

Scheduling for Fast Turnaround Time on Institutional Desktop grid

Patricio Domingues¹, Artur Andrzejak², Luis Silva³,

*¹School of Technology and Management - Polytechnic Institute of Leiria
2411-901 Leiria, Portugal
patricio@estg.ipleiria.pt*

*²Zuse-Institute Berlin
Takustr. 7, 14195 Berlin, Germany
andrzejak@zib.de*

*³CISUC - Centre for Informatics and Systems of the University of Coimbra
Polo II - Pinhal de Marrocos
3030-290 Coimbra, Portugal
luis@dei.uc.pt*



CoreGRID Technical Report
Number TR-0027

January 30, 2006

Institute on System Architecture

CoreGRID - Network of Excellence
URL: <http://www.coregrid.net>

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-00426

Scheduling for Fast Turnaround Time on Institutional Desktop grid

Patricio Domingues¹, Artur Andrzejak², Luis Silva³,

*1School of Technology and Management - Polytechnic Institute of Leiria
2411-901 Leiria, Portugal
patricio@estg.ipleiria.pt*

*2Zuse-Institute Berlin
Takustr. 7, 14195 Berlin, Germany
andrzejak@zib.de*

*3CISUC - Centre for Informatics and Systems of the University of Coimbra
Polo II - Pinhal de Marrocos
3030-290 Coimbra, Portugal
luis@dei.uc.pt*

*CoreGRID TR-0027
January 30, 2006*

Abstract

Institutional desktop grids, that is, grids comprised by the desktop machines of an institution (academic or corporate) can act as an important source of computing power for local user, if the high volatility of such resources is properly harnessed. In this paper, we focus on scheduling techniques that can improve turnaround time of bag-of-tasks applications. For that purpose, we combine shared-checkpoint management with several scheduling methods like FCFS, adaptive timeout, simple replication and short-term availability predictions. The main goal is to minimize turnaround time by reducing negative effects of resources volatility on the executions of bag-of-tasks applications. We present and assess several scheduling policies oriented toward fast turnaround times, and assess them through trace-based simulations over the resources of 32 machines of two classrooms of an academic environment.

1 Introduction

In the last years, desktop grids, defined as pool of networked desktop computers, have emerged as an attractive way to exploit resources of desktop systems, which would otherwise remain largely idle. Indeed, it is a well-studied fact that machines primarily devoted to regular human-dependent interactions, like e-office applications (work processing, spreadsheets, etc.) and e-communications such as instant messaging, e-mail, and Internet browsing barely use their

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

resources. For instance, Heap [1] reports nearly 95% CPU idleness amongst Unix machines, with an even higher value of 97% measured in Windows machines assigned to academic classrooms [2]. Furthermore, in their comprehensive study of more than 200000 SETI@home [3] hosts, Anderson and Fedak [4] report that an average of 89.9% of CPU was volunteered to public computing through the BOINC platform [5], meaning that roughly 90% of CPU would have been wasted if it was not exploited by BOINC projects.

The potential of desktop grid is huge: a modern but none the less regular PC offers a respectable computational power, undreamed of just a few years ago. For instance, as of November 2005, the average PC volunteering resources to SETI@home delivered 1.568 GFlops, had 806 MB of main memory, 1.78 GB of swap space, and a 60 GB disk with unused space amounting to 35 GB [4]. The recent trend toward multiple CPU cores probably means that in a short-term future, average resources idleness will be higher, possibly including entire CPU core.

Along with to PC continuously increasing performance, the exponential growth of Internet has permitted the success of major public-computing projects like SETI@home, Einstein@home, lhc@home and folding@home, just to name a few. A complete reference for public computing projects can be found at [6].

Practically all desktop grid applications follow the master-worker paradigm, upon which a central master entity coordinates the whole computation. Under this model, a worker requests tasks from the master, processing the assigned tasks in background mode and under minimal local system priority, since priority to resource usage is always given to local users.

A major hurdle that poises efficient usage of desktop grids lies in the high volatility of resources. Indeed, in addition to expectable resources failures (hardware and software) that cause machine crashes, network outages and other similar problems, machines can become, without prior warning, inaccessible for volunteer computing simply because the machine's owner has claimed back the machine's resources. Choi et al. classify desktop grid resource failures into two broad classes: volatility failures and interference failures [7]. The former includes network outages and machines crashes that render the resources inaccessible. The latter emerges from the volunteer nature of resources which are ultimately under the control of their respective owner.

Checkpointing is a well-know fault-tolerance mechanism that might be used to reduce the limitations imposed by the high volatility of resources. It consists in periodically saving the application or the executing process's state to stable storage. So, whenever a failure interrupts a volunteer computation, the application can be resumed from the last stable checkpoint as soon as the original resource or an equivalent one is available. This way, the lost caused by a failure are reduced to the computing time elapsed since the last usable checkpoint was saved.

Checkpoints can be broadly classified in two types: system-level and application-level. The former involves saving the whole state of the process that executes the application we wish to preserve. Although transparent to both programmer and user, system-level checkpointing usually generates big chunk of data to be saved, since the whole process' state need to be preserved. This significantly increases the costs of the checkpoint operations [8]. Furthermore, a system-level checkpoint is strongly tied to the operating system where it was created and thus can only be used for restoring execution in a compatible system, which frequently means a machine with compatible hardware and operating system. This seriously hinders portability of system-level checkpoints. On the contrary, checkpoint at the application-level requires the direct intervention of the application programmer to point out the meaningful data of the application that needs to be preserved. However, since only the needed data are saved, application-level checkpoints are way lighter than system-level. Moreover, if saved in a transportable format, checkpoints can be used to restore the execution in another machine, possibly with a different hardware and operating

system (as long as a version of the application exists for the restore machine), thus permitting task migration. Apart from Condor [9], which relies on system-level checkpoint (and only on certain Unix platforms) [8], all major volunteer middlewares like BOINC and XtremWeb resort to application-level checkpoint.

1.1 Institutional desktop grids

Institutions like academics and enterprise often own hundredths of PCs, primarily devoted to low demanding computing activities like e-office and alike. Although these environments could potentially be good contributors of public computing projects, several reasons may preclude their participation. First, security issues might forbid execution of foreign and unauthorized software, in fear of the consequence that a worm exploiting eventual vulnerabilities of the desktop grid middleware might damage the computing infrastructure or that weaknesses might compromise the integrity and privacy of data kept in the institution computing system. Furthermore, an institution might be unwilling to commit its resources like computers, electric power and staff to support them, to volunteer projects that might not provide a direct and immediate return to the institution. This is especially true for corporate environments. More importantly, the institution idle resources might benefit the own institution if properly harnessed. Indeed, while most users barely load the computing resources available to them, a minority of users need all the computational power they can get. Thus, properly fueled, unexploited resources from low demanding users could benefit resource hungry users.

Institutional desktop grids frequently benefit from more homogeneous computing infrastructures, with, for instance, a substantial percentage of machines matching a unique operating system and software profile (hardware might be more variable, with newer machines delivering better performance than older ones). Additionally, it is rather frequent for an institutional desktop grid to have resources concentrated in a single geographical point, with a fast network infrastructure. This is especially true for resources that are located within distance of local area network technology, for instance in an academic campus or at a corporate headquarter. Also, security can be more effectively controlled, with incidents (malicious usage, anomalies) being potentially traced back to the source in a more efficient manner.

All the above cited characteristics facilitate the deployment and management of desktop grid middleware, easing exploitation of institutional desktop grid resources.

1.2 Motivation

In this work, we study scheduling strategies focused on delivering fast turnaround time for typical bag-of-tasks applications, that is, applications comprised of independent tasks. Our main motivation is to devise scheduling policies optimized for delivering fast turnaround time in institutional desktop grid environments.

For bag-of-tasks applications, turnaround time is defined as the elapsed time between the submission of the first task until the last task is completed. Fast turnaround times are especially relevant in iterative and/or speculative research based on computing, like for instance research dependent on computer simulation [10]. Indeed, in iterative/speculative-based research, next steps of the work are dependent on the outcome of the current execution. Thus, speeding up execution turnaround time permits exploring more hypotheses or alternatively to faster reach a solution.

To overcome volatility that hinders efficiency of resource usage in desktop grids and undermines turnaround time, we resort to shared checkpoints, adaptive execution timeouts, task replication and short-term predictions of resource availability.

The remainder of this paper is organized as follows. Section 2 introduces the studied scheduling methodologies, while section 3 describes the simulated scenarios used to assess the proposed scheduling policies. In section 4, we discuss main results. Section 5 describes related work. Finally, section 6 concludes the paper.

2 Scheduling methodologies

Since they do not require intra-task communication and present a high computation-to-communication ratio, bag-of-tasks applications are reasonably fitted for desktop grid environments. However, traditional eager scheduling algorithms like First Come First Served (FCFS), which yield good performance in high-throughput oriented systems, are normally inefficient if applied unchanged in environments where fast turnaround times are sought [11].

In order to adapt FCFS to fast turnaround-oriented environments we applied several changes. First, we added support for shared checkpoints. Under the shared model, checkpoints are kept in a common repository and not exclusively at the executing machine's storage. This permits checkpoints to be shared amongst machines, allowing tasks to be resumed, moved or replicated to another machines, accordingly to the circumstances. This effectively provides for task mobility, and consequently for an improved usage of resources relatively to private checkpointing. A requirement of the shared checkpoint model is the existence of a storage infrastructure accessible to all machines and able to support the possible concurrent access of the computers involved in computation. In a medium-sized institutional environment, a regular PC acting as file server should suffice to implement a shared checkpoint service. Furthermore, this service can be located jointly with the scheduler service, to promote synergies between the scheduling and the checkpointing systems.

Based on shared checkpoints, and using FCFS as base, several scheduling policies were devised, namely FCFS-AT (AT stands for *adaptive timeout*), FCFS-TR (*task replication*) and FCFS-PRDCT-DMD (*prediction on demand*).

2.1 FCFS-AT

FCFS-AT improves the base FCFS with the inclusion of an adaptive timeout that sets a time limit for the termination of a task. This timeout defines the maximum time conceded to the machine for completing the execution of the assigned task. Should the timeout expire before the task has been completed, the scheduler considers the task as non-completed and can therefore reassign it to another machine, where it will be restarted, possibly from the last stable checkpoint (assuming shared checkpoint is enabled).

The timeout is defined when a task is assigned to a requesting machine and its duration takes into account the needed CPU reference time to complete the task as well as the machine computing performance as given by the Bytemark benchmark indexes [12]. These values are used to compute the minimum time needed by the candidate machine to complete the task, estimating an ideal execution, that is, a fully dedicated and flawless machine. To accommodate for reality, a tolerance is added to the base timeout. This tolerance, defined by way of a percentage of the timeout's base-time, varies accordingly to the CPU reference time still needed to complete the task (the bigger the reference CPU time needs, the bigger the tolerance percentage). However, if

the execution is scheduled for a night-period or for a weekend, the tolerance is fixed, respectively to 10% and 5%. This is to take advantage of the stability of the desktop grid resources during period of low or non-existence of human presence.

2.2 FCFS-TR

FCFS-TR adds task replication on top of the FCFS-AT scheduling policy. The main strategy is to resort to task replication at the terminal phase of the computation, when all uncompleted tasks are already assigned and there is at least one free machine. The underlying principle is that replicating a task, especially if the replica is scheduled to a faster machine than the current one, augments the probability of a faster completion of the task, and thus might reduce the turnaround time. Even, if the replica is assigned to a slower or equal performance machine, it can still be useful acting as backup in case if the primary machine fails or get delayed.

In order to avoid excessive replicas of a same task, something that could perturb the balanced access to resources, the number of replicas of a task is maintained under a predefined threshold. Therefore, tasks which level of replication has already reached the limit can only be further replicated when one of the current replicas is interrupted. For the experimental results presented in this paper, the maximum number of replicas was set to 3. Furthermore, when a task is terminated all other results produced by replicas that might exist are simply discarded (we do not consider redundancy for sabotage-tolerance purposes, assuming that all computing resources behaves honestly).

2.3 FCFS-PRDCT-DMD

The FCFS-PRDCT-DMD scheduling policy resorts to short-term prediction regarding machines' availability on top of FCFS-TR. When a prediction indicates that a currently requested machine might fail in the next scheduling interval, the scheduler orders a checkpoint (henceforth referred as "checkpoint on demand") and then promotes the creation of a replica if conditions are met (that is, at least a free machine exists and the maximum number of replicas for the considered task has not yet been reached). The rationale behind this policy is to anticipate unavailability of machines, taking the proper measures to reduce or even eliminate the effect of the machine unavailability on 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 in a previous study [13].

3 Simulated scenarios

In this section, we present the computing environments that were simulated to assess the scheduling methodologies. We resorted to simulation due to the infeasibility of conducting repeatable experiments in real desktop grid systems. Specifically, we used trace-driven simulations.

3.1 Trace

All simulations were driven by a trace collected from two academic classrooms of 16 machines each. The trace is comprised of samples, collected every two minutes at the machines,

through the Distributed Data Collector (DDC) [14]. Each sample aggregates several metrics, like uptime, CPU idleness and memory to name just the metrics that are relevant to this work. Besides being used for classes, machines of both classrooms are accessible to any student of the institution, for the purpose of performing their practical assignments, to access e-mail 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 obligated to shut it down. Therefore, machines may remain powered on for several days [2].

The trace was collected during a class period, over 39 consecutive days and contains nearly 27000 samples. **Figure 1** plots the number of accessible machines over time for both traces. Analyzing the left plot, it is possible to identify the night-time and weekend period by their flat lines, since the number of powered on machines remain practically constant when the classrooms are closed (from 4 am to 8 am on weekdays and from Saturday 9 pm to Monday 8 am on weekends).

Figure 1 plots the number of accessible machines over time for the trace. Analyzing the plot, it is possible to identify the night-time and weekend period by their flat lines, since the number of powered on machines remain practically constant when the classrooms are closed (from 4 am to 8 am on weekdays and from Saturday 9 pm to Monday 8 am on weekends). For the whole trace, the mean count of accessible machines, represented by a flat line in the plot, was 17.42, yielding an availability of 54.44%.

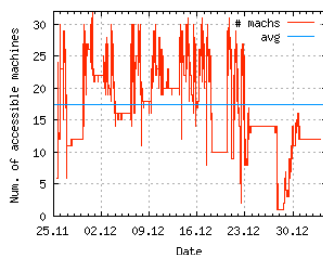


Figure 1: Count of machines over time

3.2 Machine sets

To assess the behavior of the scheduling policies relatively to raw machine speed and heterogeneity of resources, two machine sets, labeled as *M01* and *M04*, were simulated. The *M01* set holds 32 identical fast machines of type D (see Table 1), while the *M04* set, as the name implies, is a mix of four different groups of machines (each one of the four group holds 8 machines of a type depicted in Table 1). The *M01* group delivers a higher computational power than *M04*, since only 8 machines of *M04* are as fastest as the machines of the *M01* set.

The four types of machines simulated are summarized in Table 1. The column “CPU” identifies the machine’s CPU type, while the next column indicates the performance index of the respective machine group. This performance index corresponds to the arithmetic mean of the Bytemark benchmark indexes INT and FP [12], with higher values expressing higher performances. Finally, the fourth column corresponds to the reference performance ratio of machines relatively to the reference machine. The reference machine, shown in the last row of Table 1, is used as reference for calibrating execution of tasks. For instance, a task requiring 3600 seconds of CPU when run on the reference machine, would take nearly 6950 seconds on a type A machine.

Type	CPU	Performance index	Ratio to reference machine
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 1: Main characteristics of types of machines

4 Main results

In this section, we present and discuss the simulation results of the proposed scheduling policies. Due to space limitation, only the most relevant results are shown.

4.1 Simulated tasks

Simulations were carried out with applications comprised of 25 and 75 tasks, with individual tasks requiring 1800 and 7200 seconds of reference machine's CPU time.

The influence of checkpoint policies over the execution turnaround time were measured by varying the number of saved checkpoints during the execution of a task. For this purpose, simulations were carried out modeling executions with one checkpoint, nine checkpoints, besides non-checkpointed executions. In single-checkpointed executions, the checkpoint is saved when the task reached half of its execution, while for the nine-checkpoint executions checkpointing occurs every 5% of the execution. The size of individual checkpoint was set to 1 MB for all simulated executions.

Regarding access to desktop grid resources, we assumed that the presence of a local user at a machine would not suspend the execution of an opportunistic task at that machine. Instead, preserving the quality of service for a local user relies on the priority mechanism of the host machine's operating system, with foreign tasks run under the lowest priority level. This behavior is similar to the one permitted by the BOINC client.

To assess the effects of the weekday/weekend variations of the trace over the workloads (weekends present much lower volatility of resources), separated simulations were carried out for weekdays and weekends. Finally, to prevent results biased by particular specificities of the trace, all simulations were carried multiple times from different starting points, with reported turnaround times corresponding to the mean average of the multiple executions.

4.2 Ideal execution time

In order to permit a clearer assessment of the results, turnaround times are reported relatively to the Ideal Execution Time (IET). Specifically, reported results correspond to the slowdown ratio, that is, the ratio of the application turnaround time relatively to the IET for the given tasks characteristics (number of tasks and reference CPU time requirement). The IET measures the theoretical turnaround time that would be required for the execution of the application if

performed under ideal conditions, such as fully dedicated and failure-free machines, and with absolutely no overhead. Table 2 reports the IETs (in minutes) for the scenarios analyzed in this study.

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 2: Ideal Execution Time for the machine/task pairs

4.3 Results

In this section we present the main results. We first analyze the results obtained for weekdays and then results for weekends.

Instead of reporting turnaround times, we present, hopefully, more meaningful slowdown ratios relative to IET. Specifically, every plot aggregates the slowdown ratios for the following scheduling policies: adaptive timeout (AT), first come first served (FCFS), transfer replicate (TR) and transfer replicate with prediction and checkpoint on demand (TR-PRDCT-DMD). It is important to note that plots display slowdown ratios, with lower values meaning faster, and thus better, turnaround times.

To assess the effect of sharing checkpoints on turnaround time, the shared version of a policy (identified by the S suffix) is plotted next to the private version (labeled with a P suffix). Finally, to evaluate the effect of checkpoint frequency over the turnaround time, each plot holds the results for the three studied checkpoint frequencies: no checkpoint (left-most group of bars), one checkpoint (center group) and nine checkpoints (right-most group).

A common property of the results is that, for every scheduling policy, the shared-checkpoint version almost always yields a better turnaround time than the private version. This is due to the fact that a shared-checkpoint scheduler holds a global view of the system (refreshed at the periodicity of the system, that is, 2-minute in the case of our simulated scenario). Thus, the scheduler knows about the whole state of the system (i.e., which machines are available and which are down), and consequently can react much more promptly when a failure occurs (for instance, a machine goes down), by rescheduling tasks accordingly to availabilities. In fact, the positive effects of a global view of the system on turnaround time can be evaluated comparing shared and private policies when checkpointing is disabled.

Weekdays

Figure 2 aggregates the slowdown ratio plots of machine sets M01 (left) and M04 (right) for the execution of 25 tasks of 1800-second CPU time on weekdays. For the M01 set (henceforth identified as 25/1800/M01), best results are achieved with the TR policy with the adaptive timeout (AT) scheduling also performing reasonably. Surprisingly, the prediction-based policy yields poor results, even for the shared approach. We hypothesize that the prediction methodology, while trying to replicate tasks executing on machine predicted as unavailable on the next scheduling round, might occupy other machines that would otherwise run more useful

replications (from the point of view of turnaround time). As we should see later on, the prediction-based method yields more positive results over larger applications.

For the M04 machine set, the transfer based policies yielded the best results. This can be explained by the medium heterogeneity of the M04 set, which creates opportunities for beneficial replications, namely when a task is replicated to a faster machine, something that does not occur under M01, since all machines of this set are equal. The results also point out that benefits of checkpointing are visible even for single-checkpointed executions.

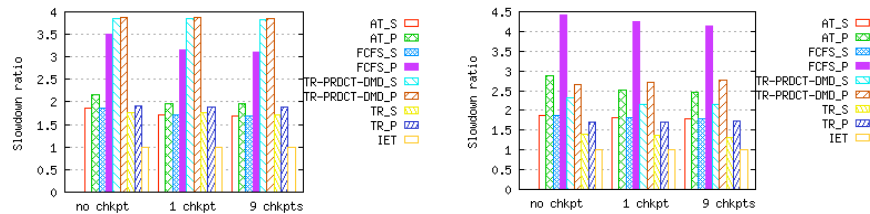


Figure 2: Slowdown ratio for 25/1800 tasks on M01 (left) and M04 (right)

For the 25/7200 case (Figure 3), the prediction-based policy performs practically on par with the simple replication policy that still delivers the best turnaround time on the M01 set. For the M04 set, TR-PRDCT-DMD outperforms the simple replication scheduling. In fact, the prediction-based scheduling yields much better results for 7200-second tasks than it does for 1800-second ones, indicating that the policy is better suited for longer tasks, especially in heterogeneous machine environments like M04. Once again, while checkpoint benefits on turnaround times are seen with low checkpoint frequency (i.e. one checkpoint), increasing the checkpoint frequency to nine checkpoints per execution only yields marginal improvements.

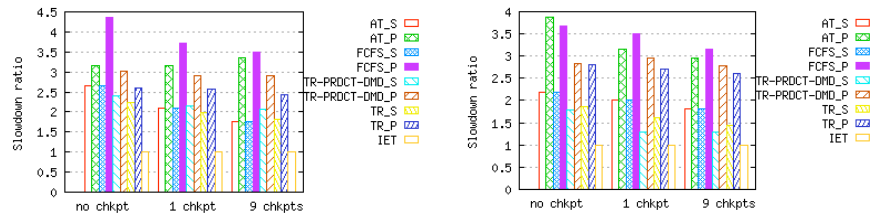


Figure 3: Slowdown ratio for 25/7200 tasks on M01 (left) and M04 (right)

The relative shape of the 75/1800 plots (Figure 4) are somewhat similar to 25/1800 indicating that the number of tasks does not seem to influence much the behavior of the scheduling policies. In this case, the replication based policy is consistently the fastest policy regardless of the machine set. Again, results demonstrate that the prediction-based policy is not suited for short tasks. Finally, and quite surprisingly, the basic FCFS policy performs reasonably for the M04 set.

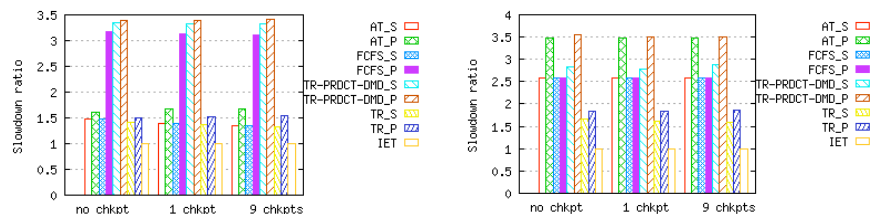


Figure 4: Slowdown ratio for 75/1800 tasks on M01 (left) and M04 (right)

Slowdown ratios for the 75/7200 case, shown in **Figure 5**, further confirm our supposition regarding the appropriateness of the prediction-based policy to longer tasks in heterogeneous environments. In fact, independently of the checkpoint frequency, the prediction-based policy outperforms the other scheduling methods. For the homogeneous machine set M01, basic tasks replication (TR) yields the best results.

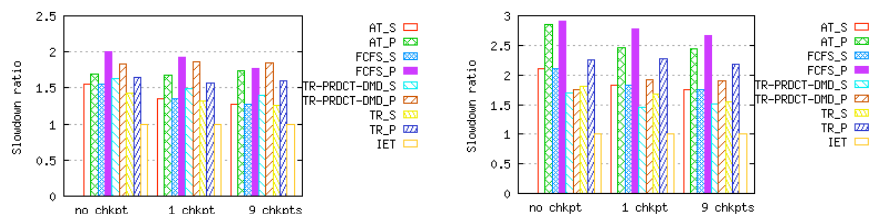


Figure 5: Slowdown ratio for 75/7200 tasks on M01 (left) and M04 (right)

Weekends

For weekends executions, only the 75/7200 case is shown (**Figure 6**) since results for all the other cases follow a similar pattern. With a homogeneous set of machines (set M01), the simple replication (TR) scheduling outperforms all others policies, which behaves similarly to each other. This means that the TR policy is suited to stable and homogeneous environments like the one found on weekends.

Once again, the heterogeneity of the M04 set seems to be the main cause of the low slowdown ratios obtained by the replication-based policies. The fact that the prediction-based scheduling yields good results for the M04 set confirms the appropriateness of such policy for a high number of long tasks when run in heterogeneous environments.

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, with the shared-checkpoint method advantage of early detection of interrupted tasks being useless, since practically no interruption occurs.

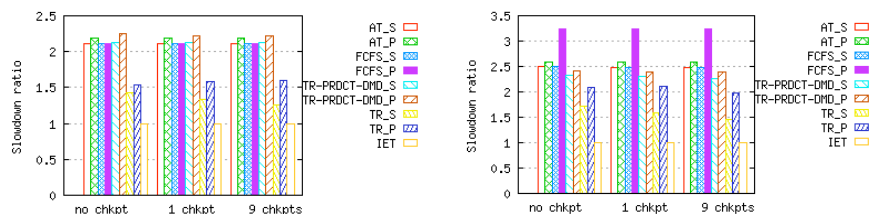


Figure 6: Slowdown ratio for weekend execution of 75/7200 tasks on M01 (left) and M04 (right)

5 Related Work

In this section we briefly review studies related to scheduling bag-of-tasks over desktop grids. Scheduling methodologies for speeding turnaround times of task parallel applications executed over desktop grids were exhaustively studied by Kondo et al. [11]. The authors used a self-developed Perl simulator to analyze several scheduling strategies such as resource exclusion (excluding less performance resources), resource prioritization (intrusting tasks preferably to the CoreGRID TR-0027

fastest machines), and task duplication. Similarly to our approach, the study resorted to trace-based simulations, using traces collected from the Entropia desktop grid environment [15]. 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 task migration, neither checkpointing, requiring instead that interrupted tasks be restarted from scratch. Although acceptable for small task duration like 5 to 15 minutes, the need to fully restart interrupted tasks can be unbearable for bigger tasks, especially in highly volatile environments.

Cirne et al. [16] promote the OurGrid project in the quest of attracting computing laboratories from any place to create a global community of shared resources. For that purposes, laboratories that join OurGrid grant access to their own resources, receiving in turn the permission to use other laboratories' resources. The OurGrid projects implements the workqueue with replication policy (WQR) scheduling policy. Under this scheme, tasks are assigned to requesting workers, in a FCFS-like way, regardless to the machine performance characteristics (OurGrid scheduling assumes no knowledge about the machines connected to the project). When all tasks have been distributed to workers, and if there are enough resources, replicas from randomly chosen tasks are created. OurGrid acts as a best-effort scheduler, not guaranteeing the execution of all tasks. In fact, the authors conclude that task replication significantly augment the probability of an application being terminated.

Anglano and Canonico [17] propose the work queue with replication and fault-tolerance (WQR-FT) scheduling algorithm for bag-of-tasks applications. WQR-FT enhances the basic WQR methodology with replication and checkpointing. Similarly to WQR, WQR-FT is a knowledge-free scheduler, meaning that it does not require full knowledge about the state of the resources that comprise the execution system. The study concludes that for environments with unknown availability, checkpoint is recommended since it may yield significant improvement when volatility of resources is high. A limitation of WQR-FT lies in its best-effort approach, in the sense that the scheduler does not guarantee that all tasks comprising a given application are effectively executed. 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.

Weng and Lu [18] 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 for scheduling tasks. They conclude that the size of task's input data influences the performance of the studied heuristic-based algorithms. In our work we assume that task input size is not meaningful relatively to the 1MB checkpoint size used.

Zhou and Lo [19] propose the Wave scheduler for optimizing turnaround time of bag-of-tasks run over volunteer peer-to-peer systems. Wave scheduler uses time zones to organize peers, so that tasks are run during workers' nighttime periods. At the end of the nighttime period, unfinished tasks are migrated to a new nighttime zone. This way, tasks ride a wave of idle cycles around the world, to reduce turnaround time. However, the exploitation of moving night zones is only feasible in a wide-scale system.

The authors of [20] define a scheduling methodology based on availability and reliability of workers. Specifically, workers are classified based on their availability and reliability, yielding four classes: high availability/low reliability (HA/LR), high availability/high reliability (HA/HR), low availability/low reliability (LA/LR) and low availability/high reliability (LA/HR). The scheduling strategy deploys tasks in order to match the task priority to an appropriate worker. For instance, high priorities tasks are preferentially assigned to HA/HR workers. The proposed

scheduling policy targets BOINC-based projects since this middleware already collects enough information to classify individual workers.

6 Conclusions

This paper presented and evaluated several checkpoint-based scheduling methodologies oriented toward delivering fast turnaround times in institutional desktop grid environments. The evaluation of policies was carried out through trace-driven simulations, resorting to a trace collected in two classrooms of an academic institution.

As expected, for all proposed scheduling policies, the shared-checkpoint versions yielded much better turnaround times than the private-checkpoint ones. This advantage is partially due to the fact that the proposed shared-checkpoint scheduler maintains a minimal global view of the desktop grid system, knowing which machines are accessible (for instance, through a simple heartbeat mechanism). This permits early detection of machine failures and consequently allows for a possibly fast rescheduling of interrupted tasks, with the advantage of using the last stable checkpoint to resume execution, thus preserving previously done computation. On the contrary, private-checkpoint based schemes can only address the issue of detecting interrupted tasks through timeouts, or speculatively scheduling replicas.

Apart some minor situations, all scheduling policies performed better than the classical FCFS methodology. In fact, only the prediction-based method produced worse turnaround time than FCFS, and only for small applications, with a low number of small tasks (the 25/1800 cases). However, for longer applications and namely longer tasks (7200 seconds), the prediction-based policy outperformed the other scheduling methodologies.

Replication-based methodologies are appropriate to heterogeneous machine environments, since replication of tasks to faster machines yields opportunities for quicker termination and thus shorter turnaround times.

Regarding checkpoints, results show that the effects of checkpoint frequency (that is, the number of checkpoints saved during the execution of a task) over turnaround time are minors, only yielding marginal improvements. This means that the used desktop grid resources are only mildly hindered by volatility, providing an interesting environment for resource harvesting. However, checkpoint usage would certainly be more rewarding if suspension of opportunistic execution of a task occurred whenever a local user occupied a machine, since task interruption frequency would then be much higher.

Acknowledgements

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

References

- [1] D. G. Heap, "Taurus - A Taxonomy of Actual Utilization of Real UNIX and Windows Servers," IBM White Paper GM12-0191, 2003.
- [2] P. Domingues, P. Marques, and L. Silva, "Resource Usage of Windows Computer Laboratories," presented at International Conference Parallel Processing (ICPP 2005)/Workshop PEN-PCGCS, Oslo, Norway, 2005.
- [3] SETI, "SETI@Home Project (<http://setiathome.berkeley.edu/>)," 2005.

- [4] D. Anderson and G. Fedak, "The Computational and Storage Potential of Volunteer Computing," 2005.
- [5] D. Anderson, "BOINC: A System for Public-Resource Computing and Storage," presented at 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA., 2004.
- [6] "Distributed Computing Info (<http://distributedcomputing.info/>)," 2006.
- [7] 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.
- [8] 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.
- [9] D. Thain, T. Tannenbaum, and M. Livny, *Distributed computing in practice: The condor experience*, 2004.
- [10] D. Petrou, G. Gibson, and G. Ganger, "Scheduling speculative tasks in a compute farm," presented at Proceedings of the 2005 ACM/IEEE conference on Supercomputing, 2005.
- [11] 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.
- [12] BYTE, "BYTEmark project page (<http://www.byte.com/bmark/bmark.htm>)," Byte, 1996.
- [13] 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.
- [14] P. Domingues, P. Marques, and L. Silva, "Distributed Data Collection through Remote Probing in Windows Environments," presented at 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'05), Lugano, Switzerland, 2005.
- [15] 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.
- [16] 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.
- [17] C. Anglano and M. Canonico, *Fault-Tolerant Scheduling for Bag-of-Tasks Grid Applications*, 2005.
- [18] 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.
- [19] 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 In Conjunction with ICS 2005, The Cambridge Marriott-Kendall Square, Cambridge, MA, 2005.
- [20] 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.