

Using Checkpointing to Enhance Turnaround Time on Institutional Desktop Grids

Patricio Domingues
*School of Technology and Management
Polytechnic Institute of Leiria, Portugal
patricio@estg.ipleiria.pt*

Artur Andrzejak
*Zuse-Institute Berlin
Berlin, Germany
andrzejak@zib.de*

Luis Moura Silva
*Dep. Engenharia Informática
Univ. Coimbra, Portugal
luis@dei.uc.pt*

Abstract

In this paper, we present a checkpoint-based scheme to improve the turnaround time of bag-of-tasks applications executed on institutional desktop grids. We propose to share checkpoints among desktop machines in order to reduce the negative impact of resource volatility. Several scheduling policies are evaluated in our study: FCFS, adaptive timeouts, simple replication, replication with checkpoint on demand, and prediction-based checkpointing combined with replication.

We used a set of real traces collected from an academic desktop grid environment to perform trace-driven simulations of the proposed scheduling algorithms. The results show that using a shared checkpoint approach may considerably reduce the turnaround time of the applications when compared to the private checkpoints methodology.

1. Introduction

It is a well-known fact that desktop computers have large amounts of idle CPU cycles that can be harnessed. For instance, in their survey comprising around 330,000 SETI@home's hosts, Anderson and Fedak found that an average of 89.9% of CPU cycles was volunteered to the public-computing project [1]. To exploit these desktop resources, several middleware frameworks have been developed. Examples include academic projects like BOINC [2], XtremWeb [3], Condor [4] and Alchemi.Net [5], and commercial solutions like Entropia [6], to name just a few.

The success of Internet-based grid computing can be measured by the numerous existing volunteer projects, like the popular SETI@home [7] among many others [8], with the most popular of them attracting tens of thousands of volunteers.

Usefulness of desktop grids is not limited to public volunteer projects. It is frequent for institutions like

corporations and academics to own a significant number of personal computers, primarily devoted to low demanding computing activities like e-office and alike. These machines, if properly organized as an institutional desktop grid, can be used by local users to execute demanding e-science applications like simulations. The attractiveness of institutional desktop grids is reinforced by their fast local or metropolitan networks, especially, if the institution is concentrated in a single geographical point as it is often the case.

One of the major problems of desktop grids lies in the volatility of resources [9]. Choi et al. classify the failures of desktop grid resources into two broad classes: volatility failures and interference failures [10]. The former includes network outages and machines crashes. Interference failures simply correspond to the case where the owner of machine claims it back and any running foreign task may have to be interrupted.

Checkpointing is a common solution to cope with the high failure rate of computing nodes. It consists in periodically saving the application state into stable storage. When a failure interrupts a running task, the application can be resumed from the last available checkpoint. There are two main types of checkpoints: system-level and application-level [11]. Apart from Condor [12], which relies on system-level checkpoint, all major middleware tools, like BOINC and XtremWeb, make use of application-level checkpointing. Therefore, in this study, we only consider application-level checkpointing.

An important issue regarding checkpointing lies in the physical location where checkpoints are stored. A limitation of the existing middleware like BOINC is that checkpoints are private, being stored in the same machine where the task is running. If that machine becomes unavailable, the checkpoint file cannot be used and the task has to be restarted from scratch in another machine.

In this paper, we explore the benefits of sharing checkpoints in institutional desktop grid environments for the purpose of optimizing the turnaround time. Under our approach, the checkpoints are saved in a reliable central storage and thus can be used for restoring, moving or replicating tasks among machines, independently of the availability of the machine(s) that saved the checkpoints. Specifically, we evaluate several scheduling policies focused on delivering fast turnaround time for typical bag-of-tasks applications [13] executed over institutional desktop grids.

The rest of this paper is organized as follows. Section 2 describes the proposed scheduling methodologies. Section 3 details the simulated environment, while section 4 presents the main results. In Section 5 we review related work, and section 6 concludes the paper.

2. Scheduling Policies

Bag-of-tasks applications are usually the most suitable for desktop grids since they do not require intra-task communication. While traditional eager scheduling algorithms like First-Come-First-Served (FCFS) yield good performance in high throughput-oriented systems, they are normally inefficient for delivering fast turnaround times [14].

We propose several variations to the FCFS policy with the goal of improving the turnaround time. All those variations include support for shared checkpoints. This enables checkpoint mobility and fast recovery of tasks, especially when compared to private checkpoints. The combination of the shared checkpointing with FCFS scheduling results in the following scheduling policies:

- adaptive timeout (FCFS-AT),
- task replication (FCFS-TR),
- task replication with checkpoint on demand (FCFS-TR-DMD),
- prediction combined with checkpoint on demand (FCFS-PRDCT-DMD).

2.1. FCFS-AT

FCFS-AT improves the base FCFS strategy by including an adaptive timeout that defines the maximum time given to the machine for completing the execution of the assigned task. If the timeout occurs before the task has been completed, the scheduler can reassign the task to another machine where it will be restarted, hopefully from the last checkpoint. In the private checkpoint model, the task

needs to be restarted from scratch in a free machine, unless the original executing machine returns to the available state.

The timeout takes into account the CPU reference time to complete the task together with the performance metric obtained with the Bytemark benchmark [15]. Both values are used to estimate the minimum time needed by the candidate machine to complete the task assuming an ideal execution. A tolerance is added to the base timeout to cope with overheads and unpredictability. This tolerance depends on the expected task length, on the time of the day and on the day of the week. Table 1 shows the tolerance factor applied to the estimated ideal completion time to obtain the total timeout time. We introduce different factors for the nighttime periods and the weekends to take advantage of the higher stability of the desktop grid resources during these periods of low or non-existence of human presence.

CPU Reference Time (sec) / Time of day	Timeout factor
≤ 1800	1.5
]1800,3599]	1.325
≥ 3600	1.25
Nighttime (0.00-8.00 am)	1.10
Weekend (Sat, Sun)	1.05

Table 1: Timeout tolerance factors for FCFS-AT

2.2. FCFS-TR

FCFS-TR adds task replication on top of the FCFS-AT scheduling policy. The strategy is to make use of task replication at the final phase of the computation, when all uncompleted tasks have already been assigned and, at least, one machine is free. Replicating a task augments the probability of a faster completion of the task, especially if the replica is scheduled to a faster machine than the current one. Even if the replica is assigned to a slower or equal performance machine, it can still be useful acting as a backup in case the primary machine fails or gets delayed.

In all replica-based policies, the number of replicas-per-task (henceforth *replica count*) is limited by a predefined threshold. The purpose of the replica count limitation is to promote fairness in the replica functionality avoiding that some excessively replicated tasks clutter the resources. Note that we assume that institutional desktop grids are reliable and trustable environments, and thus we do not resort to task redundancy for validation of results [16].

2.3 FCFS-TR-DMD

FCFS-TR-DMD adds the checkpoint on demand feature to task replication implemented by FCFS-TR. In this strategy the scheduler might ask a worker to perform a checkpoint operation of its current task. This checkpoint is useful immediately before the creation of a replica (see FCFS-TR), since it allows the replica to start with an up-to-date state of the source task.

It is important to note that checkpointing on demand introduces two requirements to the grid infrastructure: (i) master-initiated communication to workers (to ask for a checkpointing operation); (ii) the ability of a task to be checkpointed at any time, potentially at some special breakpoints of the application. Master-initiated communication might be avoided if the desktop grid systems allows the master to send information and commands in response to a worker's periodic heartbeat. Depending on the granularity of the task and how the checkpoints are programmed, it is reasonable to assume that checkpoints can be saved between two different iterations of a task. In this work, we do not consider the delays that may be imposed by such imperfect timing of checkpoints on demand.

2.4. FCFS-PRDCT-DMD

The FCFS-PRDCT-DMD scheduling policy resorts to short-term prediction of machines' availability on top of FCFS-TR-DMD. When a prediction result indicates that a particular machine might be unavailable in the next scheduling interval, the scheduler requests a checkpoint on that running task. It then initiates the creation of a replica if the conditions of the FCFS-TR strategy are met, that is, at least a free machine exists and the replica count for the considered task has not yet been reached. The rationale behind this policy is to anticipate the unavailability of machines, taking the proper measures to reduce the effects of unavailability in the execution of the application. In this work, the prediction method used was the sequential minimal optimization (SMO) algorithm which yielded the best prediction results for machine availability in a previous study [17].

3. Evaluation Setup

We now present the computing environment that was simulated to assess the scheduling methodologies, and then we describe the traces employed to drive the simulations.

3.1 Machines

We used two machine sets, henceforth identified as M01 and M04, as two reference environments. The M01 set holds 32 identical fast machines of type D (see Table 2), while the M04 set is a mix holding 8 machines of each type, that is, 8 machines of type A, 8 of type B, 8 of type C and 8 of type D. The M01 group is worth 54.14 reference machines, against 36.66 for M04, meaning that M01 is 1.48 times faster than M04.

The four types of simulated machines are summarized in Table 2. The column CPU describes the machine's CPU type and clock speed, while the next column indicates the performance index of the corresponding machine group. This index corresponds to the arithmetic mean of the Bytemark benchmark indexes INT and FP [15]. The fourth column corresponds to the performance ratio of machines relatively to the reference machine. This machine, shown in the last row of Table 2, is used as a reference for calibrating execution of tasks. For instance, a task requiring 1800 seconds of CPU on the reference machine, would take nearly 3475 seconds on a type A machine, and slightly more than 1063 seconds on a type D machine.

Type	CPU	Perf. index	Ratio to reference
A	PIII@650 MHz	12.952	0.518
B	PIII@1.1 GHz	21.533	0.861
C	P4@2.4 GHz	37.791	1.511
D	P4@3.0 GHz	42.320	1.692
Avg.	--	28.649	1.146
Reference	P4@1.6 GHz	25.008	1.00

Table 2: Groups of simulated nodes and their performance

3.2 Trace

All simulations were driven by a trace collected from two academic classrooms of 16 machines each, with all machines running the Windows 2000 operating system. The trace is comprised of 473,556 samples collected over 39 consecutive days every two minutes at the machines, for a total of 27,193 sample periods.

Each sample aggregates several metrics, like machine uptime and CPU idleness, to name just the relevant metrics for this work. The machines are used for classes and by students to implement their practical assignments, and to access email and the web. An important issue regarding the machines is that no shutdown policy exists, that is, when leaving a machine, a user is advised but not obliged to shut it down. Therefore, machines may remain powered on for several days.

The weekly distribution of the number of available machines is depicted in Figure 1. The plot displays both the average (top line) and the standard-deviation (bottom line) of the number of available machines. Analyzing the plot, it is possible to clearly identify the weekend period and, in a lesser degree, the nighttime periods by their flat lines. For the whole trace, the mean count of accessible machines was 17.42 (represented by a flat line in the plot), yielding an availability of 54.44%.

To quantify the volatility of the trace, we computed the *variation count* for every machine. A *variation* is defined as a change of state in a machine’s availability, either from available to unavailable, or vice-versa. The variation count of a trace is then a set which holds, for every machine of the trace, the machines’ variation count. To ease comparisons, the *variation ratio* of a machine is defined as the ratio between its actual variation count and the maximum number of variations (which corresponds to the number of samples minus one). We assume the broad definition of availability, upon which a machine is considered available as long it is powered on, independently of the existence of an interactive user. This definition follows what can be configured for some desktop grid workers like BOINC or even Condor (by default, Condor does not permit the coexistence of interactive usage and volunteer computation, but this can easily be changed).

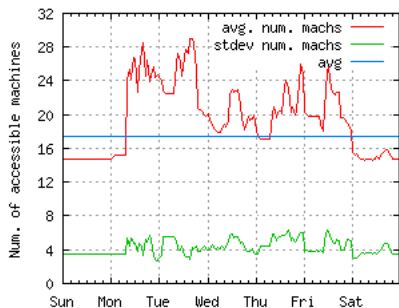


Figure 1: Weekly distribution of accessible machines

For the considered trace, the mean variation count is 91.25 per machine (standard deviation is 22.68), with the variation ratio ranging from 0.17 to 0.60. The average variation ratio is 0.35 (standard deviation is 0.10), a very low value, indicator of the low volatility of the machines. Indeed, although the low average variation ratio can in part be explained by the usage of our broad definition of availability, it is essentially caused by the stability of the studied academic resources. This is due to the lack of a mandatory power off policy, which permits that the machines maintain their availability state for long periods.

To evaluate the proposed scheduling policies under a more stringent environment, namely with a higher level of volatility, a threshold condition regarding the minimum level of idle CPU was defined. Specifically, under the CPU Idle Threshold condition (CIT), a machine is defined as candidate to scavenging at a given time t , only if its CPU idleness is greater than CIT. In practical terms, the CIT condition is equivalent to a resource owner defining that the machine should only be considered for opportunistic computing when CPU idleness is above a predefined threshold. In this paper, besides the nil CIT, which corresponds to the original trace, simulations were carried out with a 90% CIT condition. This corresponds to a scenario, where less than 10% CPU is consumed by the resource owner. As expected, variability of resources for a 90% CIT is much more pronounced than the nil CIT. The mean variation count is 707.78 (with a 447.7 standard deviation), and the average variation ratio is 2.69 (standard deviation is 1.62). Thus, relatively to the nil CIT, resources measured under the 90% CIT are nearly 8 times more volatile.

4. Results

Simulations were carried out with applications comprised of either 25 or 75 tasks, with individual tasks requiring either 1800 or 7200 seconds of CPU time when run on the reference machine. The impact of checkpointing policies on the execution turnaround time was measured by varying the number of saved checkpoints per task execution. Specifically, simulations were carried out with no checkpoints, one checkpoint and nine checkpoints per task. In single-checkpointed executions, the checkpoint is saved when the task reached half of its execution, while for the nine-checkpoint executions, a checkpointing operation is performed every 5% of the execution.

The size of individual checkpoint was set to 1 MB in each case. In fact, simulations with bigger checkpoint sizes (up to 10 MB) did not yield significant differences, mostly due to the fast communication links of LAN environments (the network speed was set to 100 Mbps).

To assess the effects of the weekday/weekend variations of the trace over the workloads, separated simulations were carried out for these periods. Additionally, to prevent bias caused by particular specificities and localities of the trace, all simulations were executed multiple times from different starting points, with reported turnaround times corresponding to the mean average of the multiple executions.

Finally, the replication count threshold was set to 3 for all replica-based scheduling policies, since this proved to be a balanced value.

4.1 Ideal Execution Time

The turnaround times are reported relatively to the Ideal Execution Time (IET). Specifically, unless otherwise stated, results correspond to the slowdown ratio, that is, the ratio of the application turnaround time relatively to the IET for the given task characteristics (the number of tasks and the reference CPU time requirement). The IET measures the theoretical turnaround time that would be required for the execution of the application under ideal conditions, that is, with absolutely no overhead. Table 3 lists the IETs (in minutes) for the studied scenarios.

Number of tasks	Task (seconds)	Turnaround (minutes)	
		M01	M04
25	1800	17.73	35.46
25	7200	70.91	141.83
75	1800	53.19	70.91
75	7200	212.74	283.66

Table 3: Ideal Execution Time for M01 and M04

4.2 Results for Shared Checkpointing

To assess the effect of sharing checkpoints on turnaround time, we simulated both private and shared checkpoint techniques. On the plots, the shared version of a scheduling policy is identified with an *S* suffix, while the private version is labeled with a *P* suffix.

The checkpoint sharing approach yields a faster turnaround time than the private checkpoint technique as shown in Figure 2. Figure 2 displays the turnaround time for the possible combinations of the number of tasks (25 and 75) with the length of individual tasks (1800 and 7200 seconds), considering the single-checkpoint FCFS policy run over the M04 set of machines, and the CIT respectively sets to 0% (left plot) and to 90% (right plot). In all cases, the benefit of using a central checkpoint server clearly pays-off the overhead of checkpointing and allows the scheduler to get faster turnaround times. As expected, turnaround times for the 90% CIT cases are comprehensively longer than the equivalent ones when a 0% CIT is considered. This is due to the fewer usable cycles that follows from a 90% CIT.

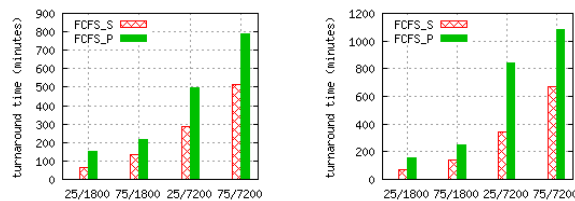


Figure 2: Turnaround times for the shared and private FCFS, with the M04 set, one checkpoint per execution and CIT=0% (left) and CIT=90% (right)

4.3 Results for the Scheduling Policies

In this section, we first present the turnaround times yielded with a 0% CIT, and then analyze the policies under a 90% CIT. We split results between weekdays and weekends. Furthermore, instead of reporting the absolute turnaround times, we present slowdown ratios relative to IET, with lower values meaning faster, and thus better, turnaround times. Every plot aggregates the slowdown ratios for the following scheduling policies: adaptive timeout (AT), first-come first-served (FCFS), simple replication (TR), replication with checkpoint on-demand (TR-DMD), replication with prediction and checkpoint on-demand (TR-PRDCT-DMD).

4.3.1. Weekdays with nil CIT.

Figure 3 aggregates the slowdown ratio plots of the machine sets M01 (top) and M04 (bottom) for the execution of 25 tasks of 1800-second CPU time on weekdays under a 0% CIT.

For the M01 set, all scheduling methodologies perform equally, apart from the prediction-based policy and the private-FCFS, which yield the worse slowdown ratios. As expected in a homogeneous set of machines, replication yields no advantage.

For the M04 machine set, the replication-based policies delivered the best turnaround times, with the checkpoint on-demand mechanism yielding benefits relatively to the simple replication policy. The positive performance of the replication-based scheduling can be explained by the heterogeneity of the M04 set, which creates opportunities for replication, namely when a task is replicated to a faster machine, something that could not occur with the homogeneous M01 machine set.

The 75/1800 case (not shown due to space consideration) yielded results similar to the 25/1800 one, indicating that, for the turnaround times, the number of tasks is not as relevant as the task duration. This is further confirmed by the 25/7200 case (also not shown) which behaves similarly to the 75/7200 one.

Figure 4 presents the results for the slowdown ratios considering the 75/7200 case. These results confirm our thoughts regarding the appropriateness of the prediction-based policy to longer tasks in heterogeneous environments. For the homogeneous machine set M01, both basic tasks replication (TR) and replication with checkpoint on-demand (TR-DMD) deliver the fastest turnaround time.

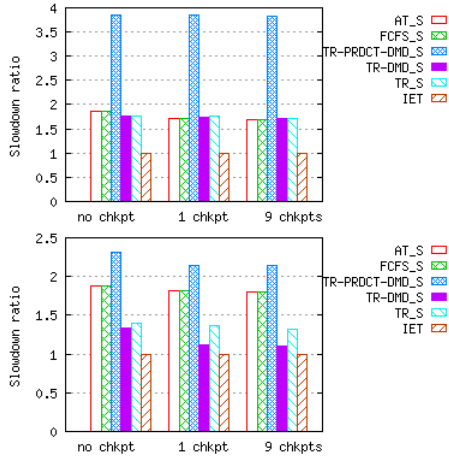


Figure 3: Slowdown ratios for 25/1800 tasks on M01 (top) and M04 (bottom) for a 0% CIT

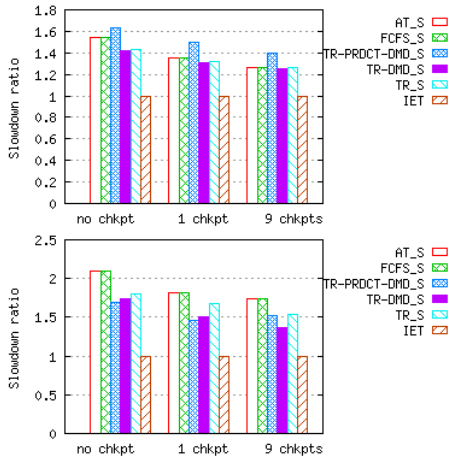


Figure 4: Slowdown ratios for 75/7200 tasks on M01 (top) and M04 (bottom) for a 0% CIT

4.3.2. Weekends with nil CIT.

For weekends executions, only the 75/7200 case is shown (Figure 5) since the results for all the other cases follow a similar pattern.

With the homogeneous set of machines M01, the simple replication (TR) scheduling outperforms all other policies. This means that the TR policy is suited to stable and homogeneous environments like the one found on weekend periods.

An interesting observation is that, contrary to what occurs on the weekday period, the shared-checkpoint versions present no real advantage over the private-checkpoint ones, especially with the homogeneous machine set (M01). This is due to the higher stability of resources on weekends.

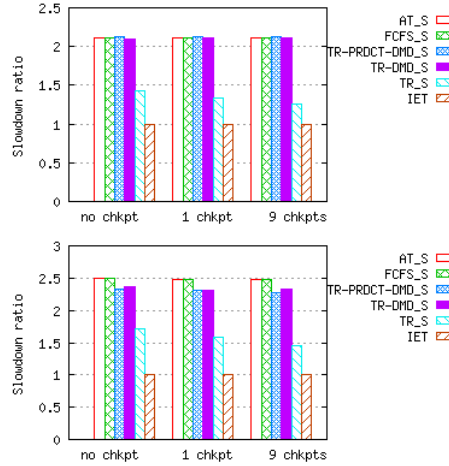


Figure 5: Slowdown ratios for weekend execution of 75/7200 tasks on M01 (top) and M04 (bottom)

4.3.3. Weekdays with 90% CIT.

Results regarding 90% CIT display similar shapes that the one observed for a 0% CIT. This is noticeable on the comparison of Figure 6, which presents the slowdown ratio for the 75/7200 case with a 90% CIT, and Figure 4, which depicts the same case for a 0% CIT.

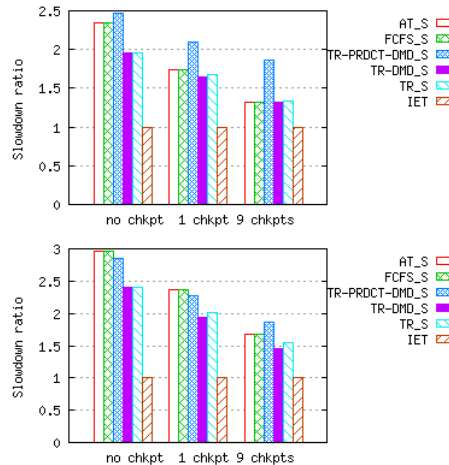


Figure 6: Slowdown ratios for 75/7200 tasks on M01 (top) and M04 (bottom) for a 90% CIT

As expected, yielded turnaround times are lengthier for the 90% CIT. Another consequence of the higher

CIT is seen in the effectiveness of checkpointing, with 9-checkpoint executions performing much better than the versions with no checkpoint. For all shown cases, the TR-DMD methodology consistently yields the fastest executions. Once again, task replication only brings benefits in heterogeneous machine sets.

5. Related work

Scheduling methodologies for reducing turnaround times of task-parallel applications in desktop grids were thoroughly studied by Kondo et al. [14]. The authors analyzed several strategies such as resource exclusion, resource prioritization, and task duplication. The study also resorted to trace-based simulations, using traces collected from the Entropia desktop grid environment [18]. However, the study targeted only small sized-tasks, with the length of tasks being 5, 15 and 35 minutes of CPU time. Moreover, the work did not consider migration nor checkpointing, assuming that interrupted tasks would be restarted from scratch. Although acceptable for small tasks, this assumption is not appropriate for long-running tasks, especially in volatile environments.

The OurGrid project implements the workqueue-with-replication scheduling policy (WQR) [13]. Under this scheme, tasks are assigned in a FCFS-like manner, regardless of the metrics related with performance of machines. When all tasks have been distributed to workers, and if there are enough free resources, the system creates replicas from randomly chosen tasks. WQR acts as a best-effort scheduler, not guaranteeing the execution of all tasks. This differs from our work, which is based on the premise that an application must be completely executed, thus requiring that the scheduler enforces the execution of all the tasks. The authors conclude that task replication significantly augments the probability of an application being terminated.

Anglano and Canonico [19] propose WQR-FT which extends the basic WQR methodology with replication and checkpointing. They recommend checkpointing usage for environments with unknown availability, since it yields significant improvements when volatility of resources is high. Likewise WQR, WQR-FT is also limited by its best-effort approach, with no guarantee that all tasks comprising a given application are effectively executed.

Weng and Lu [20] study the scheduling of bag-of-tasks with associated input data over grid environments (LAN and WAN) of heterogeneous machines. They propose the Qsufferage algorithm which considers the influence of input data

repositories' location on scheduling. The study confirms that the size of the input data of tasks has an impact in the performance of the heuristic-based algorithms.

Zhou and Lo [21] propose the Wave scheduler for running tasks in volunteer peer-to-peer systems. Wave scheduler uses time zones to organize peers, so that tasks are preferentially run during workers' nighttime periods. At the end of the nighttime period, unfinished tasks are migrated to a new nighttime zone. In this way, tasks ride a wave of idle cycles around the world, which reduces the turnaround time. However, the exploitation of moving night zones is only feasible in a wide-scale system.

Taufer et al. [22] define a scheduling methodology based on availability and reliability of workers. Specifically, workers are classified based on their availability and reliability. The scheduler deploys the tasks based on this classification, assigning tasks to workers accordingly to the priority of tasks and the reliability and availability of workers. The proposed scheduling policy targets BOINC-based projects, taking advantage of the fact that this middleware already collects enough information to classify individual workers.

6. Conclusions

This paper presented a set of checkpoint-based scheduling strategies which were devised for reducing the turnaround time of bag-of-task applications in institutional desktop grids.

The shared-checkpoint versions obtained much faster turnaround times than the private-checkpoint versions and thus its usage should be promoted for users with soft real time deadlines, at least, in controlled institutional environments. The results also point out that replication-based policy merged with checkpoint on demand can be effective, especially in heterogeneous clusters. Regarding prediction-based techniques, although they seem inappropriate for short tasks, they yield substantial benefits with long-running tasks.

Finally, as demonstrated by our traces, the day of the week and time strongly influence the availability of resources. This suggests that for better efficiency and faster turnaround time, schedulers for institutional desktop grids should adapt their scheduling policies and associated parameters (checkpoint frequency, etc.) to the day of the week and hour of the day.

Acknowledgments

This research work was carried out in part under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

References

- [1] D. Anderson and G. Fedak, "The Computational and Storage Potential of Volunteer Computing," presented at 6th International Symposium on Cluster Computing and the Grid (CCGRID06), Singapore, 2006.
- [2] D. Anderson, "BOINC: A System for Public-Resource Computing and Storage," presented at 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA., 2004.
- [3] G. Fedak, C. Germain, V. Neri, and F. Cappello, "XtremWeb: A Generic Global Computing System," presented at 1st Int'l Symposium on Cluster Computing and the Grid (CCGRID'01), Brisbane, 2001.
- [4] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," presented at 8th International Conference on Distributed Computing Systems (ICDCS), Washington, DC, 1988.
- [5] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Alchemi: A.NET-Based Enterprise Grid Computing System," presented at 6th International Conference on Internet Computing (ICOMP'05), Las Vegas, USA, 2005.
- [6] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropy: architecture and performance of an enterprise desktop grid system," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 597-610, 2003.
- [7] SETI, "SETI@Home Project (<http://setiathome.berkeley.edu/>)," 2005.
- [8] D. Computing, "Distributed Computing Info (<http://distributedcomputing.info/>)," 2005.
- [9] A. Gupta, B. Lin, and P. A. Dinda, "Measuring and understanding user comfort with resource borrowing," presented at High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on, 2004.
- [10] S. Choi, M. Baik, C. Hwang, J. Gil, and H. Yu, "Volunteer availability based fault tolerant scheduling mechanism in DG computing environment," presented at 3rd IEEE International Symposium on Network Computing and Applications (NCA'04), 2004.
- [11] L. M. Silva and J. G. Silva, "System-level versus user-defined checkpointing," presented at IEEE Symposium on Seventeenth Reliable Distributed Systems, West Lafayette, Indiana, USA, 1998.
- [12] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System," University of Wisconsin, Madison, Computer Sciences. Technical Report 1346, 1997.
- [13] W. Cirne, F. Brasileiro, N. Andrade, R. Santos, A. Andrade, R. Novaes, and M. Mowbray, "Labs of the World, Unite!" Universidade Federal de Campina Grande, Departamento de Sistemas e Computação, UFCG/DSC Technical Report 07/2005, 2005.
- [14] D. Kondo, A. Chien, and H. Casanova, "Resource management for rapid application turnaround on enterprise desktop grids," presented at 2004 ACM/IEEE conference on Supercomputing, 2004.
- [15] BYTE, "BYTEmark project page (<http://www.byte.com/bmark/>)," Byte, 1996.
- [16] L. Sarmanta, "Volunteer Computing (PhD)," in *Dept. of Electrical Engineering and Computer Science: MIT*, 2001.
- [17] A. Andrzejak, P. Domingues, and L. Silva, "Classifier-based Capacity Prediction for Desktop Grids," presented at Integrated research in Grid Computing - CoreGRID workshop, Pisa, Italy, 2005.
- [18] D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. Chien, "Characterizing and evaluating desktop grids: an empirical study," presented at 18th International Parallel and Distributed Processing Symposium (IPDPS'04), 2004.
- [19] C. Anglano and M. Canonico, "Fault-Tolerant Scheduling for Bag-of-Tasks Grid Applications," *Proc. of the 2005 European Grid Conference (EuroGrid 2005). Lecture Notes in Computer Science*, vol. 3470, pp. 630, 2005.
- [20] C. Weng and X. Lu, "Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid," vol. 21, pp. 271, 2005.
- [21] D. Zhou and V. Lo, "Wave Scheduler: Scheduling for Faster Turnaround Time in Peer-based Desktop Grid Systems," presented at 11th Workshop on Job Scheduling Strategies for Parallel Processing (ICS 2005), Cambridge, MA, 2005.
- [22] M. Taufer, P. Teller, D. Anderson, and I. C. L. Brooks, "Metrics for Effective Resource Management in Global Computing Environments," presented at 1st IEEE International Conference on e-Science and Grid Technologies (eScience 2005), Melbourne, Australia, 2005.