Providing Fault-Tolerance in Unreliable Grid Systems Through Adaptive Checkpointing and Replication

Maria Chtepen[1], Filip H.A. Claeys[2], Bart Dhoedt[1], Filip De Turck[1], Peter A. Vanrolleghem[2], and Piet Demeester[1]

[1] Department of Information Technology (INTEC), Ghent University, Sint-Pietersnieuwstraat 41, Ghent, Belgium
{maria.chtepen, bart.dhoedt, filip.deturck}@intec.ugent.be
[2] Department of Applied Mathematics, Biometrics and Process Control (BIOMATH), Ghent University, Coupure Links 653, Ghent, Belgium
{filip.claeys, peter.vanrolleghem}@biomath.ugent.be

Abstract. As grids typically consist of autonomously managed subsystems with strongly varying resources, fault-tolerance forms an important aspect of the scheduling process of applications. Two well-known techniques for providing fault-tolerance in grids are periodic task checkpointing and replication. Both techniques mitigate the amount of work lost due to changing system availability but can introduce significant run-time overhead. The latter largely depends on the length of checkpointing interval and the chosen number of replicas, respectively. This paper presents a dynamic scheduling algorithm that switches between periodic checkpointing and replication to exploit the advantages of both techniques and to reduce the overhead. Furthermore, several heuristics are discussed that perform on-line adaptive tuning of the checkpointing period based on historical information on resource behavior. Simulation-based comparison of the proposed combined algorithm versus traditional strategies based on checkpointing and replication only, suggests significant reduction of average task makespan for systems with varying load.

1 Introduction

A typical grid system is an aggregation of (widespread) heterogeneous computational and storage resources managed by different organizations. The term "heterogeneous" addresses in this case not only hardware heterogeneity, but also differences in resources utilization. Resources connected into grids can be dedicated supercomputers, clusters, or merely PCs of individuals utilized inside the grid during idle periods (so-called desktop grids). As a result of this highly autonomous and heterogeneous nature of grid resources, failure becomes a commonplace feature that can have a significant impact on the system performance. A failure can occur due to a resource or network corruption, temporary unavailability periods initiated by resource owners, or sudden increases in resource

load. To reduce the amount of work lost in the presence of failure, two techniques are often applied: task checkpointing and replication. The checkpointing mechanism periodically saves the status of running tasks to a shared storage and uses this data for tasks restore in case of resource failure. Task replication is based on the assumption that the probability of a single resource failure is much higher than of a simultaneous failure of multiple resources. The technique avoids task recomputation by starting several copies of the same task on different resources. A drawback of this approach is that replication introduces significant overhead, which can penalize normal task execution. Therefore, the number of replicas should be carefully chosen in function of system characteristics and the expected load. Since our previous work [1] has extensively studied the task replication issue, this paper is mainly dedicated to the checkpointing approach.
The purpose of checkpointing is to increase fault-tolerance and to speed up application execution on unreliable systems. However, as was shown by Oliner et al. [2], the efficiency of the mechanism is strongly dependent on the length of the checkpointing interval. Overzealous checkpointing can amplify the effects of failure, while infrequent checkpointing results in too much recomputation overhead. As can be presumed, the establishment of an optimal checkpointing frequency is far from a trivial task, which requires good knowledge of the application and the distributed system at hand. Therefore, this paper presents several heuristics that perform on-line adaptive tuning of statically provided checkpointing intervals for parallel applications with independent tasks. The heuristics make use of run-time information on job progress and historical data on stability of the resource where the job is executing. The designed heuristics are intended for incorporation in a dynamic scheduling algorithm that switches between job replication and periodic checkpointing to provide fault-tolerance and to reduce potential job delay resulting from the adoption of both techniques. An evaluation in a simulation environment (i.e. DSiDE [3]) has shown that the designed algorithm can significantly reduce the task delay in systems with varying load, compared to algorithms solely based on either checkpointing or replication.
This paper is organized as follows: in Section 2 the related work is discussed; the assumed system model is presented in Section 3; Section 4 elaborates on the adaptive checkpointing heuristics and the proposed scheduling algorithm; simulation results are introduced in Section 5; Section 6 summarizes the paper.

2 Related Work

Much work has already been accomplished on checkpointing performance prediction and determination of the optimal checkpointing interval for uniprocessor and multi-processor systems. For uniprocessor systems, selection of such an interval is for the most part a solved problem [4]. The results for parallel systems are less straightforward [5] since the research is often based on particular assumptions, which reduce the general applicability of the proposed methods. In particular, it is generally presumed that failures are independent and identically distributed. Studies of real systems, however, show that failures are correlated

temporally and spatially, are not identically distributed. Furthermore, the behavior of checkpointing schemes under these realistic failure distributions does not follow the behavior predicted by standard checkpointing models [2, 6].
Since finding the overall optimal checkpointing frequency is a complicated task, other types of periodic checkpointing optimization were considered in literature. Quaglia [7] presents a checkpointing scheme for optimistic simulation, which is a mixed approach between periodic and probabilistic checkpointing. The algorithm estimates the probability of roll-back before the execution of each simulation event. Whenever the event execution is going to determine a large simulated time increment then a checkpoint is taken prior to this event, otherwise a checkpoint is omitted. To prevent excessively long suspension of checkpoints, a maximum number of allowed event executions between two successive checkpoint operations is fixed. Oliner [8] proposes a so-called cooperative checkpointing approach that allows the application, compiler, and system to jointly decide when checkpoints should be performed. Specifically, the application requests checkpoints, which have been optimized for performance by the compiler, and the system grants or denies these requests. The latter uses a combination of network traffic data and critical event predictions to make its decisions. This approach has a disadvantage that applications have to be modified to trigger checkpointing periodically at appropriate points in their execution. Research performed by Tantawi et al. [9] investigates an analytical checkpointing strategy based on the concept of *equicost*, which varies the checkpointing interval according to a balance between the checkpointing cost and the probability of failure.
Job replication and determination of the optimal number of replicas are other rich fields of research [1, 10]. However, to our knowledge, no methods dynamically altering between replication and checkpointing were introduced so far.

3 The System Model

A grid system running parallel applications with independent tasks is considered. The system is an aggregation of geographically dispersed sites, assembling collocated interchangeable computational (CR) and storage (SR) resources, and a number of services, such as a scheduler (GSched) and an information service (IS). It is assumed that all the grid components are stable except for CRs. The latter possess a varying failure and restore behavior, which is modelled to mimic reality as much as possible. As outlined by Zhang et al. [6], failures on large-scale distributed systems are mostly correlated and tend to occur in bursts. Besides, the failure distribution among nodes is not uniform. In fact, there are strong spatial correlations between failures and nodes, where a small fraction of the nodes incur most of the failures.
The checkpointing mechanism is either periodically activated by the running application or by the GSched, with a frequency predetermined by the end-user / application developer. In both cases, it takes W seconds before the checkpoint is completed and thus can be utilized for an eventual job restore. Furthermore, each checkpoint adds V seconds of overhead to the job run-time. Both parame-

ters largely depend on size C of the saved job state. There is also a recovery overhead P, which is the time required for a job to restart from a checkpoint. Obviously, the overhead introduced by periodic checkpointing and restart may not exceed the overhead of the job restores without use of checkpointing data. To limit the overhead, a good choice of checkpointing frequency I is of crucial importance. Considering the assumed grid model, I^j_{opt}, which is the optimal checkpointing interval for a job j, is largely determined by the following function $I^j_{opt} = f(E^j_r, F_r, C^j)$, where E^j_r is the execution time of j on the resource r and F_r stands for the mean time between failures of r. Additionally, the value of I_{opt} should be within the limits $V < I_{opt} < E_r$ to make sure that jobs make execution progress despite of periodic checkpointing.
The difficulty of finding I_{opt} is in fact that it is often hard to determine the exact values of the application and system parameters. Furthermore, E_r and F_r can vary over time as a consequence of changing system loads and resource failure/restore patterns. This suggests that a fixed statically determined checkpointing interval may be an inefficient solution when optimizing system throughput. In what follows, a number of heuristics for adaptive checkpointing are presented.

4 Adaptive Checkpointing Strategies

4.1 Last Failure Dependent Checkpointing (LFDC)

One of the main disadvantages of unconditional periodic task checkpointing (UTC) is that it performs identically whether the task is executed on a volatile or a stable resource. To deal with this shortcoming, LFDC adjusts the initial job checkpointing interval to the behavior of each individual resource r and the total execution time of the considered task j, which results in a customized checkpointing frequency I^j_r. For each resource a timestamp T^f_r of its last detected failure is kept. When no failure has occurred so far, T^f_r is initiated with the system "start" time. GSched evaluates all checkpointing requests and allows only these for which the comparison $T_c - T^f_r \leq E^j_r$ evaluates to true, where T_c is the current system time. Otherwise, the checkpoint is omitted to avoid unnecessary overhead as it is assumed that the resource is "stable". To prevent excessively long checkpoints suspension, a maximum number of checkpoint omissions can be defined, similar to the solution proposed in [7].

4.2 Mean Failure Dependent Checkpointing (MFDC)

Contrary to LFDC, MFDC adapts the initial checkpointing frequency in function of a resource mean failure interval (MF_r), which reduces the effect of an individual failure event. Furthermore, the considered job parameter is refined from the total job length to the estimation of the remaining execution time (RE^j_r). Each time the checkpointing is performed, MFDC saves the task state and modifies the initial interval I to better fit specific resource and job characteristics. The adapted interval I^j_r, is calculated as follows: if r appears to be sufficiently stable

or the task is almost finished ($RE_r^j < MF_r$) the frequency of checkpointing will be reduced by increasing the checkpointing interval $I_r^j = I_r^j + I$; in the other case it is desirable to decrease I_r^j and thus to perform checkpointing more frequently $I_r^j = I_r^j - I$. To keep I_r^j within a reasonable range, MFDC always checks the newly obtained values against predefined boundaries, in such a way that $I_{min} \leq I_r^j \leq I_{max}$. Both I_{min} and I_{max} can either be set by the application or initialized with default values $I_{min} = V + (E_r^j/100)$ and $I_{max} = V + (E_r^j/2)$. In both equations the V term ensures that time between consecutive checkpoints is never less than the time overhead added by each checkpoint, in which case more time is spent on checkpointing than performing useful computations. After the I_r^j interval expires, either the next checkpointing event is performed, or a flag is set indicating that the checkpointing can be accomplished as soon as the application is able to provide a consistent checkpoint.
In case of rather reliable systems, the calibration of checkpointing interval can be accelerated by replacing the default increment value I by a desirable percentage of total or remaining task execution time.

4.3 Adaptive Checkpoint and Replication-Based Scheduling (ACRS)

Checkpointing overhead can be avoided by providing another means to achieve system fault-tolerance. Replication is an efficient and almost costless solution, if the number of task copies is well chosen and there is a sufficient amount of idle computational resources [1]. On the other hand, when computational resources are scarce, replication is undesirable as it considerably delays the start of new jobs. In this section, an adaptive scheme is proposed that dynamically switches between task checkpointing and replication, based on run-time information about system load. When the load is low, the algorithm is in "replication mode", where all tasks with less than R replicas are consequently assigned to the available resources. Different strategies can be defined to determine the order of the assignment, a frequently used one sorts jobs according to the number of already started replicas, to reduce the wait time of new jobs. The selected task replica is then submitted to a grid site s with minimal load $Load_s^{min}$ and the minimum number of identical replicas. The latter is important to reduce the chance of simultaneous replica failure. $Load_s^{min}$ is calculated as follows: $Load_s^{min} = min_{s \in S}((\sum_{r \in s} n_r)/(\sum_{r \in s} MIPS_r))$, where S is the collection of all sites, n_r is the number of tasks on the resource r; and $MIPS_r$ (Million Instructions Per Second) is the CPU speed of r. Inside the chosen site, the task will be assigned to the least loaded available resource with again the smallest number of task replicas. The load of a resource $Load_r$ is determined as $Load_r = n_r/MIPS_r$. The algorithm switches to the "checkpointing mode" when idle resource availability IR drops to a certain limit L. In this mode, ACRS rolls back, if necessary, the earlier distributed active task replicas AR_j and starts task checkpointing. When processing the next task j the following situations can occur:

- $\boldsymbol{AR_j > 0}$: start checkpointing of the most advanced active replica, cancel execution of other replicas

– $\boldsymbol{AR_j = 0}$ and $\boldsymbol{IR > 0}$: start j on the least loaded available resource within the least loaded site
– $\boldsymbol{AR_j = 0}$ and $\boldsymbol{IR = 0}$: select a random replicated job i if any, start checkpointing of its most advanced active replica, cancel execution of other replicas of i, submit j to the best available resource

5 Simulation Results

Performance of the proposed methods was evaluated using the DSiDE simulator, on the bases of a grid model composed of 4 sites (3 CRs each) with varying availability. The availability parameter, which is defined to be the fraction of the total simulation time that the system spends performing useful work, is modelled as a variation of the CR's failure and restore events [1]. Distribution of these events is identical for each resource inside the same site and is depicted in Table 1. The table also shows the distributions with which the burst-based correlated nature of resource failures is approached. A failure event namely triggers the whole burst (see "Burst size") of connected resource malfunctions spread within a relatively short time interval (see "Burst distribution"). To simplify the algorithm comparisons, a workload composed of identical tasks with the following parameters was considered: $S = 30min, In(inputsize) = Out(outputsize) = 2MB, W = 9s, V = 2s, P = 14s$. Furthermore, each CR has 1 MIPS CPU speed and is limited to process at most 2 jobs simultaneously. Initially, the grid is heavily loaded since tasks are submitted in bursts of 6 up to 25 tasks followed by a short (5 to 20 min) idle period. It is also assumed that the application can start generating the next checkpoint, as soon as it is granted permission from GSched. The described grid model is observed during 24 hours of simulated time.
The left part of Figure 1 shows the comparison, in terms of successfully executed

Table 1. Distributions of site failure and restore events together with distributions of the number and frequency of correlated failures

	Failure	**Restore**	**Burst size**	**Burst distribution**
Site 1	Uniform:1-300(s)	Uniform:1-300(s)	Uniform:1-3	Uniform:300-600(s)
Site 2	Uniform:1-900(s)	Uniform:1-300(s)	Uniform:1-3	Uniform:300-600(s)
Site 3	Uniform:1-2400(s)	Uniform:1-300(s)	Uniform:1-3	Uniform:300-600(s)
Site 4	No failure	-	-	-

tasks, between UPC, LFDC and MFDC checkpointing heuristics. The comparison is performed for a varying initial checkpointing frequency. For the MFDC algorithm I_{min} is set to the default value, while no limitation on the I_{max} is imposed. The results show that the performance of UPC strongly depends on the chosen checkpointing frequency. As can be seen on the UPC curve, excessively frequent checkpointing penalizes system performance to a greater extent than

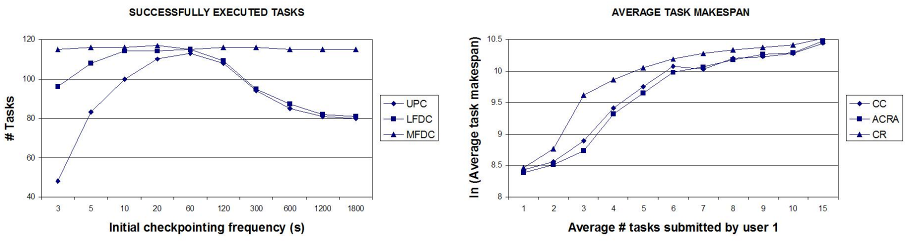

Fig. 1. (a) UPC, LFDC and MFDC checkpointing strategies performance; (b) CC, CR and ACRS scheduling performance

insufficient checkpointing. It can be explained by the fact that the considered grid model is relatively stable, with a total system availability of around 75%. In this case the checkpointing overhead exceeds the overhead of task recomputation. LFDC partially improves the situation by omitting some checkpoints. Since the algorithm doesn't consider checkpoint insertion, the performance for an excessively long checkpointing interval is the same as for UPC. Finally, the fully dynamic scheme of MFDC proves to be the most effective. Starting from a random checkpointing frequency it guarantees system performance close to the one provided by UPC with an optimal checkpointing interval.
Finally, the ACRS ($R = 2, L = 7, I = 30$) is compared against common checkpointing (CC) and replication-based (CR) algorithms. The term "common checkpointing algorithm" refers to an algorithm monitoring resource downtimes and restarting failed jobs from their last saved checkpoint. The considered CC algorithm, as well as ACRS, makes use of the MFDC heuristic with $I = 30$ to determine the frequency of task checkpointing. The principle of the replication-based algorithm is quite straightforward: $R = 2$ replicas of each task are executed on preferably different resources, if a replica fails it is restarted from scratch [1]. It is clear that replication-based techniques can be efficient only when the system possesses some free resources. Fortunately, most of the observed real grids alternate between peak periods and periods with relatively low load. To simulate this behavior, the initial task submission pattern is modified to include 2 users: the first sends to the grid a limited number of tasks every 35-40 min; the second launches significant batch sizes of 15 up to 20 tasks every 3-5 hours. Figure 1 (right) shows the observed behavior of the three algorithms. During the simulations the number of tasks simultaneously submitted by the first user is modified as shown in the figure, which results in variations in system load among different simulations. When the system load is sufficiently low, ACRS and CC process an equal number of tasks, since each submitted task can be assigned to some resource. However, ACRS results in lower average task makespan. When the system load increases, ACRS switches to checkpointing mode after a short transitive phase. Therefore, the algorithm performs almost analogous to CC, except for a short delay due to the mode switch. Finally, CR considerably

underperforms the other algorithms with respect to the number of executed tasks and average makespan. In the considered case ACRS provided for up to 15% reduction of the average task makespan compared to CC. The performance gain certainly depends on the overhead of checkpoints and the number of generated checkpointing events, which is close to optimal for the MFDC heuristic.

6 Conclusion

This paper introduces a number of adaptive mechanisms, which optimize job checkpointing frequency as a function of task and system properties. The heuristics are able to modify the checkpointing interval at run-time reacting on dynamic system changes. Furthermore, a scheduling algorithm combining checkpointing and replication techniques for achieving fault-tolerance is introduced. The algorithm can significantly reduce task execution delay in systems with varying load by transparently switching, when appropriate, between both techniques.

References

1. Chtepen, M., Claeys, F.H.A., Dhoedt, B., De Turck, F., Demeester, P., Vanrolleghem, P.A.: Evaluation of Replication and Rescheduling Heuristics for Grid Systems with Varying Availability. In Proc. of Parallel and Distributed Computing and Systems, Dallas (2006)
2. Oliner, A.J., Sahoo, R.K., Moreira, J.E., Gupta, M.: Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems. In Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Washington (2005)
3. Chtepen, M., Claeys, F.H.A., Dhoedt, B., De Turck, F., Demeester, P., Vanrolleghem, P.A.: Dynamic Scheduling of Computationally Intensive Applications on Unreliable Infrastructures. In Proc. of the 2nd European Modeling and Simulation Symposium, Barcelona, Spain (2006)
4. Vaidya, N.H.: Impact of Checkpoint Latency on Overhead Ratio of a Checkpointing Scheme. IEEE Transactions on Computers, **46-8** (1997) 942–947
5. Wong, K.F., Franklin, M.: Checkpointing in Distributed Systems. Journal of Parallel and Distributed Systems, **35-1** (1996) 67–75
6. Zhang, Y., Squillante, M.S., Sivasubramaniam, A., Sahoo, R.K.: Performance Implications of Failures in Large-Scale Cluster Scheduling. In Proc. of the 10th Workshop on Job Scheduling Strategies for Parallel Processing, New York (2004)
7. Quaglia, F.: Combining Periodic and Probabilistic Checkpointing in Optimistic Simulation. In Proc. of the 13th workshop on Parallel and distributed simulation, Atlanta (1999)
8. Oliner, A.J.: Cooperative Checkpointing for Supercomputing Systems. Master Thesis. Massachusetts Institute of Technology (2005)
9. Tantawi, A.N., Ruschitzka, M.: Performance Analysis of Checkpointing Strategies. In ACM Transactions on Computer Systems, **110** (1984) 123–144
10. Li, Y., Mascagni, M.: Improving Performance via Computational Replication on a Large-Scale Computational Grid. In Proc. of the 3st International Symposium on Cluster Computing and the Grid, Washington (2003)