing problem caused by burstiness, we further propose new *hybrid* schedulers to balance bursty workloads across all computing sites, and thus to improve the overall system performance. Detailed simulation results validate the effectiveness of our new proposed schedulers under both bursty and non-bursty workloads.

II. Simulation environment

In this section, we first describe the simulation environment for evaluating performance under various schedulers and then evaluate the degree of impacts on scheduler performance under different intensities of burstiness.

The simulation is developed on the CSim library [1]. In this paper, we limit our investigation to 8-site homogeneous computing grids. Each site of the grid has 64 processors and is managed by the host resource management (RMS), or job queue, employing the conservative backfill queuing policy [13].

Our study focuses on computational intensive applications that consist of parallel jobs. A parallel job description includes the job size (or the number of processors) and its estimated execution time. In our simulations, the job size, estimated execution time, and actual runtime are generated randomly. A computing grid, constructed based on multiple computing platforms, is a shared environment among different groups of users. We simulate scheduling of two types of jobs, local and global. Local jobs are submitted to specified computing sites, simulating applications from local users each of whom is allowed to access resources within one site. Global jobs can be submitted to and executed on any site, simulating applications targeting the computing grid. A local workload, consisting of $100,000$ of local jobs, is generated for each site, creating a level of resource utilization on the site. Interarrival times for local jobs are exponentially distributed, while creating a load level of 0.6 for each site. Additionally, there are $100,000$ global jobs in the global workload.

Based on the SGR framework, we implemented ranking-based decentralized scheduling for the global workload in the simulation environment. Figure 1 illustrates the structure of the simulation environment with the decentralized scheduler, local and global workloads, host resource management systems (RMSs) and computing site. The main task of the global job scheduler is to perform site selection and job submission. To select a site for a global job, the scheduler periodically query the load condition of all sites from the host RMSs. The scheduler selects the most desirable computing site by ranking all sites based on the load condition, such as the shortest job queue length, or the shortest estimated queuing time. Consequently, the scheduling process consists of ranking criterion query, ranking procedure, and job submission. The ranking criteria we investigate include: (1) *Rand*, which allows random site selection, (2) *Qlen*, minimum queue length, (3) *Util*, minimum system utilization, (4) *Est. QT*, minimum estimated execution time of all queued or running global and local jobs, and (5) *Act. QT*, minimum total actual execution times of all queued or running jobs[1]. The *Act. QT* scheduler produces ideal scheduling results which are used for the performance evaluation purpose only.

[1]The actual service times for all queuing jobs are assumed to be known in advance under the *Act. QT* algorithm.

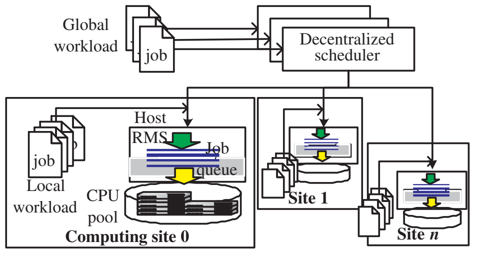

Fig. 1. The structure of the simulation environment.

A. Remote submission delay

In real distributed systems, it takes time for the global job scheduler to get the load information of all computing sites and to compute their rankings. Similarly, when a global job is submitted to a site, it also takes time for that site's RMS (or job queue) to receive the job request and process it. To simulate this effect, we impose a remote submission delay for each global job in its scheduling process. After the ranking procedure completes, the global job is not submitted to the top-ranked site immediately. Instead, it will be held for a fixed period of time, *delay*, before its submission in our simulation.

The submission delay represents system responses time between the arrival of a global job and the corresponding system workload change at the selected site's RMS. Because of the delay, global job submissions will not change the system load status immediately after its scheduling decision is made. The decentralized schedulers, running independently from one another, may make obsolete decisions for global jobs based on the outdated information. For example, multiple global jobs arriving within the period of delay could be scheduled to the same top-ranked site. Consequently, surges of system load on the top-ranked computing site may not be detected by the decentralized schedulers in time. Therefore, bursty workloads will cause significant load increases on the particular sites, resulting the performance degradation. Unless there is $0 - delay$, the decentralized schedulers are able to detect system load changes right away, and no excessive jobs are scheduled to the same top-ranked site. However, such a scenario is unrealistic in the computing grid environment, where long communication delays and large system overheads are commonly observed.

B. Impact of burstiness

In implementations of our decentralized schedulers that use the SGR framework, each scheduling process is launched by an individual application or user, and runs independently from one another. It is therefore not possible to detect an incoming job burst, nor to coordinate the scheduling of jobs arriving in bursts by the decentralized schedulers. Consequently, the SGR schedulers could suffer large performance degradation in bursty conditions, as we show below.

To demonstrate the performance impact of bursty arrivals, we run simulations under three different global workloads, created by three intensities of burstiness, including strong bursty, weak bursty and non-bursty. The bursty global work-

	Rand scheduler		*Qlen* scheduler	
Workload	Mean	$\geq$ 100s	Mean	$\geq$ 100s
Non-bursty, ID = 1	48s	15%	6s	0.3%
Weak bursty, ID = 55	82s	29%	102s	31%
Strong bursty, ID = 384	125s	48%	405s	75%

TABLE I
THE MEAN JOB QUEUE TIMES AND PERCENTAGES OF JOBS WITH QUEUE TIME $\geq$ 100S UNDER THE THREE WORKLOADS AND THE REMOTE SUBMISSION DELAY IS 1.

loads (i.e., strong bursty and weak bursty) are drawn from 2-state Markov-modulated Poison Processes (MMPPs) [2] with the mean of 10s and squared coefficient-of-variation (SCV) of 20, but different burstiness profiles, while the non-bursty workload is generated from an exponential distribution with the same mean of 10s.

Table I shows that the performance (e.g., the mean job queue times) of both *Rand* and *Qlen* schedulers decreases dramatically under strong and week bursty conditions. We also present the index of dispersion (ID) [8] of the three traces in the table. Index of dispersion has been widely used to capture the intensity of burstiness: high values of ID usually indicate stronger burstiness in workloads. In addition, the effects of burstiness on the two schedulers are very different. While the performance under the *Rand* scheduler decreases by approximately 2 times in the strong bursty condition, *Qlen*'s performance decreases by more than a factor of 60. Such performance degradation effects can also be observed in the distributions of job queue times. As the intensity of burstiness increases, *Rand* and *Qlen* obtain longer tails in the distributions of job queue times, see Table I.

As we discussed in the previous subsection, a global job scheduler which can detect system load surge on computing sites (i.e., $delay = 0$) is able to perform better than the one which cannot in bursty conditions. Here, we show the evidence of this expected performance. In Figure 2, we show the performance gap, see Eq.(1) between the four schedulers and the ideal scheduling result, i.e., system load surges can be detected ($delay = 0$) and all global jobs are assigned to the sites with the shortest queue times (*Act. QT*). As the simulation granularity in CSim is $1s$, we here set the remote submission delay to $1s$. We remark that $delay$ can be varied under different system configurations, e.g., network bandwidth, and system processing speed.

$$Perf\_Gap = 100\% \times \frac{\text{Scheduler's Perf} - \text{Ideal's Perf}}{\text{Ideal's Perf}} \quad (1)$$

Figure 2 shows that under the strong bursty condition, the *Rand* scheduler maintains similar performance (i.e., around 40% performance gap), regardless of the different values of the remote submission delay. While *Qlen*, *Util* and *Est. QT* perform better than *Rand* when *delay* is 0, their performance deteriorates dramatically when *delay* is 1, allowing the job queue time to increase to approximately 4 times longer than the ideal scheduling result. Therefore, we argue that such deleterious effects due to burstiness and remote submission delay must be considered in the performance evaluation and scheduler design for computing systems.

[2]Markov-modulated Poison Process (MMPP) is a special case of the Markovian Arrival Process (MAP) [14], which is used here to generate bursty flows because it is analytically tractable and allows us to generate traces with the same Probability Density Function (PDF) but different burstiness.

We summarize that although instantaneous detection of system load surges now becomes critical for system design, such an instantaneous detection may not be possible when computing sites in a distributed environment are geographically separated. When a large number of jobs arrive in a burst and are scheduled to the same computing site by the decentralized schedulers, this site could be overloaded in a very short period, while other sites remain lightly loaded. To avoid this load unbalancing, jobs arriving in a burst should be assigned to different computing sites. We therefore develop new schedulers in this paper, which can efficiently utilize available resources by taking into account burstiness effects and introducing various levels of randomness in balancing workloads when the schedulers cannot detect system load surges.

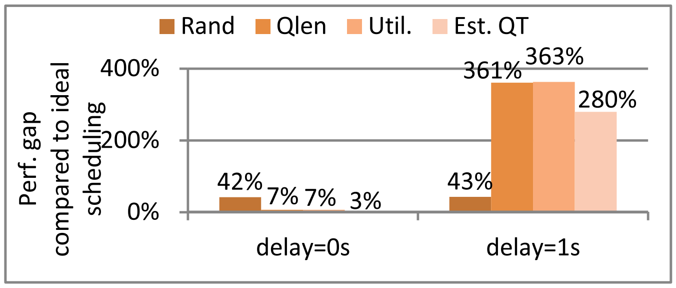

Fig. 2. Performance gap between the results of four decentralized schedulers and the ideal scheduling result (delay=0 and *Act. QT*) in the backfill grid under the strong bursty condition.

III. HYBRID SCHEDULERS AND PERFORMANCE EVALUATION

Now, we turn to describe our methodology for grid and cloud computing. Our proposed hybrid schedulers are evaluated in different workloads with varied intensity of burstiness.

A. Duplication and hybrid schedulers

We have demonstrated the significant performance impact of bursty job arrivals on computing grids when using decentralized schedulers. We have also observed the advantage of scheduling jobs randomly across all computing sites in bursty conditions, as shown in Figure 2. To address the load unbalancing problem, we therefore propose new hybrid schedulers that incorporate some randomness.

Based on the SGR framework, a global job can be duplicated multiple times, and these copies are submitted to those top-ranked sites simultaneously. Duplication allows execution of a job to be chosen dynamically among multiple computing sites; a job can avoid being executed on a heavily loaded site, since it starts to run on a lightly loaded site first. The remainder of the redundant duplications are invalidated without consuming resources to execute the duplicated jobs. Note that this allocation mechanism does not waste resources as *only* one duplication will be executed while all the remaining ones are just queued at the computing sites. In addition, the system penalty (e.g., 1 second) will be added for each duplication invalidation. We refer the interested readers to [19] for the details of duplication invalidation. If we assign jobs in bursts and their duplications to multiple randomly selected computing sites, then the bursty workload can be shared among these sites, therefore alleviating the imbalance of load.

Here, we propose two new hybrid schedulers by using the duplication-invalidation method. The hybrid scheduling combines the *Qlen* scheduler with the *Rand* scheduler. Given a global job, the *Hybrid-1* scheduler uses the shortest queue length to locate the site for the first submission, and the remaining duplications are submitted to sites that are randomly selected. The *Hybrid-2* scheduler uses the queue length as the ranking criterion to select two sites for submissions, and the remaining duplications are submitted to randomly selected computing sites.

B. Performance in the strong bursty condition

In Figure 3, we compare the scheduler performance under the strong bursty condition and 1-delay to the ideal scheduling result. Testing the grid with the backfill policy, we select four schedulers for comparison, giving a range of duplications. We observe a significant performance improvement by using duplication for all schedulers, with decreases in queuing times for global jobs. *Rand* obtains the best performance using 2 duplications, while *Hybrid-1* is able to achieve similar performance, i.e., 3.6% in Figure 3. The strong bursty condition favors *Hybrid-1* over *Hybrid-2*, though *Rand* outperforms them all. For *Rand*, *Hybrid-1* and *Hybrid-2*, 7-duplication or flooding actually degrades the overall performance, due to the system penalty of 1s on each duplication invalidation.

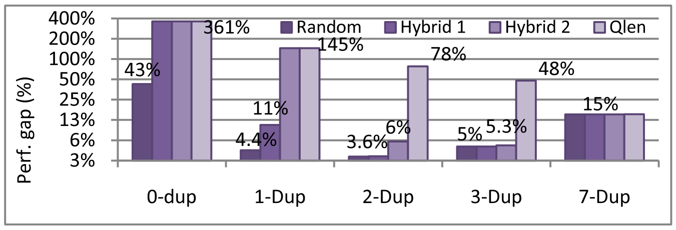

Fig. 3. Performance gap compared to ideal scheduling (*Act. QT* and 0-delay) in the strong bursty condition.

Figure 4 shows the composition of jobs in a series of queuing time ranges. Under the bursty workload condition, more than 70% of jobs are either queued for more than 100s or less than 10s. Using different duplications, the increase in the fraction of jobs with short queuing times (i.e. < 10s) is correlated with the decrease in the fraction of jobs with long queuing times (i.e. ≥ 100s). *Hybrid-2* reaches its top performance using 2 duplications, as indicated in the right bottom plot of Figure 4, while *Hybrid-1* only needs 1 duplication, as shown in the left bottom plot of Figure 4. Both *duplications* and *randomness* enable our new schedulers to counteract the effects of burstiness on system performance. We observe that during bursty periods, the *Hybrid* schedulers select the top first or second ranked sites based on out-of-date load information, but they randomly select sites for the remaining duplications, which become more likely to obtain the required resource first and balance the load surges across all the computing sites.

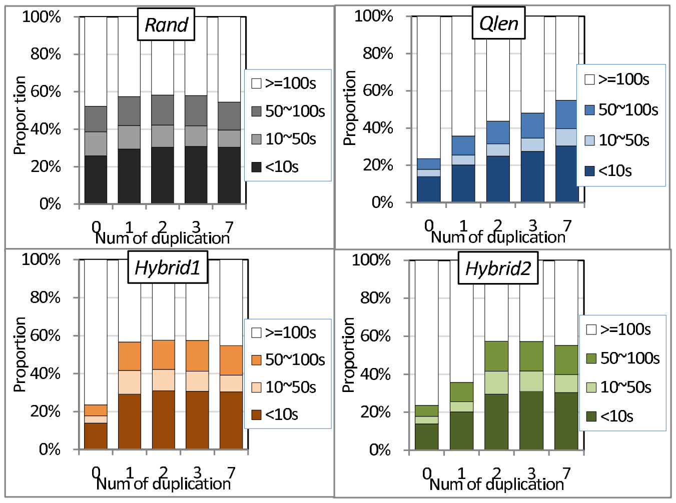

Fig. 4. Proportions of queuing times of global jobs obtained under the strong bursty condition for the *Rand*, *Qlen*, *Hybrid-1*, and the *Hybrid-2* scheduler.

C. Performance in the Weak bursty condition

In the weak bursty condition, the change in performance with the number of duplications is similar to that in the strong bursty condition. *Hybrid-1* and *Hybrid-2* can outperform *Rand* when using 1 and 2 duplications, respectively, as shown in Figure 5. Under a moderate intensity of burstiness, applications can use more duplications to improve performance, while *Hybrid-2* can deliver a performance edge over *Hybrid-1*.

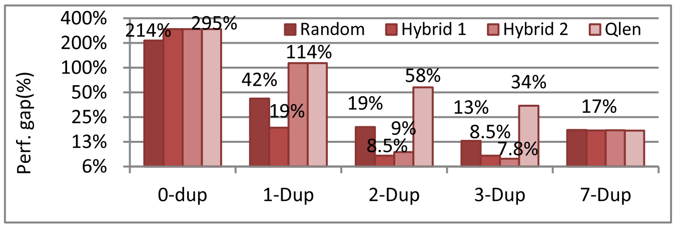

Fig. 5. Performance gap compared to ideal scheduling (*Act. QT* and 0-delay) in the weak bursty condition.

D. Performance in the non-bursty condition

We also applied the four schedulers to the environment without burstiness. The non-bursty workload is created by using exponential interarrival times for global jobs with the mean of 4s. Without the presence of burstiness, *Qlen* outperforms *Hybrid-1* and *Hybrid-2* only by small margins. *Hybrid-2* behaves the same way as *Qlen*, unless 2 or more duplications are used. Although duplication improves the performance of *Rand* by more than 7 times (using 3 duplications), we still observe a substantial under performance for the *Rand* scheduler. This experiment result suggests that our two new hybrid methods perform well, and the *Rand* scheduler should be avoided, when the workload is not bursty.

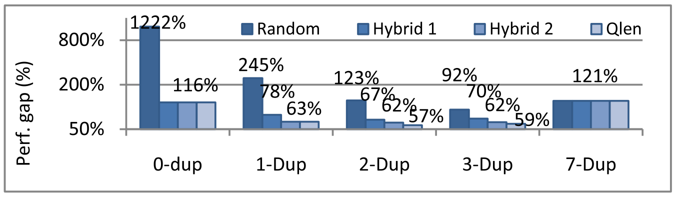

Fig. 6. Performance gap compared to ideal scheduling (*Act. QT* and 0-delay) in the non-bursty condition (exponential interarrival times).

In summary, using 2 or 3 duplications can allow the hybrid schedulers to obtain the shortest queuing time for most global jobs. However, in strong bursty conditions, less duplication is recommended. The two hybrid schedulers can match the performance of *Rand* in bursty conditions, while providing an equivalent performance to *Qlen* in the non-bursty condition. Comparing the two schedulers, *Hybrid-1* has the advantage if strong burstiness is expected. In weak and non-bursty conditions, *Hybrid-2* is preferable.

IV. Related work

The classic job scheduling algorithms, such as Min-Min and Max-Min [9], targeting the problem of scheduling of a bag tasks, assume that the execution times of all jobs are known. Meta-schedulers, like Condor-G [6], rely on accurate queue time prediction in scheduling jobs on computing grids. However, accurate predictions become more difficult in bursty conditions. Other researchers use advance-reservation and job preemption techniques to allocate resources on grids [16]. However, extra system overheads and job queue disruptions could decrease the overall system throughput. Some studies use resource brokers [17] or auction-based scheduling [2] methods on grids, which rely on the centralized scheduler and require host RMSs to comply with additional queuing policies and support complex management functions.

Burstiness has been studied extensively in many fields, particularly, in network communication. However, less work has investigated bursty workloads for computational platforms. To the best of authors' knowledge, this study is among the first to address the performance impact of burstiness on decentralized schedulers in computing grid environments.

V. Conclusion

In this paper, we have demonstrated that bursty job arrivals can have a significantly negative effect on the performance of a decentralized scheduler. We introduced two new hybrid schedulers based on duplication-invalidation to address job scheduling in bursty conditions. The main advantage of these new schedulers is that using both *randomness* and *duplication* mechanisms, they achieve robustness and effectiveness under varying levels of burstiness, yet do not rely on burst detection or coordination of job scheduling.

A simulation environment is used to evaluate the performance of our proposed schedulers under various intensities of burstiness. The simulation results validate our design by showing that compared to the ideal scheduling results under the bursty condition, the new schedulers only lead to a $3.6\% - 6\%$ performance degradation, whereas such a performance gap increases to $48\% - 78\%$ for the conventional SGR-supported schedulers. In the non-bursty condition, our proposed schedulers can provide similar performance to the SGR-supported schedulers. Therefore, we show the robustness of our new decentralized schedulers, which excel in both bursty and non-bursty conditions, as well as in mixed bursty non-bursty and conditions. In the future, we will refine our schedulers for heterogenous computing systems, where computing sites can vary in computing ability and capacity. We will develop new schedulers which can adaptively adjust the number of duplications based on the levels of detected burstiness, and consider variations in the global job submission delay which can be caused by network congesting and system overloading.

References

[1] CSIM19 development toolkit for simulation and modeling. http://www.mesquite.com, 2005.

[2] B.N. Chun and D.E. Culler. User-centric performance analysis of market-based cluster batch schedulers. *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 30–30, may 2002.

[3] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Trans. Netw.*, 4(2):209–223, 1996.

[4] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Appl.*, 11(2), 1997.

[5] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[6] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002.

[7] M. Grossglauser and J. Bolot. On the relevance of long-range dependence in network traffic. In *In Proc. of SIGCOMM*, pages 15–24, 1996.

[8] R. Gusella. Characterizing the variability of arrival processes with indexes of dispersion. *IEEE JSAC*, 19(2):203–211, 1991.

[9] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *J. of the ACM*, 24(2), 1977.

[10] H. Li and M. Muskulus. Analysis and modeling of job arrivals in a production grid. *ACM SIGMETRICS Perform. Eval. Rev.*, 34(4):59–70, 2007.

[11] N. Mi, G. Casale, and E.Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *ACM/IFIP/USENIX 9th Intl. Middleware Conf. (Middleware'08)*, pages 265–286, 2008.

[12] N. Mi, Q. Zhang, A. Riska, E. Smirni, and E. Riedel. Performance impacts of autocorrelated flows in multi-tiered systems. In *Perf. Eval.*, volume 64(9-12), 2007.

[13] A.W. Mu'alem and D.G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):529–543, 2001.

[14] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications*. Marcel Dekker, New York, 1989.

[15] A. Riska and E. Riedel. Long-range dependence at the disk drive level. In *QEST 2006 Conference*, pages 41–50. IEEE Press, 2006.

[16] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proc. of the IPDPS Conference*, pages 127 – 132, 2000.

[17] P. Xavier, W.T. Cai, and B.S. Lee. Workload management of cooperatively federated computing clusters. *J. Supercomput.*, 36(3):309–322, 2006.

[18] J. Zhang. *Adaptive Grid Computing*. PhD thesis, Northeastern University, 2010.

[19] J. Zhang and W. Meleis. Adaptive grid computing for MPI applications. In *Proc. of Parallel and Distributed Computing and Systems*, 2009.