Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids

Cosimo Anglano [1], John Brevik [2], Massimo Canonico [1], Dan Nurmi [2], Rich Wolski [2]
[1] Dipartimento di Informatica, Universita' del Piemonte Orientale (Italy),
email:{cosimo.anglano,massimo.canonico}@unipmn.it
[2] Department of Computer Science, University of California, Santa Barbara,
email:{jbrevik,nurmi,rich}@cs.ucsb.edu

***Abstract*— Desktop Grids have proved to be a suitable platform for the execution of Bag-of-Tasks applications but, being characterized by a high resource volatility, require the availability of scheduling techniques able to effectively deal with resource failures and/or unplanned periods of unavailability. In this paper we present a set of *fault-aware* scheduling policies that, rather than just tolerating faults as done by traditional fault-tolerant schedulers, exploit the information concerning resource availability to improve application performance. The performance of these strategies have been compared via simulation with those attained by traditional fault-tolerant schedulers. Our results, obtained by consider a set of realistic scenarios modeled after real Desktop Grids, show that our approach results in better application performance and resource utilization.**

I. Introduction

Grid Computing can be defined as the coordinated resource sharing and problem solving in dynamic, multi-enterprise collaborations [1], and typically involves using many resources (computer, data, I/O, instruments, etc.) to solve a single, large problem that could not be solved on any one resource. Recently, the exploding popularity of the Internet has created a new much large scale opportunity for Grid computing. As a matter of fact, millions of desktop PCs, whose idle cycles can be exploited to run Grid applications, are connected to wide-area networks both in the enterprise and in the home. These new platforms for high throughput applications are called *Desktop Grids* [2]. The inherent wide distribution, heterogeneity, and dynamism of Desktop Grids makes them better suited to the execution of loosely-coupled parallel applications rather than tightly-coupled ones. *Bag-of-Tasks* applications [3](BoT) (parallel applications whose tasks are completely independent from one another) have been shown to be particularly able to exploit the computing power provided by Desktop Grids [4] and, despite their simplicity, are used in a variety of domains, such as parameter sweeps, simulations, fractal calculations, computational biology, and computer imaging. In order to take advantage of Desktop Grid environments, suitable scheduling strategies, tailored to BoT applications, have been proposed in the literature. These algorithms can be broadly classified into *knowledge-based* [5], [6], [7], where the scheduler relies to variable extents on information concerning the application and/or the resources of the Grid, and *knowledge-free* [4], that do not require such information. Unfortunately, all these algorithms are based on the unrealistic assumption that resources are always available (i.e., they never crash or they are never taken off-line). However, as shown in [8], [9], Desktop Grid resources are characterized by a very high resource volatility, since resources can be disconnected from the Grid at any time without any advance notice. Consequently, these scheduling algorithms yield suboptimal performance when applied to realistic scenarios where failures actually occur more often than not.

In order to cope with resource failures, recent work has focused on *fault-tolerant scheduling* [10], [11], [12], where the scheduler relies on fault-handling mechanisms (such as task replication and checkpoint-and-restart) to deal with the occurrence of resource failures or unavailability. These schedulers, however, being knowledge-free, do not use any information concerning the tasks and the resources, with the consequence that their resource usage is suboptimal.

In this paper we propose an alternative approach to BoT scheduling in Desktop Grids (named *fault-aware scheduling*) that, instead of just tolerating resource failures, tries to avoid them as much as possible by jointly exploiting the fault-handling mechanisms, and the knowledge of the effective computing power delivered by resources [13], [14] and the distributions of their fault times [15] (i.e. the time elapsing between two consecutive faults), to improve scheduling performance. We show how this information can be exploited to improve both *task selection* (the choice of the next task to be executed) and *machine selection* (the choice of the machine on which it will be executed) with respect to fault-tolerant schedulers. More specifically, we propose two task selection and four machine selection policies that, when combined, give rise to 8 scheduling policies. In order to show the effectiveness of our approach, we have conducted a thorough simulation study that has shown that our strategies outperform fault-tolerant schedulers for a variety of operational scenarios.

The rest of the paper is organized as follows. In Section II we place our work in the context of the related literature. In Section III we discuss our fault-aware scheduling policies, while in Section IV we present the results obtained in our experiments. Finally, Section V concludes the paper and outlines future research work.

II. Related work

Existing algorithms for scheduling BoT applications on Desktop Grids can be classified along two dimensions, namely (a) their reliance on task/resource information (i.e., we have knowledge-free and knowledge-aware strategies), and (b) the way they handle resource failures (i.e., we have fault-agnostic and fault-aware strategies). Although this classification gives rise to four different combinations, the literature provides examples belonging to only three of them. *Knowledge-based, fault-agnostic* schedulers rely on resource/task information, but are based on the implicit assumption that resources never fail. Schedulers in this class assume the knowledge of the execution time of individual tasks, and exploit various type of static [2], [7] or dynamic [5], [6] resource information to perform machine selection. Our scheduling strategies also use dynamic resource information, but unlike the previous ones are able to efficiently handle resource failures or unavailabilities.

Knowledge-free, fault-agnostic schedulers [4] make blind task and machine selections. *Knowledge-free, fault-tolerant* schedulers [7], [10], [11], [12] improve over their knowledge-free counterparts by using task replication to reduce the effects of poor task assignments, and automatic restart (possibly coupled with checkpointing) to deal with resource failures. These strategies are able to obtain reasonable performance even in face of failures, but waste a great deal of CPU cycles to execute needless task replicas. Our strategies also exploit task replication and checkpoint-and-restart, but their usage of task and resource information allows them to greatly reduce the amount of wasted CPU cycles, that are instead used to improve application performance.

III. Fault-aware scheduling

In this section we describe our fault-aware scheduling policies, that are obtained by combining a task selection policy with a machine selection policy. We start by describing first the system model on which our policies are based, and we then proceed with the presentation of the various task selection and machine selection policies.

A. System model

In this paper, we consider Desktop Grids composed of a set of heterogeneous machines connected by a generic communication network. Each machine M is characterized by its *nominal computing power* $P(M)$, a real number whose value is directly proportional to its speed (i.e., a machine with P=2 is twice faster than a machine with P=1). We assume that the machines may also run local jobs (i.e., job started by their legitimate owners), so that at each time instant t it delivers to Grid applications an *effective computing power* $EffPwr(M,t)$ defined as:

$$EffPwr(M,t) = P(M) * AvailCPU(M,t) \quad (1)$$

where $AvailCPU(M,t)$ is a real value that represents the fraction of M's CPU capacity available to Grid processes (i.e., unused by local jobs) at time t.

```
while (there are unfinished task) do
  t=SelectTask();
  r=SelectMachine();
  assign(t,r);
  ReplicateTasks();
end while
```

Fig. 1. The scheduling algorithm

Furthermore, we assume that resources can become unavailable at unpredictable times because of hardware/network problems or simply because they are taken out from the Desktop Grid (we consider both events as resource failure). To cope with failures, we assume that the Desktop Grid encompasses one or more *Checkpoint Servers*, in charge of storing and providing access to checkpoints. We assume that some form of virtual machine is used to run Desktop Grid applications (e.g. as done in *Entropia* [16] or in *United Devices* [17]), so that a given checkpoint can be used on any machine, and that the Checkpoint Server stores only the checkpoint corresponding to the replica that is the closest one to complete its task. The *residual execution time* of to a task checkpoint, that is required to decide whether a given checkpoint must be stored or not by the Checkpoint Server, is computed as discussed in Section III-C.

B. Fault-aware scheduling policies

As mentioned before, the core of our scheduling policies are the task and machine selection policies, as schematically shown in the pseudo-code of Fig. 1.

The scheduler works by repeatedly selecting a task to execute (by means of function *SelectTask()*) and the machine on which it will be executed (by means of function *SelectMachine()*, until all the tasks of the bag have been completed. When a task fails (because of the crash or the unavailability of the machine it is using), it inserted again in the list of unfinished tasks. If all unfinished tasks are running, and there are idle resources, function *ReplicateTasks()* starts the execution of tasks replicas, in such a way that each task has a number of replicas lower than or equal to a pre-defined *replication threshold* (more details on the replication policy can be found in [12]). In this paper we consider a *static* replication policy, where the number of running replicas per task never exceeds the replication threshold. Dynamic replication policies, where the above threshold may be exceeded, are also possible, but we dot not consider them here since we have experimentally observed that they result in negligible performance gains, while they require a much more complex replica management.

In this paper we propose two task selection and four machine selection policies, whose combinations give rise to the 8 scheduling policies listed in Fig. 2 (where rows and columns correspond to task selection and machine selection policies, respectively), obtained by combining two task selection policies with four machine selection policies that are described in the next two subsections. These selection policies

	Blind	EffCPUKnown	FTDKnown	EffCPU+FDTKnown
SRET	SRET-Blind	SRET-EffCPU	SRET-FTD	SRET-EffCPU-FTD
LRET	LRET-Blind	LRET-EffCPU	LRET-FTD	LRET-EffCPU-FTD

Fig. 2. Scheduling policies

are based on the knowledge of the execution time of each task T_i on a reference machine with nominal computing power $P = 1$, and on the availability of estimates of the effective computing power delivered by resources, that can be computed as discussed in [14], and of their fault time distributions, that can be performed as discussed in [15], [9].

C. Task selection policies

The task selection policies we propose are based on the notion of the *residual execution time* of tasks, i.e. the time needed to complete a task starting from its last saved checkpoint (or from its beginning if a checkpoint does not exist). A checkpoint taken after executing a task for an interval lasting C time units on machine M decreases its residual execution time by $C \cdot \int_0^C EffPwr(M,t)dt$, where the integral gives the average effective computing power delivered by the machine during that interval. For example, if a task executes for 25 time units and the average delivered computing power in that interval has been 3, the residual execution time is reduced by $25 \cdot 3 = 75$ time units on the reference machine. This information is stored with the checkpoint, and used by the Checkpoint Server to decide when a given checkpoint should replace the saved one or not.

In this paper we consider two task selection policies that exploit the residual execution time of tasks, namely:

- *Shortest Residual Execution Time* (*SRET*): The scheduler chooses the task with the shortest residual execution time whose number of running replicas is lower than the replication threshold. Intuitively, *SRET* attempts to complete the shortest tasks first, in order to have enough free machines to execute replicas of the longest tasks;
- *Longest Residual Execution Time* (*LRET*): The scheduler selects the task with longest residual execution time whose number of running replicas is lower than the replication threshold. Intuitively, *LRET* is a form of critical path scheduling. As a matter of fact, a Bag-of-Tasks can be seen as a very simple DAG, where all the tasks are spawned from an initial dummy node, and there is a dummy final node that must wait the completion of all the tasks in a Bag. By giving precedence to longest tasks, *LRET* seeks to complete the tasks in the critical path as soon as possible, in the attempt to reduce the makespan a BoT.

D. Machine selection policies

The machine selection policies proposed in this paper are based either on estimations of the effective CPU power, of the fault time distribution of machines, or both. More specifically, we propose the following three machine selection policies (in addition to the trivial policy – named *Blind* – that performs machine selection in a blind way):

- *EffCPUKnown*: This policy chooses the machine M_j with the highest predicted effective CPU power $PredCPU(M_j)$, in the attempt to minimize the execution time of the selected task. *EffCPUKnown* assumes that $PredCPU(M_j)$ represent the average CPU powered delivered by M_j during the execution of the task;
- *FTDKnown*: This policy uses estimates of the fault-time distribution (FTD) of individual machines, and uses them to select the machine with the longest *residual life time*, that is the machine whose next fault is the farthest ahead in time. The rationale behind FTDKnown is to avoid (or to delay as much as possible) the occurrence of a machine failure during the execution of a task, so that the number of task rollbacks are minimized. Delaying the occurrence of a fault also increases the probability of updating the checkpoint relative to the running task, thus reducing its residual execution time;
- *EffCPU+FTDKnown*: This policy exploits both types of information as follows. First, the execution time $ET(T_i, M_j)$ of the chosen task T_i is computed for each candidate machine M_j as $ET(T_i, M_j) = T_i/PredCPU(M_j)$. Then, for each candidate machine the probability $P(F, T_i, M_j) = P\{Fault_Time \geq ET(T_i, M_j)\}$ that its next fault occurs after T_i terminates its execution is computed, and only those machines such that $P(F, T_i, M_j) \geq 0.95$ are selected. Finally, if more than one machine has been selected, or if no machine satisfies the above inequality, the one with the highest $EffPwr()$ value is chosen.

IV. Performance analysis

In order to assess the performance of the proposed scheduling policies, we performed an exhaustive simulation study carried out by means a discrete-event simulator, in which we considered a large set of operational scenarios and compared all our strategies among them and also with WQR-FT [12], a knowledge-free fault-tolerant scheduler that, at the best of our knowledge, provides the best performance w.r.t. other alternative schedulers in the same family. In order to obtain realistic results, in our simulations we considered a set of realistic scenarios and workloads, obtained from the analysis of measurement traces taken on a real Desktop Grid. Our comparison has been carried out by using as metrics the average *BoT completion time* and the *relative CPU wasted time*. The BoT completion time is defined as the time elapsing between the submission of a bag and the termination of all its tasks, while the relative CPU wasted time is defined as the

ratio of the total CPU time wasted by replicas and the total amount of used CPU time, that is:

$$RelativeWasted = \frac{WastedTime}{WastedTime + UsefulTime}$$

The CPU time used to run a replica is considered wasted if the replica fails without completing the task or producing a checkpoint better than the stored one (i.e., that reduces the residual execution time of the corresponding task). In order to explain whether the CPU time used to run a replica is considered to be wasted or not, let us consider the scenarios depicted in Fig. 3, where we assume to have four running replicas (that is, R_0, R_1, R_2 and R_3) of a given task. Consider first the case (Fig. 3(a)) in which all these replicas terminate their execution (either successfully or not) before a checkpoint is taken. In particular, assume that R_1 and R_3 fail (the "x" symbol denotes a failure), that R_2 completes the task (the "o" symbol denotes task completion) and, consequently, that R_0 is killed (the "k" symbol denotes task killed). In this case, we consider as useful only the CPU time used to run R_2 (denoted as CT_2), since no checkpoint is produced by the other replicas. Therefore, the relative CPU wasted time is given by

$$RelativeWasted = \frac{CT_0 + CT_1 + CT_3}{CT_0 + CT_1 + CT_2 + CT_3}$$

Consider now a different scenario (Fig. 3(b)), where initially we have a single task replica running (R_0), that takes its first checkpoint at time t_1. After the checkpoint has been saved on the Checkpoint Server, a machine becomes available, so a new replica R_1 of the same task is created and, since a checkpoint is available, its computation is started from the checkpoint. Then, after some time, R_0 fails, but the task is eventually completed by R_1. In this case, the CPU wasted time is only the time elapsed between the checkpoint saving and the replica failure (that is, CT_1), since both CT_0 and CT_2 have contributed to the task completion. Therefore, the relative CPU wasted time is

$$RelativeWasted = \frac{CT_1}{CT_0 + CT_1 + CT_2}$$

A. Simulated scenarios

Our simulation study considered the two different Desktop Grid platforms, and the variety of workloads described in this section.

1) Simulated platforms: In our study, we considered two distinct simulation scenarios, corresponding respectively to an enterprise and to a public-resource computing Desktop Grid. The enterprise Grid represents scenarios in which machines are relatively homogeneous, and faults are relatively infrequent, while the public-resource one represents platforms where resources are scattered across many independent users, so are characterized by a large heterogeneity and frequent failures.

For the enterprise Desktop Grid we considered the 85 machines of the *Computer Science Instructional Laboratory* (CSIL) of the University of California, Santa Barbara (UCSB). These machines run the Linux operating system, are equipped with either an Intel Pentium or XEON processor with different clock rates (from 2.2GHz to 3GHz), and memory capacities (from 512MB to 1GB). The nominal computing powers of these machines have been computed by running the *nbench* [18] benchmark on each of them, and by normalizing them to the lowest measured value. These process resulted in 3 different values for the nominal power of machines, namely $\{1, 1.125, 1.4375\}$, that are uniformly distributed across the various machines.

As stated by Eq. 1, the effective computing power delivered by each machine at any time instant t depends also on $AvailCPU(t)$, that is a random variables whose values change over time. In our simulations, these changes have been described by the Markov model depicted in Fig. 4, where each state corresponds to a CPU load value and the label M_{xy} on the arcs corresponds to the probability of moving from state X to state Y. The parameters of this model have been

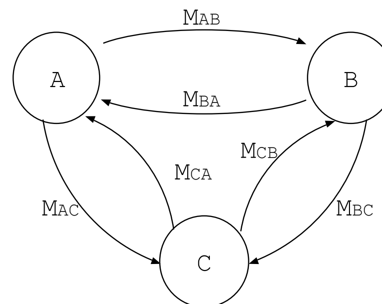

Fig. 4. Markov model describing $AvailCPU(t)$

computed by using the data collected, in a 6 months period spanning from January to July 2004, by the CPU load sensor of the *Network Weather Service* (NWS) [19] monitoring and forecasting system. In particular, as shown in Fig. 4, three different states – corresponding to the availability of 100% (A), 50% (B), and 33% (C) of CPU power – have been obtained from the analysis of the above data. The transition probabilities shown in Fig. 4 have been also estimated from the above data as

$$M_{xy} = \frac{Count(xy)}{Count(x)}$$

where $Count(xy)$ contains the number of transitions from state X to state Y, while $Count(x)$ contains the number of occurrences of state X. To model the variability over time of the effective CPU powered delivered by individual resources, our simulator computes a new value (by means of Eq. 1 every 10 seconds of simulated time. The actual values for the transition probabilities, that vary from one machine to another, have not been reported because of space constraints (the interested reader may refer to [20] for more details).

The fault time distribution of these machines have been also estimated from a set of data collected in the same time period by means of the Fault Time sensor of the NWS. As reported in [21], the actual distribution is best approximated by the Weibull function, whose parameters (shape and scale) have

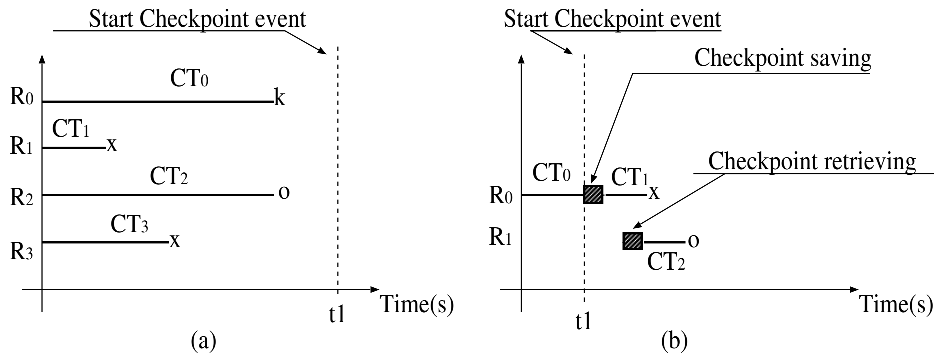

Fig. 3. Wasted and useful CPU time

been computed using the *Maximum Likelihood Estimation* (MLE) method [21]. The actual values for the Weibull parameters depend from the particular machine, and are not reported here because of space constraints (the interested reader may refer to [20] for more details). In all the experiments the repair time (i.e. the time required to bring a machine back to a functioning state) was instead set to 120 sec., the average value computed from the above collected data.

The public-resource Desktop Grid scenario has been obtained from the enterprise one as follows. First, in order to introduce a larger machine heterogeneity, the nominal computing power was assumed to be distributed according to a Gaussian distribution with mean and standard deviation equal to 10. Second, in order to represent a larger frequency of machine failures, we set the parameters of the Weibull distribution in such a way that the average fault time was 100 times smaller than the one measured for the enterprise Desktop Grid.

2) Simulated 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, and RR represents the average number of tasks for each Grid machine. For example, in our scenarios $N = 85$, so when $RR = 3$ the BoT comprise 255 tasks. The duration of each task is assumed to be a random variable uniformly distributed in the interval $[0.5 * BaseTime, 1.5 * BaseTime]$ seconds, where $BaseTime$ is a workload parameters that represents the mean execution time of a task submitted to a machine with computing power $P = 1$. By suitably setting RR and $BaseTime$, very different workloads may be generated. Due to space constraints, we present only the results obtained when RR took values in the set $\{3, 5, 7, 10, 20, 50\}$, and $BaseTime = 35000$, thus generating 6 different workloads. However, the results obtained for other values of the above parameters do not significantly vary from those reported in this paper.

During its computation, each replica performs its checkpoints with a frequency computed according to the Young's formula [22] that approximates the optimum checkpoint interval. In our study, we assume that the time taken to transfer a checkpoint file to the server is 480 sec. (the value measured for 500 MB files on the enterprise Desktop Grid network). Finally, we performed experiments for various values of the replication threshold (i.e. the maximum number of replicas that the scheduler attempts to keep running for each task), but we observed that using more than 2 replicas does not result in significant performance improvements. Therefore, all the experiments described in this section correspond to scenarios in which the replication threshold was set to 2.

B. Results

Let us describe now the results obtained for the two Desktop Grids discussed before. We performed a set of experiments in which we progressively increased RR, and compared the performance of our 8 scheduling policies among them and also with respect to the WQR-FT scheduling algorithm. In all our experiments, we computed 98% confidence intervals with a relative error of 2.5% or less for the average BoT completion time. For the sake of clarity of the figures, for each scenario we report only the results obtained by the best blind and informed scheduling policy.

1) The Enterprise Desktop Grid scenario: Fig. 5(a) reports the results concerning the average BoT time, that has been normalized w.r.t. WQR-FT, for values of RR ranging from 3 to 50. As can be seen by these results, the best scheduling strategies are those that use *LRET* as task selection policy, that obtain performance gains w.r.t. WQR-FT ranging from 10% (for $RR = 3$) to about 1% (for $RR = 50$). This confirms our intuition that starting the longest tasks first results in an overall smaller BoT completion time. The reduction of the relative performance benefits of *LRET* for increasing values of RR is explained by the fact that, when the number of tasks is much larger than the number of resources, knowledge-free schedulers like WQR-FT are close to being optimal [2]. Another observation stemming out from our results is that the machine selection policy has a negligible impact on performance (the performance attained by *LRET-FTD* and *LRET-EffCPU-FTD* are practically identical to those showed in the figure). This is due to the fact that, being all the resource similar and very reliable, the machine chosen for a given task

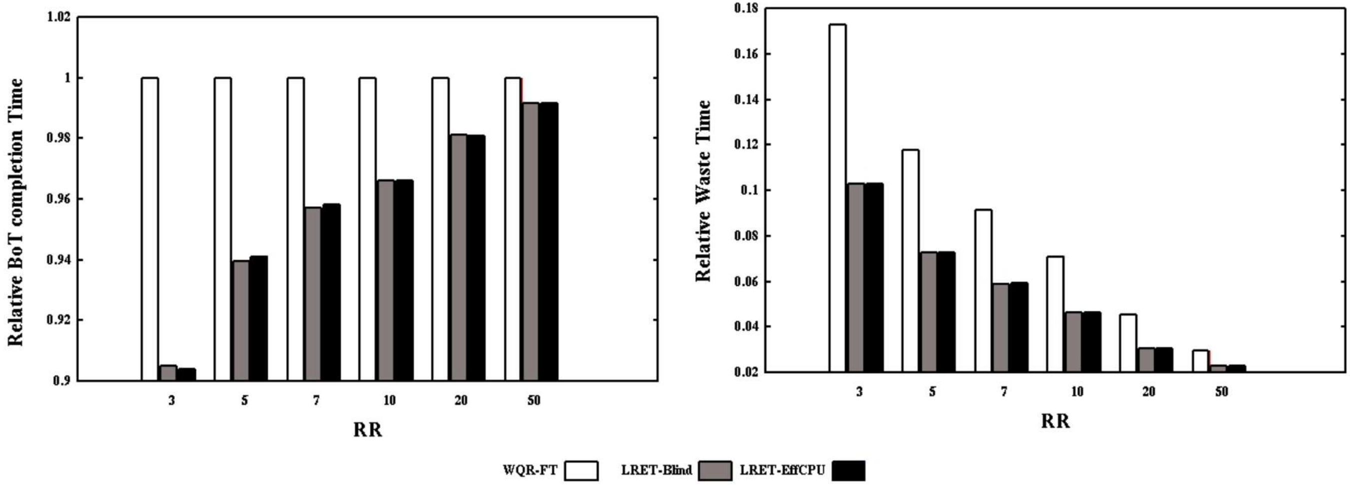

Fig. 5. Enterprise Desktop Grid: (a) Average BoT completion time, and (b) relative wasted time

by all scheduling policies is often the one that completes it (so the other replicas are useless). In the infrequent cases in which this machine fails, there is little advantage provided by the availability of a better checkpoint, since a new replica must wait 480 sec. (the time taken to transfer a checkpoint file) before starting its execution. However, as will be shown later, in scenarios characterized by higher heterogeneity and failure rates, performance gains tend to be more evident.

Similar results can be observed also for the amount of wasted CPU time, as shown in Fig. 5(b), where the gains of *LRET*-based strategies w.r.t. WQR-FT are evident. To understand why this happens, consider that at the beginning of the execution of a BoT very few replicas are created, since all the idle machines are used to run the first instance of each task. Conversely, when most of the tasks have been completed, the amount of replicas increases. With *WQR-FT* half of these replicas (on average) correspond to longer tasks, that are more prone to incur into a machine failure than shorter ones, with the consequence that more resubmissions are generated w.r.t. *LRET*-based strategies. It is interesting to observe also that for increasing values of RR all the strategies waste less CPU time. This depends on the fact that, when the number of tasks grows, the probability to have an idle resource decreases, so the scheduler cannot start many replicas.

2) The public-resource Desktop Grid scenario: The results obtained for this scenario, characterized by a higher resource heterogeneity and a lower availability, shows that – unlike the previous case – the machine selection policy actually makes a difference both in terms of performance (Fig. 6(a)) and wasted CPU time (Fig. 6(b)). First of all, our results indicate again that *LRET* is the best task selection policy both in terms of performance and efficiency so, as mentioned before, the graphs in the figures do not include the bars corresponding to *SRET*-based policies.

Let us start by discussing the results concerning the average BoT completion time, reported in Fig. 6(a) (showing the average BoT completion time normalized w.r.t WQR-FT), that indicate that the best informed machine selection policy is *EffCPU*, that results in a reduction in the BoT completion time ranging from from 18% for $RR = 3$ to 7% for $RR = 10$ and 3% for $RR = 50$. More precisely, *LRET-FTD* results in performance worse than *LRET-EffCPU*, but better than WQR-FT (from about 11% for $RR = 3$ to about 3% for $RR = 50$), thanks to the fact that, by choosing the most reliable machine the number of resubmissions decreases and consequently also the probability than one replica may be scheduled on a faster machine decreases. Conversely, the performance attained by *LRET-EffCPU-FTD* are practically identical, but *LRET-EffCPU* should be considered better since it requires a smaller amount of knowledge. Furthermore, we note that the *Blind* machine selection policy does not pay off, as *LRET-Blind* constantly results in performance worse than *LRET-EffCPU*.

Another interesting observation that can be made by comparing Fig. 6(a) with Fig. 5(a) is that even the *Blind* machine selection policy provides better performance than plain WQR-FT, as can be observed by the fact that the difference between these two scheduling policies remains larger for larger values of RR than in the Enterprise Desktop Grid scenario. This can be explained by the fact that the large number of resubmissions occurring in the public-resource Desktop Grid (due to the lower resource availability) increases the probability of submitting a task to a faster machine (since in this scenario the computational power of the resource varies significantly).

Let us discuss the results concerning the CPU wasted time, shown in Figure 6(b). From these results, the advantages of using the *LRET* task selection policy in conjunction with the *EffCPUKnown* machine selection policy clearly emerge. However, in general all the strategies are less efficient than in the Enterprise Desktop Grid scenario, as indicated by the fact that now the wasted CPU time ranges from 21% to 30%, while in the previous scenario it ranged between 2% and 18%. This is not a surprise, however, since the lower availability implies a larger number of task resubmissions, that in turn result in a

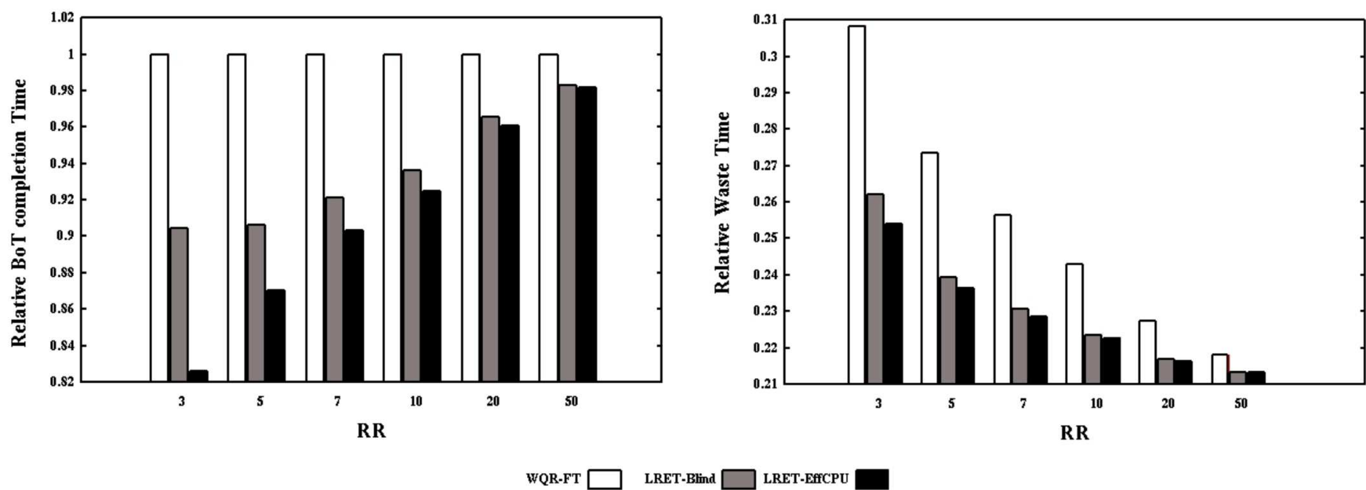

Fig. 6. Public-resource Desktop Grid: (a) Average BoT completion time; (b) Wasted CPU time

larger amount of wasted time.

V. Conclusions and future work

In this paper we have proposed *fault-aware scheduling*, a novel approach to scheduling Bag-of-Tasks applications on Desktop Grids that, rather than just tolerating resource failures and unplanned periods of unavailability, tries to avoid them as much as possible by jointly exploiting the information concerning tasks requirements and resource availability. We have proposed 8 different scheduling policies and have evaluated them, for a set of realistic scenarios and workloads, via simulation. As shown by our results, fault-aware scheduling results both in better application performance and resource utilization than WQR-FT, the fault-tolerant scheduler that – at the best of our knowledge – provides the best performance among those of its class.

As indicated by our results, *LRET*-based policies are constantly better than their *SRET*-based counterparts. As for the machine selection policies, we have observed that for Desktop Grids characterized by a relatively low heterogeneity and high availability it practically does not matter which one is chosen, while for Desktop Grids exhibiting dual properties *EffCPU* results in better performance. Our results also indicate that the knowledge of the fault-time distribution does not improve the performance over machine selection policies exploiting that concerning the effective computing power delivered by resources. We believe, however, that the benefits of this type of this information can be increased by better machine selection policies, that we plan to investigate as part of our future work. Furthermore, we plan to study the effects of other scheduling mechanisms, such as dynamic replication threshold or the usage of task-dependent replication thresholds. Finally, we plan to carry out a more thorough experimentation, involving workloads where multiple Bag-of-Tasks are simultaneously submitted.

Acknowledgments

This work has been supported by the Italian MIUR grant no. RBNE01WEJT (FIRB "WebMINDS" project) and by the National Science Foundation grant no. NGS-0305390.

References

[1] J. Nabrzyski, J. Schopf, and J. Eglarz, *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2003.

[2] D. Kondo, A. Chien, and H. Casanova, "Resource management for rapid application turnaround on enterprise desktop grids," in *Proc. of Super Computing Conference*, 2004.

[3] W. Cirne and et al., "Grid computing for bag of tasks applications," in *Proc. of* 3rd *IFIP Conf. on E-Commerce, E-Business and E-Government*, 2003.

[4] D. Paranhos, W. Cirne, and F. Brasileiro, "Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids," in *Proc. of the Euro-Par 2003: International Conference on Parallel and Distributed Computing*, 2003.

[5] F. Berman, R. Wolski, and et al., "Adaptive computing on the grid using apples," *IEEE Trans. on Parallel and Distributed Systems*, vol. 14, no. 4, 2004.

[6] 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*, 2000.

[7] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," *Proceedings of the 9th Heterogeneous Computing Workshop*, p. 349, 2000.

[8] D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. Chien, "Characterizing and evaluating desktop grids: An empirical study," in *Proc. of Int. Parallel and Distributed Symposium (IPDPS'04)*, 2004.

[9] J. Brevik, D. Nurmi, and R. Wolski, "Quantifying machine availability in networked and desktop grid systemsr," in *Proc. of IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2004)*. IEEE Press, 2004.

[10] J. Abawajy, "Fault-tolerant scheduling policy for grid computing systems," in *Proc. of* 18th *Int. Parallel and Distributed Processing Symposium*, 2004.

[11] J. Weissman and D. Womack, "Fault tolerant scheduling in distributed networks," Department of Computer Science, University of Texas, San Antonio, Tech. Rep. TR CS-96-10, September 1996.

[12] C. Anglano and M. Canonico, "Fault-tolerant scheduling for bag-of-task grid applications," in *Proc. of European Grid Conference, EGC 2005*, 2005.

[13] P. A. Dinda, "Online prediction of the running time of tasks," in *SIGMETRICS/Performance*, 2001, pp. 336–337.

[14] R. Wolski, N. T. Spring, and J. Hayes, "Predicting the CPU availability of time-shared unix systems on the computational grid," *Cluster Computing*, vol. 3, no. 4, pp. 293–301, 2000. [Online]. Available: citeseer.ist.psu.edu/wolski98predicting.html

[15] J. Brevik, D. Nurmi, and R. Wolski, "Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems," in *Proc. of 4th Int. Workshop on Global and Peer-to-Peer Computing*, 2004.

[16] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia: architecture and performance of an enterprise desktop grid system," *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 597–610, 2003.

[17] B. Uk, M. Taufer, T. Stricker, G. Settanni, and A. Cavalli, "Implementation and characterization of protein folding on a desktop computational grid - is charmm a suitable candidate for the united devices metaprocessor?" Institute for Comutersystems, ETH Zurich, Tech. Rep. 385, 2002.

[18] "The nbench project," 2005, http://www.tux.org/ mayer/linux/bmark.html.

[19] R. Wolski, N. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Journal of the Future Generation Computer Systems*, vol. 15, no. 5, pp. 757–768, 1999.

[20] M. Canonico, "Scheduling Algorithms for Bag-of-Tasks Applications on Fault-Prone Desktop Grids," Ph.D. dissertation, University of Turin, 2006.

[21] D. Nurmi, J. Brevik, and R. Wolski, "Modeling machine availability in enterprise and wide-area distributed computing environments," Department of Computer Science, University of California, Santa Barbara, Tech. Rep. CS2003-28, 2003.

[22] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Communications of the ACM*, vol. 17, pp. 530–531, 1974.