Fault-Tolerant Scheduling for Bag-of-Tasks Grid Applications*

Cosimo Anglano and Massimo Canonico

Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria, Italy
{cosimo.anglano, massimo.canonico}@unipmn.it

Abstract. In this paper we propose a fault-tolerant scheduler for Bag-of-Tasks Grid applications, called *WorkQueue with Replication Fault Tolerant* (WQR-FT), obtained by adding checkpointing and replication to the *WorkQueue with Replication* (WQR) scheduling algorithm. By using discrete-event simulation, we show that WQR-FT not only ensures the successful completion of all the tasks in a bag, but also achieves performance better than WQR and other fault-tolerant schedulers obtained by coupling WQR with replication only, or with checkpointing only.

1 Introduction

Grid Computing technology provides resource sharing and resource virtualization to end-users, allowing for computational resources to be accessed as a utility. By dynamically coupling computing, networking, storage, and software resources, Grid technology enables the construction of virtual computing platforms capable of delivering unprecedented levels of performance. However, in order to take advantage of Grid environments, suitable application-specific scheduling strategies, able to select, for a given application, the set of resources that maximize its performance, must be devised [2]. The inherent wide distribution, heterogeneity, and dynamism of Grid environments makes them better suited to the execution of loosely-coupled parallel applications, such as *Bag-of-Tasks* [11] (BoT) applications, rather than of tightly-coupled ones. Bag-of-Tasks applications (parallel applications whose tasks are completely independent from one another) are particularly able to exploit the computing power provided by Grids [6] and, despite their simplicity, are used in a variety of domains, such as parameter sweep, simulations, fractal calculations, computational biology, and computer imaging. Therefore, scheduling algorithms tailored to this class of applications have recently received the attention of the Grid community [3, 4, 6]. Although these algorithms enable BoT applications to achieve very good performance, they suffer from a common drawback, namely their reliance on the assumption that the resources in a Grid are perfectly reliable, i.e. that they will

* This work has been supported by the Italian MIUR under the project *Societá dell'Informazione, Sottoprogetto 3 - Grid Computing: Tecnologie abilitanti ed applicazioni per eScience*, L. 449/97, anno 1999.

P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 630–639, 2005.

never fail or become unavailable during the execution of a task. Unfortunately, in Grid environments faults occur with a frequency significantly higher than in traditional distributed systems, so this assumption is overly unrealistic. A Grid may indeed potentially encompass thousands of resources, services, and applications that need to interact in order for each of them to carry out its task. The extreme heterogeneity of these elements gives rise to many failure possibilities, including not only independent failures of each resource, but also those resulting from interactions among them. Moreover, resources may be disconnected from a Grid because of machine hardware and/or software failures or reboots, network misbehaviors, or process suspension/abortion in remote machines to prioritize local computations. Finally, configuration problems or middleware bugs may easily make an application fail even if the resources or services it uses remain available [9].

In order to hide the occurrence of faults, or the sudden unavailability of resources, fault-tolerance mechanisms (e.g., *replication* or *checkpointing-and-restart*) are usually employed. Although scheduling and fault tolerance have been traditionally considered independently from each other, there is a strong correlation between them. As a matter of fact, each time a fault-tolerance action must be performed, i.e. a replica must be created or a checkpointed job must be restarted, a scheduling decision must be taken in order to decide where these jobs must be run, and when their execution must start. A scheduling decision taken by considering only the needs of the faulty task may thus strongly adversely impact non-faulty jobs, and vice versa. Therefore, scheduling and fault tolerance should be jointly addressed in order to simultaneously achieve fault tolerance and satisfactory performance. *Fault-tolerant schedulers* [1, 12, 15] attempt to do so by integrating scheduling and fault management, in order to properly schedule both faulty and non-faulty tasks. However, to the best of our knowledge, no fault-tolerant scheduler for BoT applications has been proposed in the literature. This paper aims at filling this gap by proposing a novel fault-tolerant scheduler for BoT applications, that we call *WorkQueue with Replication/Fault-Tolerant* (WQR-FT). WQR-FT extends the *WorkQueue with Replication* (WQR) [6] algorithm by adding to it both task checkpointing and automatic restart of failed tasks. As shown by our results, the simultaneous adoption of these mechanisms, besides providing fault tolerance, allows WQR-FT to achieve performance better than WQR, WQR using automatic restart only, and WQR using checkpointing-and-restart only. The rest of the paper is organized as follows. In Section 2 we present the WRQ-FT scheduling algorithm, while in Section 3 we presents the performance results we obtained in our experiments. Finally, Section 4 concludes the paper and outlines future research work.

2 WQR-FT: A Fault-Tolerant Scheduler for BoT Applications

Scheduling applications on a Grid is a non trivial task, even for simple applications like those belonging to the BoT paradigm [6]. As a matter of fact, the

set of Grid resources may greatly vary over time (because of resource additions and/or removals), the performance a resource delivers may vary from an application to another (because of resource heterogeneity), and may fluctuate over time (because of resource contention caused by applications competing for the same resource). Achieving good performance in these situations usually requires the availability of good information about both the resources and the tasks, so that a proper scheduling plan can be devised. Unfortunately, the wide distribution of Grid resources makes obtaining this information very difficult, if not impossible, in many cases. Thus, the so-called *knowledge-free* schedulers, that do not base their decisions on information concerning the status of resources or the characteristics of applications, are particularly interesting.

WorkQueue with Replication is a knowledge-free scheduling algorithm that adds task replication to the classical *WorkQueue* scheduler. Our scheduler, WQR-FT, adds both automatic restart and checkpointing to WQR, and properly coordinates the scheduling of faulty and non-faulty tasks in order to simultaneously achieve fault-tolerance and good application performance.

2.1 The Standard WQR Scheduler

In the classical WorkQueue (WQ) scheduling algorithm, tasks in a bag are chosen in an arbitrary order and are sent to the processors as soon as they become available. WQR adds task replication to WQ in order to cope with task and host heterogeneity, as well as with dynamic variations of the available resource capacity due to the competing load caused by other Grid users. WQR works very similarly to WQ, in the sense that tasks are scheduled the same way. However, after the last task has been scheduled, WQR assigns replicas of already-running tasks to the processors that become free (in contrast, WQ leaves them idle). Tasks are replicated until a predefined *replication threshold* is reached. When a tasks replica terminates its execution, its other replicas are canceled. By replicating a task on several resources, WQR increases the probability of running one of the instances on a faster machine, thereby reducing task completion time. As shown in [6], WQR performance are equivalent to solutions that require full knowledge about the environment, at the expenses of consuming more CPU cycles.

2.2 The WorkQueue with Replication – Fault Tolerant Scheduler

In its original formulation, WQR does not do anything when a task fails. Consequently, it may happen that one or more tasks in a bag will not successfully complete their execution. In order to obtain fault tolerance, we add *automatic restart*, with the purpose of keeping the number of running replicas of each task above a predefined replication threshold R. In particular, when a replica of a task t dies and the number of running replicas of t falls below R, WQR-FT creates another replica of t that is scheduled as soon as a processor becomes available, but only if all the other tasks have at least one replica running. Automatic restart ensures that all the tasks in a bag are successfully completed even in face of resource failures. However, each time a new instance must be

started to replace a failed one, its computation must start from the beginning, thus wasting the work already done by the failed instance. In order to overcome this problem, WQR-FT uses *checkpointing*, that is the state of the computation of each running replica is periodically saved with a frequency set as indicated in [13] (we postulate the existence of a reliable checkpoint server where checkpoints are stored). In this way, the execution of a new instance of a failed task may start from the latest available checkpoint. In this paper we assume that an application may be restarted from its latest checkpoint only on a machine having the same operating system of that on which the checkpoint was taken. If such a machine cannot be found when a new replica is created (to replace a faulty one), its execution will start from the beginning.

3 Performance Analysis

In order to assess the performance of WQR-FT, we compared it with both plain WQR, WQR using automatic restart only (henceforth referred to as *WQR-R*), and WQR using checkpointing only for various values of the replication threshold and checkpoint overhead, as well as for a variety of different workloads. In order to perform this comparison, we developed a discrete-event simulator, based on the CSIM [10] libraries, that has been used to perform a large number of experiments for a variety of operational scenarios. Our comparison is based on the following two metrics:

- Average *task response time*, defined as the time elapsing between the submission of the bag to which the task belong and the successful completion of its *first instance*;
- Average *BoT completion time*, defined as the time elapsing between the submission of the bag and the termination (either successful or not) of all its tasks.

3.1 Simulation Scenarios

Because the space covering the relevant parameters of our study is too large to explore exhaustively, we fix a number of system characteristics. The configuration of the Grid used in all the experiments has been obtained by means of GridG [5], that was used to generate a configuration comprising 100 clusters, each one comprising a random number of machines uniformly distributed between 1 and 16 (for a total number $N = 185$ of machines). The tasks submitted to each cluster are scheduled according to the FIFO policy. Individual machines do not run local jobs and do not support time-sharing. Each machine is characterized by its computing power P, an integer parameter whose value is directly proportional to its speed (i.e., a machine with $P = 2$ is twice faster than a machine with $P = 1$), that we assume to be uniformly distributed between 1 and 20. Moreover, each machine is associated with its operating system type, that is chosen with equal probability in the set $\{Solaris, Linux, FreeBSD\}$. A machine may

fail to complete the execution of its running task either because of a reboot (in 80% of the cases) or a crash (in 20% of the cases). The *Fault Time* (i.e., the time elapsing between two consecutive failures) is assumed to be a random variable with a Weibull distribution (in according with the results reported in [7]), while the *Repair Time* (i.e., the time elapsing from the fault to when the machine is operational again) is assumed to be uniformly distributed between 120 and 600 seconds (for a reboot) [8], or exponentially distributed with mean 2 days (for a hardware crash) [14]. The parameters of the Weibull distribution characterizing Fault Time were set according to the following procedure. We considered various scenarios in which the availability α of machines, defined as $\alpha = \frac{MTBF}{MTBF+MTTR}$ (where $MTBF$ and $MTTR$ denote the mean of the Fault Time and of the Repair Time, respectively), was set to $90\%, 50\%, 25\%$ and 10%, respectively. For a particular scenario, given the corresponding value of α and of the $MTTR$ (that in all our experiments was set to 34848 sec., according to the values reported before for the distributions of the Repair Time), we computed the $MTBF$ from the definition of α and, from this value, we obtained the parameters of the Weibull distribution.

3.2 The Workloads

In order to make an exhaustive comparison, we considered a rather large set of workloads, obtained by varying some parameters of a *base workload*. The base workload consists in a sequence of Bag of Tasks, each one comprising $RR \cdot N$ tasks, where N is the number of machines in the Grid ($N = 185$), and RR is a numerical coefficient used to change the size of each bag. The duration of each task is assumed to be a random variable uniformly distributed in the interval $[RT{*}0.5{*}BaseTime, RT{*}1.5{*}BaseTime]$ seconds, where $BaseTime$ and RT are workload parameters (note that the duration of tasks is referred to a machine with computing power $P = 1$). By suitably setting RR, RT, and $BaseTime$, very different workloads may be generated. In particular, in our study RR took values in the set $\{0.25, 0.5, 1, 1.25, 1.5\}$, RT in the set $\{0.5, 1, 1.5, 2, 3, 5\}$, and $BaseTime$ in the set $\{35000, 100000\}$, thus generating 70 different workloads.

3.3 Results

Let us now describe the results we obtained in our experiments. We first show that simple replication, as the one used by WQR, cannot guarantee the completion of all the tasks in a bag, except in scenarios characterized by high machine availability and by the use of a relatively high number of task replicas. Then, we will show that WQR-R outperforms WQR for the same number of replicas, and that WQR-FT outperforms both strategies for all the simulation scenarios we considered. In all our experiments, we computed 98% confidence intervals with a relative error of 2.5% or less for both the average task response time and the average BoT completion time.

Is Replication Sufficient? In order to verify whether the addition of further fault tolerance mechanisms to WQR is motivated by a real need, we per-

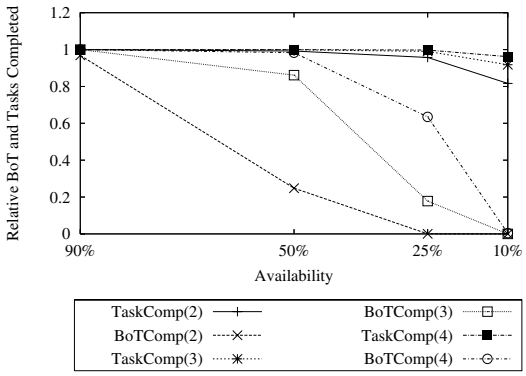

Fig. 1. Fraction of tasks and of BoTs successfully completed by WQR

formed a set of experiments in which we progressively decreased the availability of machines, and computed the fraction of tasks ($TaskComp(x)$) and of BoTs ($BoTComp(x)$) completed for various values of the replication threshold x. Our results, shown in Fig. 1 for a workload in which $RR = RT = 1$ (for the other workloads the results were not significantly different), clearly indicate that 100% of tasks (and, consequently, bag of tasks) is completed only with 4 replicas and for availability values greater than 90%. In all the other cases, the fraction of both completed tasks and bags decreases for decreasing availability values. This is due to the fact that the lower the availability, the higher the frequency of faults of each resource, and the probability of having all the instances of a task fail. Therefore, WQR without additional fault-tolerance mechanisms cannot ensure the successful completion of all the tasks, except in cases when machine availability is quite high, and a relatively large number of replicas is used.

WQR-R vs. WQR: The Benefits of Automatic Restart. Let us now compare the performance of WQR with those attained by WQR-R. Let us start with the results obtained, for various replication threshold values, for a workload in which $RR = 1$ (that is, each bag contains as many tasks as the number of Grid resources) and $RT = 1$. Fig. 2(a) and (b), where the number between parentheses indicate the value of the replication threshold, show respectively the average task response time and BoT completion time obtained by WQR and WQR-R for decreasing values of availability α and for $BaseTime = 35000$ sec. (the results for $BaseTime = 100000$ sec. are not significantly different, so are not reported here because of space constraints). As can be observed from the above figure, for availability values from 90% down to 25%, the average task response time (Fig. 2(a)) is practically not affected neither by the presence of automatic resubmission nor by the number of replicas, while a small difference in favor of WQR appears for $\alpha = 10\%$. This means that the presence of additional replicas created to replace faulty ones does not significantly increase the time

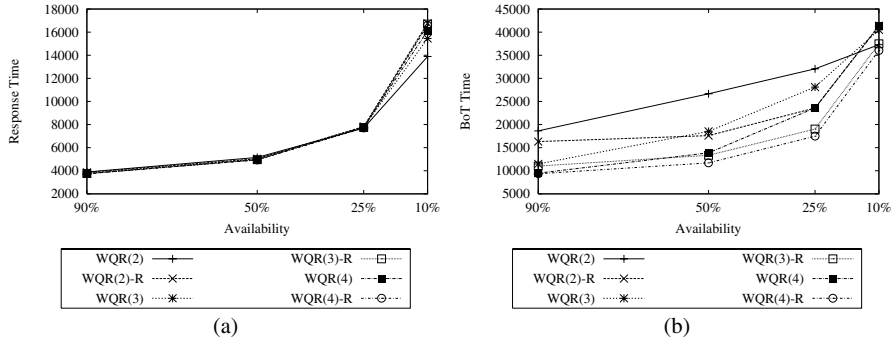

Fig. 2. (a) Average task response time, and (b) Average BoT completion time

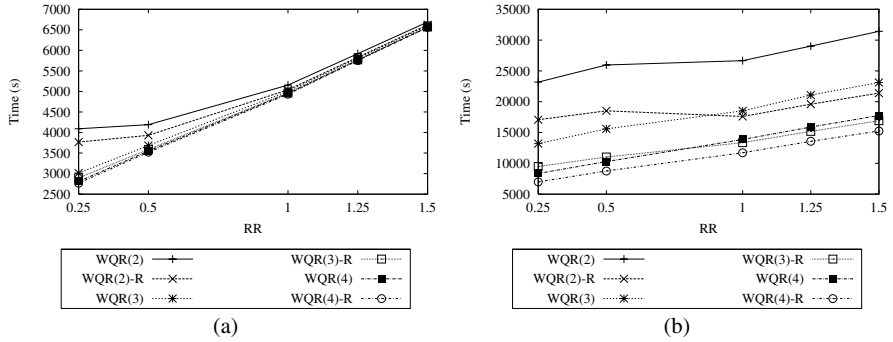

Fig. 3. (a) Average task response time, and (b) Average BoT completion time, $\alpha = 0.5$, RT=1

each tasks spends in the queue waiting to be scheduled. However, WQR-R results in a better average BoT completion time (Fig. 2(b)) than WQR for the same number of replicas. This is due to the fact that, when a replica fails, the one that replaces it may be scheduled on a faster machine, thus reducing the task execution time and, hence, the time taken to complete a bag. In order to validate this hypothesis, we performed a set of experiments in which we set $\alpha = 50\%$ and $RT = 1$, while we increased the size of the BoTs from 50% to 150% of the number of Grid resources (by varying RR accordingly). Our results show comparable task response times (Fig. 3(a)), but significants gains in terms of BoT completion time (Fig. 3(b)), especially for lower values of RR. This is due to the fact that the lower RR, the higher the number of machines on which a restarted replica may be executed, and consequently the higher the probability of choosing, for this replica, a faster machine. Therefore, our intuition is confirmed by these results.

WQR-FT vs. WQR-R: The Benefits of Checkpointing. As discussed in Sec. 2.2, the rationale behind the development of WQR-FT is to exploit the

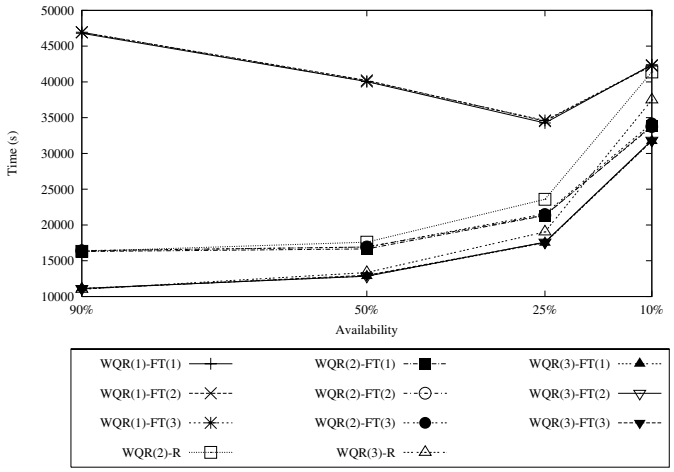

Fig. 4. Average BoT time

work already done by failed task replicas, so that better application performance may be obtained. In order to verify whether the introduction of checkpointing is beneficial or not, we performed a set of experiments in which we compared the performance of WQR-FT and WQR-R for various values of the checkpoint overhead (i.e., the time that each task is blocked to carry out a checkpoint operation) and of the replication threshold. More specifically, we run three sets of experiments in which the checkpoint overhead was assumed to be uniformly distributed between 0.1 and 0.3 sec. (*Set 1*), 6 and 10 sec. (*Set 2*), and 11 and 20 sec. (*Set 3*), respectively. In Fig. 4, where the first number between parentheses denotes the replication threshold, and the second one the experiment set (i.e., 1, 2, and 3 correspond to *Set 1, Set 2*, and *Set 3*, respectively), we report the results corresponding to the workload obtained by setting $RR = 1$, $RT = 1$, and $BaseTime = 35000$ sec., for different values of the availability α. As can be seen from this figure, the curves form three rather distinct groups, each one corresponding to a different replication threshold. The lower, intermediate, and upper group corresponds to scheduling strategies using 3 instances, 2 instances, and 1 instance for each task, respectively. As can be observed from these curves, the larger the number of replicas, the better the performance (the performance for larger values of the replication threshold show the same behavior, and have not included in the graph for the sake of readability). Within a given group is very hard to distinguish the curves corresponding to the various scheduling algorithms for availability values from 90% down to 50%, and this means that there is practically no difference among them. However, for lower availability values, checkpoint proves to be beneficial in terms of performance, and obviously the smaller the checkpointing overhead, the larger the performance gains. This is due to the fact that the higher the fault frequency, the higher the probability

that a task has already performed a large fraction of his work before failing. In these cases, the use of checkpointing is crucial to avoid wasting a large amount of already performed work. This effect is even more evident for workloads where $BaseTime = 100000$ s. (not shown here because of space constraints), since in these cases the amount of work wasted if checkpointing is not used is even larger. It is worth to point out that checkpointing alone is not sufficient to ensure good performance, as indicated by the upper set of curves shown in Fig. 4. These curves indeed show that simple checkpointing (that corresponds to WQR(1)-FT, i.e. to the use of a single task instance) attains the worst performance among all the considered alternatives. The graph concerning instead the task response time (not shown here because of space constraints), shows no difference – for a given replication threshold – among all the strategies, meaning that checkpointing does not enlarge the time spent by each tasks waiting to be scheduled. Therefore, we can conclude that, being the availability of machines usually unknown, it is better to use checkpointing rather than not, as this may result in much better performance when machine availability is low.

The results obtained for the other workloads and scenarios, not reported in this paper because of the limited amount of available space, confirm the above conclusions. In particular, we observed that the larger the execution time of tasks (i.e. the larger the values of RT and $BaseTime$), the larger the performance gains due to checkpointing. This is due to the fact that the larger the task execution time, the larger the performance loss occurring when a task replica has to restart its execution from the beginning. Moreover, the larger the value of RR, that is the number of tasks in a bag, the better the performance of checkpointing. This can be explained by the fact that, the larger RR, the lower the probability for WQR-R of finding soon an available machine faster than the one that failed. Conversely, in these situations the possibility of exploiting the work already done given by the use of checkpointing may significantly enhance performance.

4 Conclusions and Future Work

In this paper we have presented WQR-FT, a fault-tolerant scheduling algorithm for Bag-of-Tasks Grid applications based on the WQR algorithm. By simultaneously using replication and checkpointing, WQR-FT is able not only to guarantee the completion of all the tasks in a bag, but also to achieve performance better than alternative scheduling strategies. As a matter of fact, being WQR able to attain performance higher than other (non fault-tolerant) alternative strategies [6], and being WQR-FT to achieve performance better than WQR, we can conclude that WQR-FT outperforms these strategies when resource failures and/or unavailabilities are taken into account.

As future work, we plan to study the behavior of WQR-FT in more complex scenarios, such as those in which background computation load is present on the resources in the Grid, and multiple WQR-FT instances are present in the same Grid. Moreover, we plan to investigate possible extensions to WQR-FT, in order

to make it able to deal with applications that require data transfers in order to carry out their computation, and with the corresponding failure models.

References

1. J.H. Abawajy. Fault-Tolerant Scheduling Policy for Grid Computing Systems. In *Proc. of 18th Int. Parallel and Distributed Processing Symposium, Workshop on*. IEEE-CS Press, April 2004.
2. F. Berman and R. Wolski et al. Adaptive Computing on the Grid Using AppLeS. *IEEE Trans. on Parallel and Distributed Systems*, 14(4), April 2004.
3. H. Casanova, F. Berman, G. Obertelli, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proc. of Supercomputing 2000*. IEEE CS Press, 2000.
4. H. Casanova, A. Legrand, and D. Zagorodnov et al. Heuristics for Scheduling Parameter Sweeping Application in Grif Environments. In *Proc. of Heterogeneous Computing Workshop*. IEEE CS Press, 2000.
5. P. Dinda D. Lu. GridG: Generating Realistic Computational Grids. *Performance Evaluation Review*, 30, 2003.
6. D.P. da Silva, W. Cirne, and F.V. Brasileiro. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In *Proc. of EuroPar 2003*, volume 2790 of *Lecture Notes in Computer Science*, 2003.
7. J. Brevik, D. Nurmi, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. Technical Report 37, Department of Computer Science, University of California, Santa Barbara, 2003.
8. J. Brevik, D. Nurmi, and R. Wolski. Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-peer Systems. In *Proc. of 4^{t}h Int. Workshop on Global and Peer-to-Peer Computing*, Chicago, Illinois (USA), April 19-22, 2004. IEEE Press.
9. R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauvé. Fault in Grids: Why are they so bad and What can be done about it? In *Proc. 4th Int. Workshop on Grid Computing (Grid 2003)*. IEEE-CS Press, Nov. 2003.
10. H. Schwetman. Object-oriented simulation modeling with c++/csim. In *Proc. of 1995 Winter Simulation Conference*, Dec. 1995.
11. W. Cirne, et al. Grid Computing for Bag of Tasks Applications. In *Proc. of 3rd IFIP Conf. on E-Commerce, E-Business and E-Government*, Sao Paulo, Brazil, Sept. 2003.
12. J. Weissman and D. Womack. Fault Tolerant Scheduling in Distributed Networks. Technical Report TR CS-96-10, Department of Computer Science, University of Texas, San Antonio, Sept. 1996.
13. J.W. Young. A First-order Approximation to the Optimum Checkpoint. *Communications of the ACM*, 17, 1974.
14. S. Hwang, C. Kesselman. A Flexible Framework for Fault Tolerance in the Grid. *Journal of Grid Computing*, 1(3), 2003.
15. X. Zhang, D. Zagorodnov, M. Hiltunen, K. Marzullo, and R.D. Schlichting. Fault-tolerant Grid Services Using Primary-Backup: Feasibility and Performance. In *Proc. IEEE Int. Conf. on Cluster Computing*. IEEE-CS Press, Sep. 2004.