Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids

Maria Chtepen, Filip H.A. Claeys, Bart Dhoedt, *Member*, *IEEE*, Filip De Turck, *Member*, *IEEE*, Piet Demeester, *Senior Member*, *IEEE*, and Peter A. Vanrolleghem

Abstract—A grid is a distributed computational and storage environment often composed of heterogeneous autonomously managed subsystems. As a result, varying resource availability becomes commonplace, often resulting in loss and delay of executing jobs. To ensure good grid performance, fault tolerance should be taken into account. Commonly utilized techniques for providing fault tolerance in distributed systems are periodic job checkpointing and replication. While very robust, both techniques can delay job execution if inappropriate checkpointing intervals and replica numbers are chosen. This paper introduces several heuristics that dynamically adapt the abovementioned parameters based on information on grid status to provide high job throughput in the presence of failure while reducing the system overhead. Furthermore, a novel fault-tolerant algorithm combining checkpointing and replication is presented. The proposed methods are evaluated in a newly developed grid simulation environment Dynamic Scheduling in Distributed Environments (DSiDE), which allows for easy modeling of dynamic system and job behavior. Simulations are run employing workload and system parameters derived from logs that were collected from several large-scale parallel production systems. Experiments have shown that adaptive approaches can considerably improve system performance, while the preference for one of the solutions depends on particular system characteristics, such as load, job submission patterns, and failure frequency.

Index Terms—Distributed systems, performance of systems, fault tolerance, availability.

1 Introduction

Compared to other distributed environments, such as clusters, complexity of grids mainly originates from decentralized management and resource heterogeneity. The latter refers to hardware, as well as to foreseen utilization. These characteristics often lead to strong variations in grid availability, which in particular depends on resource and network failure rates, administrative policies, and fluctuations in system load. Apparently, runtime changes in system availability can significantly affect application (job) execution. Since for a large group of time-critical or time-consuming jobs delay and loss are not acceptable, fault tolerance should be taken into account.

Providing fault tolerance in a distributed environment, while optimizing resource utilization and job execution times, is a challenging task. To accomplish it, two techniques are often applied: job checkpointing and job replication. In this paper, it is argued that both techniques in their pure static form are not able to cope with unexpected load and failure conditions within grids. Therefore, several solutions are proposed that dynamically adapt the checkpointing frequency and the number of replicas as a reaction on changing system properties (number of active resources and resource failure frequency). Furthermore, a novel hybrid scheduling approach is introduced that switches at runtime between checkpointing and replication depending on the system load. Decisions taken by the abovementioned algorithms are primarily based not only on monitored grid state but also on job characteristics and on collected historical information. Currently, the proposed techniques are limited to address hardware failure in grids running applications composed of independent jobs.

Simulation-based experiments, using the discrete event grid simulator Dynamic Scheduling in Distributed Environments (DSiDE) [1] and a data set derived from real-world logs collected from different large-scale parallel production systems [2], [3] have shown that the adaptive approaches significantly improve distributed system performance. They achieve throughput and fault tolerance comparable with that of static checkpointing and replication with optimal parameters. However, to make an appropriate choice between strategies, some knowledge on system parameters is still required. To deal with the latter issue, the hybrid approach, combining the advantages of both techniques, may be preferred.

The paper is organized as follows: Section 2 discusses related work; Sections 3 elaborate on adaptive checkpointing and provides a simulation-based comparison between different checkpointing approaches; Section 4, in turn, discusses and compares replication-based and hybrid scheduling solutions; while Sections 5 concludes the paper.

- *M. Chtepen, B. Dhoedt, F. De Turck, and P. Demeester are with the Ghent University—IBBT, Department of Information Technology, Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium. E-mail: {Maria.Chtepen, Bart.Dhoedt, Filip.Deturck, Piet.Demeester}@intec.ugent.be.*
- *F.H.A. Claeys is with the CTO MOSTforWATER N.V., Koning Leopold III-laan 2, 8500 Kortrijk, Belgium. E-mail: fc@mostforwater.com.*
- *P.A. Vanrolleghem is with the modelEAU, Departement Genie Civil, Pavillon Pouliot, Universite Laval, Quebec, QC G1V 0A6, Canada. E-mail: Peter.Vanrolleghem@gci.ulaval.ca.*

Manuscript received 14 Aug. 2007; revised 1 Apr. 2008; accepted 14 May 2008; published online 22 May 2008.
Recommended for acceptance by F. Petrini.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2007-08-0278.
Digital Object Identifier no. 10.1109/TPDS.2008.93.

2 Related Work

A large number of research efforts have already been devoted to fault tolerance in the scope of distributed environments. Aspects that have been explored include the design and implementation of fault detection services [4], [5], as well as the development of failure prediction [3], [6], [7], [8] and recovery strategies [9], [10], [11]. The latter are often implemented through job checkpointing in combination with migration and job replication. Although both methods aim to improve system performance in the presence of failure, their effectiveness largely depends on tuning runtime parameters such as the checkpointing interval and the number of replicas [12], [13], [14]. Determining optimal values for these parameters is far from trivial, for it requires good knowledge of the application and the distributed system at hand.

2.1 Checkpointing

To tackle the checkpointing overhead and scalability concerns, different approaches are addressed in the literature. One well-researched technique is known as incremental checkpointing [15]. It reduces data stored during checkpointing to only blocks of memory modified since the last checkpoint. In [16], a checkpointing-based fault-tolerant protocol for MPI jobs is presented, which lowers the overhead during normal execution and allows fast crash recovery by using the ideas of message logging and object-based processor virtualization. The latter limits the reexecution to only the failed processor and allows to distribute the failed work among the other processors. Clearly, this approach is only applicable to homogeneous environments. Nevertheless, another important approach is based on determination of the optimal checkpointing frequency and is called the optimal checkpoint interval problem. Several researches have addressed this problem [17], [18], [19], but they have provided analytical solutions applicable only under specific system assumptions. For instance, it is often assumed that interoccurrence times of failures and repairs for each resource are independent and exponentially distributed. In practice, failures tend to cluster in time, while being caused by a relatively small set of computational nodes [3], [7], [8]. Since optimal solutions do not appear to be generally applicable, static suboptimal solutions were addressed. For instance, in [20], a min-max checkpoint placement method is introduced that determines the suboptimal checkpoint sequence under uncertain circumstances in terms of the system failure time distribution. However, even if the (sub)optimal checkpointing interval is computed beforehand, the distributed system or application parameters upon which the interval is based will presumably change over time. Therefore, new forms of checkpointing optimization were recently considered in literature. One of them is the so-called cooperative checkpointing concept, introduced in [21] and [22], which addresses system performance and robustness issues by allowing the application programmer, the compiler and the runtime system to jointly decide on the necessity of each checkpoint. The checkpointing algorithms proposed in this paper are based on this concept and thus are cooperative (adaptive) heuristics. In [23], another set of cooperative checkpointing schemes is proposed that dynamically adjust the checkpointing interval with as an objective timely job completion in the presence of failure. The schemes use information on remaining job execution time, time left before the deadline, and the expected remaining number of failures before job termination. The latter implies that the system failure distribution should be known in advance. Katsaros et al. [24], in turn, consider only dynamic checkpointing interval reduction in case it leads to computational gain, which is quantified by the sum of the differences between the means for fault-affected and fault-unaffected job response times. In [25], yet another adaptive fault management scheme (FT-Pro) is discussed. Opposite to the combined approach proposed in this paper that uses adaptive checkpointing in combination with replication, FT-Pro combines adaptive checkpointing with proactive process migration. The approach optimizes application execution time by considering the failure impact and the prevention costs. FT-Pro supports three prevention actions: skip checkpoint, take checkpoint, and migrate. The appropriate action is selected based on the predicted frequency of failure. Therefore, the effectiveness of FT-Pro strongly depends on the quality of this prediction.

2.2 Replication

Similar to deciding upon the best checkpointing interval, finding a generally applicable procedure to calculate the optimal number of job replicas is a complicated issue. Several studies have attempted to address this problem [13], [26], but unfortunately, they enforce a number of restrictions on the execution environment, job interdependency, etc. Nowadays, most of the replication-based fault-tolerant algorithms assume a fixed number of job duplicates. However, dynamic solutions have recently started to receive attention. In [10], a dynamic replication-based method is described, called Workqueue with Replication (WQR). Initially, the algorithm distributes a single copy of a job to random idle resources in First Come, First Served (FCFS) order. When the job queue is empty and the system has free resources, replication is activated to cope with varying availability of hosts. The disadvantage of this "delayed-copy" approach is that if a system is heavily loaded for a long period, which is often the case in large scientific or production grids, the replication will be significantly delayed or not activated at all. Furthermore, as was mentioned in [7], most of the failures in distributed environments tend to occur during peak hours, when the WQR failure prevention is turned off by definition. Other interesting research on job replication is reported on in [27], where a group-based dynamic replication mechanism for peer-to-peer grid computing environments is proposed. Whereas the algorithms introduced in our paper dynamically vary the number of job replicas dependent on the system load, the group-based approach determines the amount of replication taking into account the reliability of each volunteer group, which is a group of resources with similar properties.

2.3 Combined Approaches

Several papers [28], [29] describe schemes that combine checkpointing and job replication to deal with transient fault detection. Transient faults are often hard to detect because they do not result in a resource crash but only in a job state modification, which however can lead to wrong output. Therefore, duplicate jobs are executed on different nodes, and their state is compared to track faults. The checkpointing mechanism, in turn, serves two purposes: preservation of a job state to reduce the fault-recovery time and state comparison of job replicas. To our knowledge, no work combining checkpointing and replication was performed thus far with the objective of achieving better resource utilization and improving job execution time.

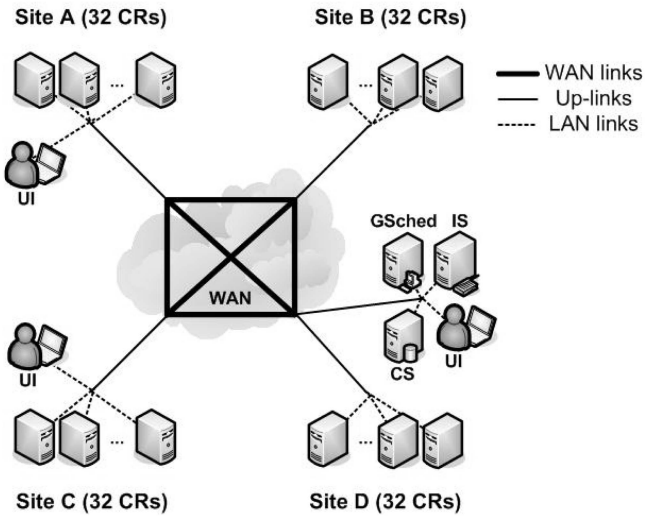


Fig. 1. Example grid architecture: UI (user interface), GSched (grid scheduler), IS (information service), CS (checkpoint server), WAN (wide area network), LAN (local area network).

3 Adaptive Checkpointing Heuristics

3.1 The Checkpointing Model

The grid model considered in this paper consists of geographically dispersed computational sites (S), aggregating altogether 128 computational resources (CR) and a number of general services (Fig. 1). The latter include a user interface (UI), through which jobs are submitted into the system; a scheduler (GSched) responsible for job-resource matchmaking; an information service (IS), which collects job and resource status information required by the GSched; and a checkpoint server (CS), where checkpointing data is made persistent (see Table 1 for a listing of acronyms). The GSched invokes the matchmaking procedure within the predefined scheduling interval I_{GSched}, while the IS collects changes in resource status with a delay I_{IS} to reflect the modification propagation time occurring in actual deployments. The grid sites reside within a Wide Area Network (WAN), while resources belonging to a single site are interconnected by Local Area Networks (LAN). Finally, it is assumed that all grid management services are protected against failure, and only CRs are unstable, with a resource failure affecting all CPUs within a given CR. Contrary to the traditional assumptions, considering failures to be independent and equally spread over all system resources with a particular distribution, failures in this work can be spatially and temporarily correlated, which has proven to be a more realistic presumption in case of large-scale distributed systems [3], [7], [8].

In this model, the benefits of checkpointing are limited by the following factors: the runtime overhead (C), which is the time delay resulting from interruption of job execution to perform checkpointing; the network latency (L) (a time interval between the checkpoint generation and its availability on the CS); and the recovery delay (R), which is the time to download a failed job checkpoint from the CS to the CR, where the job is rescheduled to run. The L and R parameters are mainly determined by the available network capacities, the distance between the CS and the considered resource and checkpoint size. The values of both parameters could be reduced by applying checkpoint replication on multiple storage servers. However, this paper concentrates on the reduction of the checkpointing runtime overhead and therefore proposes several algorithms that differentiate the checkpointing interval (I) based on history statistics and current status of a particular job and its execution environment. By this means, we will, on one hand, eliminate unnecessary checkpointing and, on the other hand, introduce extra job state savings, where the danger of failure is considered to be severe. More specifically, the optimal checkpointing interval for a job j (I^j_{opt}) running on the computational node r depends on the following parameters: E^j_r is the execution time of j on the resource r (taking into account load of r); F_r is the average time between failures of r; and CS^j is the size of the j checkpoint. Additionally, the value of I^j_{opt} should satisfy the inequality $C + I^j_{min} < I^j_{opt}$ to be sure that jobs make execution progress despite of periodic checkpointing. I^j_{min} is the minimum checkpointing interval of j, which should be initialized with a default value, for example, a small percentage of E^j_r. It is considered that after the I^j_r interval expires, either the next checkpointing event can immediately be performed by the application, or a flag is set

TABLE 1
Listing of Used Acronyms and Symbols

S	Grid Site
CR	Computational Resource
UI	User Interface
GSched	Grid Scheduler
IS	Information Service
CS	Checkpoint Server
I_{GSched}	Scheduling interval
I_{IS}	System information propagation delay
WAN	Wide Area Network
LAN	Local Area Network
C	Checkpointing run-time overhead
L	Network latency
R	Checkpoint recovery delay
I	Checkpointing interval
I^j_{opt}	Optimal checkpointing interval for job j
I^j_{min}	Minimum checkpointing interval of j
E^j_r	Execution time of j on resource r
F_r	Mean time between failures of r
CS^j	Size of j checkpoint
LF_r	Last detected failure of r
t_c	Current system time
I^j_r	Customized checkpointing interval for j running on r
RE^j_r	Remaining execution time of j on resource r
MF_r	Mean failure interval of r
A_r	Computational availability of r
A_{grid}	Grid computational availability
T_{sim}	Total simulation time
Rep_{min}	Minimum number of job copies
Rep_{max}	Maximum number of job copies
CL	Lower bound on the number of active free CPUs
CA	Number of active CPUs
$Speed_S$	Speed or capacity of site
$Speed_r$	Speed or capacity of r
$MIPS_r$	(Million Instructions Per Second) speed of r
n_r	Number of jobs running on r
AR_j	Number of active replicas of j

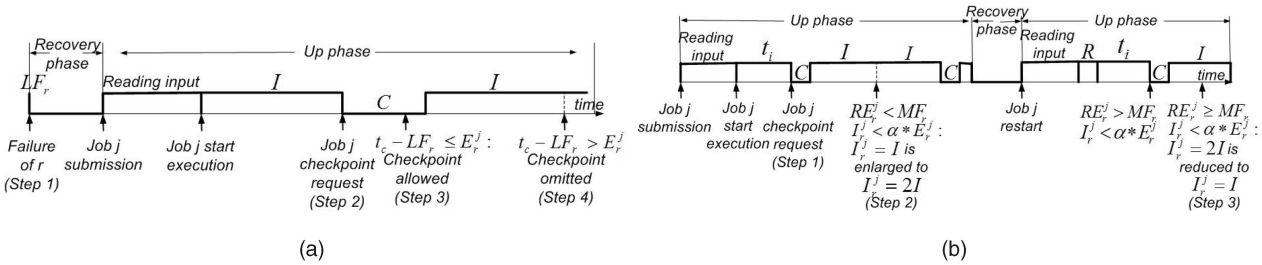


Fig. 2. (a) Operation of LastFailureCP on a resource running a single job. (b) Operation of MeanFailureCP on a resource running a single job.

indicating that the checkpointing can be accomplished as soon as the application is able to provide a consistent checkpoint. Furthermore, it is important to notice that deciding upon the job execution time is a complicated problem, often requiring an application-specific approach [30]. In our paper, the problem is simplified by assuming that the execution time can be exactly determined in advance. Therefore, the simulation results presented in the following sections show the upper bounds of the algorithms performance, with respect to this parameter. In the next sections, the proposed algorithms are discussed in more detail.

3.2 Last Failure Dependent Checkpointing (LastFailureCP)

The main disadvantage of unconditional periodic job checkpointing (PeriodicCP) is that it performs identically whether the job is executed on a volatile or on a stable resource. The goal of LastFailureCP is to reduce the overhead introduced by excessive checkpointing in relatively stable distributed environments, i.e., the algorithm omits unnecessary checkpoints of the job j based on its estimated total execution time and the failure frequency of the resource r to which j is assigned (see Fig. 2a). For each resource, the algorithm keeps a time stamp LF_r of its last detected failure (Step 1). When no failure has occurred, LF_r is initiated with the system start time. After an execution interval I, each job running on an active resource generates a checkpointing request (Step 2). The request is subsequently evaluated by the GSched and it is allowed only if the comparison $t_c - LF_r \le E_r^j$ evaluates to true (Step 3), where t_c is the current system time. As was previously mentioned, each checkpoint generation leads to runtime overhead C, which prolongs the execution of j (Step 3). If $t_c - LF_r > E_r^j$, the checkpoint is omitted to avoid the overhead as it is assumed that the resource is "stable" (Step 4). To prevent excessively long checkpoint suspension, a maximum number of omissions can be defined.

3.3 Mean Failure Dependent Checkpointing (MeanFailureCP)

Contrary to LastFailureCP that only considers checkpoint omissions, MeanFailureCP dynamically modifies the initially specified checkpointing frequency to deal with inappropriate checkpointing intervals (see Fig. 2b). The algorithm modifies the checkpointing interval based on the runtime information on the remaining job execution time (RE_r^j) and the average failure interval (MF_r) of the resource r where the job j is assigned, which results in a customized checkpointing interval I_r^j. The use of MF_r, instead of LF_r, reduces the effect of an individual failure event. While PeriodicCP and LastFailureCP are first run after the expiration of the predefined checkpointing interval, the MeanFailureCP activates checkpointing within a fixed and preferably short time period t_i after the beginning of a job execution (Step 1). The latter approach opens the possibility to modify the checkpointing frequency at the early stage of job processing. Each time the checkpointing is performed, I_r^j is adapted as follows: If $RE_r^j < MF_r$ and $I_r^j < \alpha \times E_r^j$, where $\alpha < 1$, the frequency of checkpointing will be reduced by increasing the checkpointing interval $I_r^{j_{new}} = I_r^{j_{old}} + I$ (Step 2). The first inequality in the condition ensures that either r is sufficiently stable or the job is almost finished, while the second limits the excessive growth of I_r^j compared to the job length. The latter can particularly be important for short jobs, for which the first condition almost always evaluates to true. On the other hand, when the abovementioned inequalities are not satisfied, it seems to be desirable to decrease I_r^j and thus to perform checkpointing more frequently $I_r^{j_{new}} = I_r^{j_{old}} - I$ (Step 3). When reducing the checkpointing interval, the following constraint should be taken into account: $C < I_{min} \le I_r^{j_{new}}$. I_{min} is a predefined value, which secures that the time between consecutive checkpoints is never less than the time overhead added by each checkpoint. In case of stable grid systems, it is desirable to choose relatively large values for I_{min} (5 percent-10 percent of the total job length) to prevent an undesirably steep decrease of the checkpointing interval. Experiments have shown that gradually incrementing I_r^j by I ensures rapid achievement of I_{opt} in most distributed environments. However, in case of rather reliable grids, the calibration of I_r^j can be accelerated by replacing I with a desirable percentage of the job execution time.

3.4 DSiDE Simulation Environment

Since grids are complex and often unpredictable environments, it is difficult to build a grid testbed on a realistic scale for validation and calibration of grid scheduling strategies. Therefore, the algorithms proposed in this paper were validated using a newly developed discrete event grid simulator named DSiDE. The main reason behind the development of DSiDE was a need for a simulator that provides maximum flexibility with respect to the implementation of dynamic behavior of grid resources and jobs. Regardless of the fact that there exists a broad spectrum of grid simulation frameworks, examples of which include GridSim [31], SimGrid [32], and NSGrid [33], none of them are well suited for this study because of their limited possibilities for modeling distributed system dynamics.

The DSiDE architecture is composed of two separate modules: DExec and DGen (Fig. 3). DExec implements a grid simulation environment, consisting of computational

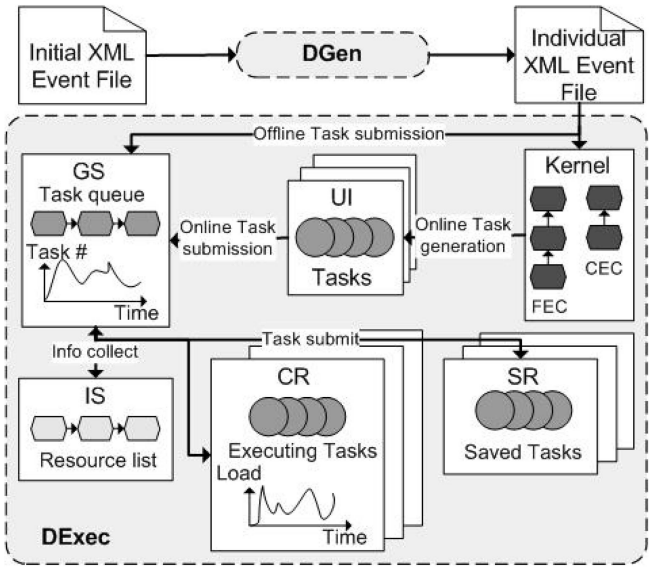


Fig. 3. The DSiDE simulator architecture.

and storage resources, and general grid services (UI, GSched, IS, and CS). Various scheduling policies can be plugged into the module in the form of Dynamically Linked Libraries (DLLs). DGen, in turn, is responsible for translating the initial, XML-based simulation scenario into a set of individual XML events. The initial specification allows the end user to create the desired grid model and a corresponding simulation scenario in a relatively short and convenient form. The DExec kernel is a standard discrete event kernel consisting of a clock and future (FEC) and current event chains (CEC). Each DSiDE event has a time stamp, i.e., an indication of the event execution time. This time stamp can be provided a priori or at runtime. In the latter case, a particular distribution (uniform, exponential, normal, etc.) of the event generation is specified. In addition to standard events, such as resource and job registration, DSiDE supports several types of dynamic system modifications, including alternating resource availability. The latter is modeled as a sequence of failure and restore events occurring with particular distributions. Failure events can either be correlated or independent. Therefore, the total grid resource availability, which is the percentage of time during which the resource performed useful computations, can be defined as

$$A_r = \left(\left(1 - \left(\sum_{n=1}^{N} \left(t_{r,n}^{f} - t_{r,n}^{r} \right) / T_{sim} \right) \right) \times 100 \right), \qquad (1)$$

where N is the number of resource failures; $(t_{r,n}^{f})$ and $(t_{r,n}^{r})$ are respectively the time stamp of the resource failure and restore; and T_{sim} is the total simulation time. From the individual resource availability, total grid availability is computed as follows:

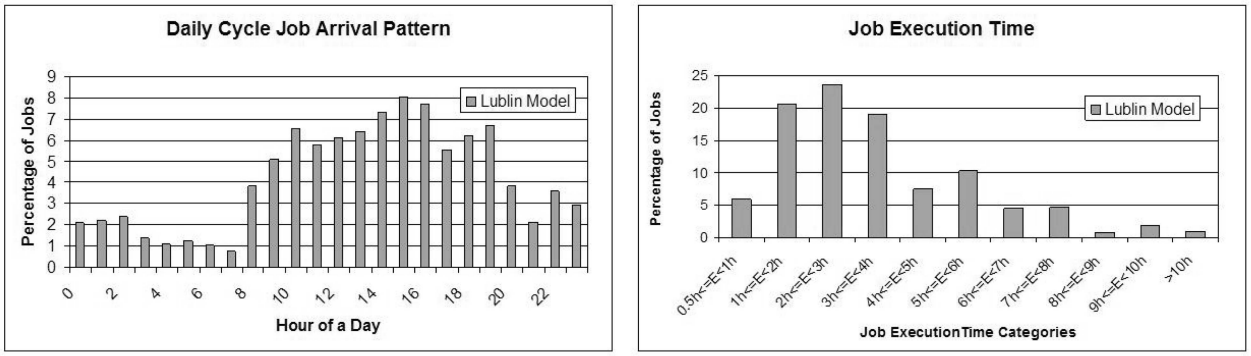
$$A_{grid} = \left(\left(\sum_{r=1}^{R} A_r \right) / (T_{sim} \times R) \right) \times 100, \qquad (2)$$

where R is the number of resources in the grid. Finally, DSiDE provides a set of events to specify network links and routes (sequence of links), which form the network model of the simulator. The DSiDE network is similar to the SimGrid network model, which differentiates between two types of links: WAN or Internet links, and LAN or intrasite links. WAN links are assumed to be fully interconnected with equal bandwidth assigned to each route going through them (a small fraction of the total bandwidth). On the other hand, intrasite links are always organized into a tree topology, and the available bandwidth is proportionally shared among the simultaneous active data transfers [34]. This simple model has proven to be a good approximation for real network behavior [35], while preserving relatively low computational complexity and short simulation times. Currently, it is assumed that the modeled network is fully reliable and thus failures can only originate from the connected CRs.

3.5 Simulation Results

To compare the performance of the proposed checkpointing heuristics, realistic workload, and system failure models, derived from production grid logs, were utilized. More specifically, the submitted workload follows the Lublin job generation model [36], where execution of batch jobs running on a single node was assumed. In this simulation scenario, the model parameters are initialized to represent a heavily loaded grid system with a daily cycle job arrival pattern. Fig. 4a depicts the proportion of jobs submitted into the grid system each hour, during an observation period of 24 hours. Apparently, most of the jobs (almost 80 percent) arrive during the daytime, while the remaining 20 percent are submitted between 9 p.m. and 7 a.m. The runtimes of the submitted jobs vary, as shown in Fig. 4b, where 11 categories of job lengths (from less than an hour to more than 10 hours) are differentiated. More than 80 percent of all submitted jobs have medium execution times, varying from 1 hour to 6 hours. Furthermore, to simplify the comparison between different algorithms, it is assumed that all jobs use inputs and outputs of 10 Mbytes; and a job checkpointing delay varies from 100 ms to 5 seconds, depending on the execution time.

The abovedescribed workload is submitted into the grid system discussed in Section 3.1. The system parameters are set as follows: WAN links have equal bandwidth of 100 Mbit/s and latency varying from 3 to 10 ms; CRs inside the sites communicate through LAN networks arranged into a star topology, with link bandwidths of 100 Mbit/s and latency of 1 ms; GSched is run every 5 minutes, while the longest propagation delay for the IS is initialized to 10 min; and finally, each CR has 1 MIPS CPU speed and is limited to process at most two jobs simultaneously. Failure and restore patterns of the grid resources follow the model represented in [3]. These models are constructed based on the analysis of failure data collected over the past nine years at the Los Alamos National Laboratory, which is currently one of the largest high-performance computing sites worldwide. In the grid environment considered, each of the four sites is modeled to posses different failure and restore behavior. Failure frequency ranges over the sites from several hours to several weeks and is modeled by a Weibull distribution with decreasing hazard rate. Mean repair time, in turn, varies across the sites from less than an hour to more than a day and is modeled by a logarithmic distribution. The considered grid system has a total availability of 90 percent.



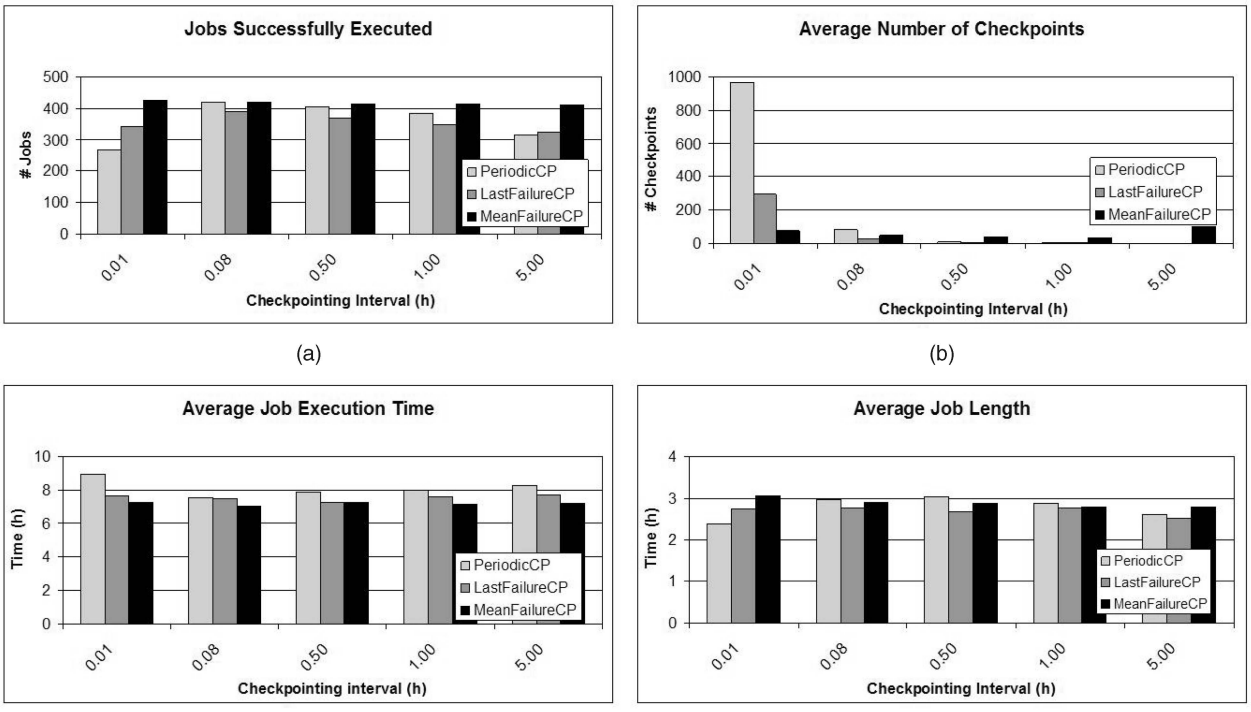
(a) (b)

Fig. 4. Lublin workload model. (a) Job arrival pattern with daily cycle. (b) Job execution time distribution.

The grid model described was observed during seven days of simulated time. Figs. 5a, 5b, 5c, and 5d show a comparison between the performance of the proposed dynamically adapting heuristics and the PeriodicCP approach for a randomly varying initial checkpointing interval (I). From the figures, it is clear that the efficiency of PeriodicCP strongly depends on the chosen value of I, which remains constant during the simulation. For instance, overly frequent and scarce checkpointing can result in up to 40 percent decrease in number of processed jobs, compared to the best achieved situation and significantly increase the average job execution time. Furthermore, in Fig. 5c, it can be observed that at high checkpointing frequencies, the average job length significantly decreases. This relates to the fact that exaggerated checkpointing substantially prolongs job execution, and therefore, only short jobs finish within the observed time interval. However, when I decreases and longer jobs can get processed, an increase in job runtime is in effect.

The results achieved with PeriodicCP are partially improved by LastFailureCP due to omission of redundant checkpoints. Apparently, the technique provides the best results for short checkpointing intervals. Since the algorithm does not consider checkpoint insertion, it performs slightly worse than PeriodicCP for large values of I. However, in the latter case, the effectiveness of LastFailureCP strongly depends on failure periodically. In the best case, when failures occur quite periodically and thus can easily be predicted by the algorithm, LastFailureCP will perform similar to PeriodicCP.

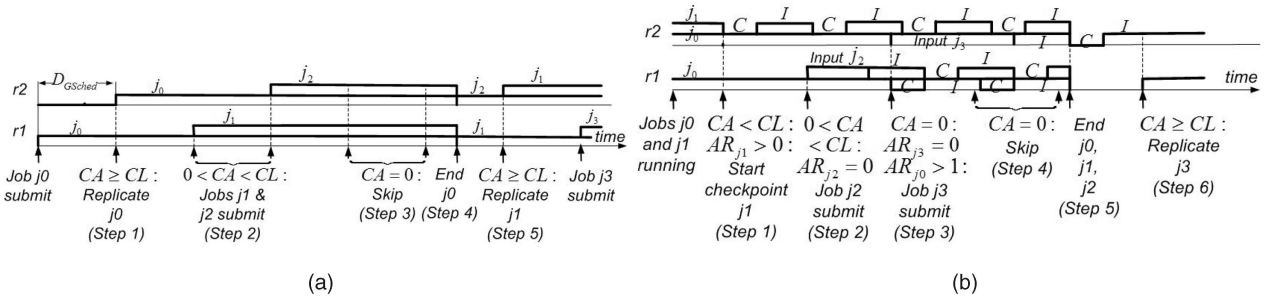
Finally, the fully dynamic scheme of MeanFailureCP proves to be the most effective. Starting from a random checkpointing frequency, it results in a number of executed jobs and average job runtime that are close to the results



(a) (b)

(c) (d)

Fig. 5. Checkpointing heuristics performance for varying initial checkpointing interval: (a) number of successfully executed jobs, (b) average number of checkpoints initiated by different heuristics, (c) job average runtime, and (d) job average length.



(a) (b)

Fig. 6. (a) Operation of LoadDependentRep on a distributed environment consisting of two resources, each able to run two jobs simultaneously. $Rep_{max} = 2$, $Rep_{min} = 1$ and $CL = 2$. (b) Operation of CombinedFT on a distributed environment consisting of two resources, each able to run two jobs simultaneously. $Rep_{max} = 2$, $Rep_{min} = 1$ and $CL = 2$. The PeriodicCP method is applied in the checkpointing mode.

achieved by PeriodicCP with the best performing checkpointing interval. It is important to notice the slight decrease in the number of checkpoints taken by MeanFailureCP as I is getting closer to the best performing checkpointing values. This decrease can be explained by a shorter calibration period required to achieve the "optimal" value of I. On the other hand, for large values of I, an increase in the number of generated checkpoints is observed, which is the consequence of the constraint $I_{min} \leq I_r^j < \alpha \times E_r^j$, where I_{min} and α were initialized with, respectively, $0.01 \times E_r^j$ and E_r^j. When I_r^j grows, this restriction evaluates to false for a larger part of jobs, and therefore, their adapted checkpointing interval starts to decrease, resulting in more checkpoints performed. As can be seen from the simulation results, this selective increase in checkpointing keeps the number of processed jobs and the average execution time of MeanFailureCP more or less constant, while in the case of the PeriodicCP and LastFailureCP algorithms, the performance drops considerably.

4 Replication-Based Heuristics

4.1 Load-Dependent Replication (LoadDependentRep)

Providing fault tolerance in distributed environments through replication has as an advantage that otherwise idle resources can be utilized to run job copies without significantly delaying the execution of the original job. Obviously, the more job copies are running on the grid, the larger is the chance that one of them will execute successfully. On the other hand, running additional replicas on a distributed environment with an insufficient number of free resources can considerably reduce throughput and prolong job execution. To deal with this dilemma, the proposed heuristic considers the system load and postpones or reduces replication during peak hours. The algorithm requires a number of parameters to be provided in advance, i.e., the minimum (Rep_{min}) and maximum (Rep_{max}) number of job copies, and the CPU limit (CL). The latter parameter specifies the lower bound on the number of active free CPUs for replication to take place. An example of the heuristic operation is shown in Fig. 6a, where the required parameters are initialized as follows: Rep_{min} and Rep_{max} are set respectively to 1 and 2; and CL is equal to two CPUs. In each iteration, the GSched consults the IS for the system status (Step 1). Based on this information, CA and CL are compared, where CA is the number of active CPUs able to execute the next job. The outcome of the comparison determines the choice for the next job to be scheduled:

- $\boldsymbol{CA \geq CL}$. Select a job j with the earliest arrival time stamp and the number of active replicas less than Rep_{max} (Step 1).
- $\boldsymbol{0 < CA < CL}$. Select a job j with the earliest arrival time stamp and the number of active replicas less than Rep_{min} (Step 2).
- $\boldsymbol{CA = 0}$. Skip the current scheduling round (Step 3).

However, even if the grid system is heavily loaded, it can be desirable to consider $Rep_{min} > 1$, since the failure rate of resources in distributed environments increases with the intensity of the workload running on them. When one of the job duplicates finishes, other replicas are automatically canceled (Step 4). If the system load decreases before the job was executed, the remaining $Rep_{max} - Rep_{min}$ replicas are activated (Step 5).

The algorithm assigns the selected job j to the site S with some free resources and with the smallest number of j replicas (Step 1, Step 5), since spreading replicas over different sites increases the probability that one of them will be successfully executed. If multiple sites have an equal number of job copies, a site that can provide for the fastest job execution is preferred. The speed or capacity of a site is defined as

$$\mathrm{Speed}_S = \left(\sum_{r \in S} \mathrm{MIPS}_r\right) \Big/ \left(\left(\sum_{r \in S} n_r\right) + 1\right), \qquad (3)$$

where Million Instructions Per Second (MIPS_r) is the speed of r, and n_r is the number of jobs on r. In the above equation, only resources executing no other replicas of j are taking into account. Distribution of similar replicas to a single CR is avoided because it is assumed that CPUs inside a single node have more chance to fail simultaneously in case of the resource malfunction. Therefore, inside the chosen site, the job will be submitted to the fastest available resource with no identical job replicas. If no such resource exists, the distribution of j is postponed, and the next job from the GSched queue is scheduled. The resource speed is determined by

$$\mathrm{Speed}_r = \mathrm{MIPS}_r/(n_r + 1). \qquad (4)$$

4.2 Failure Detection and Load Dependent Replication (FailureDependentRep)

To increase the fault tolerance of the previously discussed LoadDependentRep heuristic, the approach was combined

with a failure-detection technique. The principle of failure detection is straightforward: as soon as a resource failure is discovered by the GSched, all jobs submitted to the failed resource are redistributed. The algorithm proceeds as LoadDependentRep, except that in each scheduling round, not only newly arrived jobs are considered for submission to a CR, but also all jobs distributed to failed nodes. To construct a list of active CRs, GSched queries the IS. However, in order for a resource failure to be detected, the restore time should exceed $I_{IS} + I_{GSched}$. This means that although the method offers a higher level of fault tolerance compared to solely replication-based strategies, it does not ensure job execution.

4.3 Adaptive Checkpoint and Replication-Based Fault Tolerance (CombinedFT)

In this section, a combined checkpointing and adaptive replication-based scheduling approach is considered that dynamically switches between both techniques based on runtime information on system load. The algorithm can be particularly advantageous for grids with frequent or unpredictable alternations between peak hours and idle periods. In the first case, replication overhead can be avoided by switching to checkpointing, while in the second case, the checkpointing overhead is reduced by using low-cost replication. An example of the CombinedFT heuristic operation is shown in Fig. 6b, where the required parameters are initialized as follows: Rep_{min} and Rep_{max} are set respectively to 1 and 2; and CL is equal to two CPUs.

When the CPU availability is low $(CA < CL)$, the algorithm is in checkpointing mode (Step 1). In this mode, CombinedFT rolls back, if necessary, the earlier distributed active job replicas (AR_j) and starts job checkpointing. When processing the next job j, the following situations can occur:

- $\boldsymbol{AR_j > 0}$. Start checkpointing the most advanced active replica and cancel the execution of other replicas (Step 1).
- $\boldsymbol{AR_j = 0}$ and $\boldsymbol{CA > 0}$. Start j on the least loaded available resource within the least loaded site, determined respectively by (2) and (1) (Step 2).
- $\boldsymbol{AR_j = 0}$ and $\boldsymbol{CA = 0}$ and $\boldsymbol{\exists i : AR_i > 1}$. Select a random replicated job i if any, start checkpointing its most advanced active replica, cancel execution of other replicas of i, and submit j to the best available resource (Step 3).
- $\boldsymbol{AR_j = 0}$ and $\boldsymbol{CA = 0}$ and $\boldsymbol{\neg\exists i : AR_i > 1}$. Skip the current scheduling round (Step 4).

The algorithm switches to replication mode when either the system load decreases or enough resources restore from failure $(CA \geq CL)$ (Step 5). In replication mode, all jobs with less than Rep_{max} replicas are considered for submission to the available resources, in the order defined by the FailureDependentRep algorithm (see Section 4.1). When a job j is selected, it is assigned to the fastest resource (with no similar job replicas) connected to a grid site S with the maximum Speed_S and the smallest number of identical replicas. If j was previously in checkpointing mode and the replication completed successfully, the checkpointing of j is switched off (Step 6).

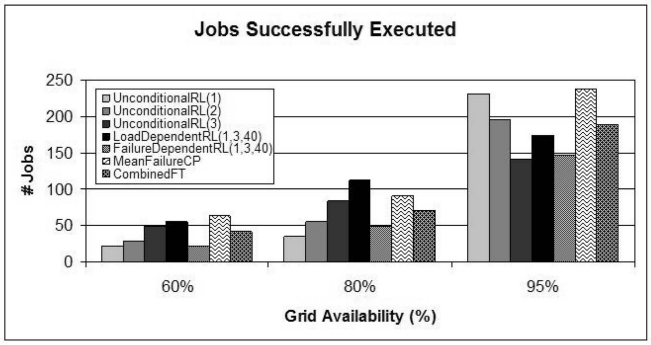
4.4 Simulation Results

In this section, the performance of the replication-based and hybrid approaches is compared against the performance of the best checkpointing heuristic (MeanFailureCP). The comparison is performed within grid systems with varying load and availability. Four replication algorithms are considered: UnconditionalRep(2), unconditional job replication with two job copies, UnconditionalRep(3), unconditional job replication with three copies, LoadDependentRL(1, 3, 40) adaptive replication with the minimum (Rep_{min}) and maximum (Rep_{max}) number of job replicas set to, respectively, 1 and 3, and the free CPU limit initialized to 40 (approximately 1/3 of the total grid capacity) FailureDependentRep(1, 3, 40), failure detection and adaptive replication-based algorithm with the same parameters as LoadDependentRep. Also, the performance of FCFS (or UnconditionalRep(1)) was observed to serve as a reference for comparison with the other algorithms. The combined approach (CombinedFT) is initialized with the same replication parameters as FailureDependentRep and switches in the checkpointing mode to the MeanFailureCP approach. The chosen parameter values for the replication-based heuristics are not necessarily optimal, but they are believed to be reasonable for the case at hand. The term "unconditional job replication algorithm" refers to an algorithm that sequentially processes jobs arriving to the GSched, based on the time stamp of their arrival. Independent of the current system load, the algorithm creates for each job a predetermined number of replicas that are assigned to different available resources, until all resources are filled.

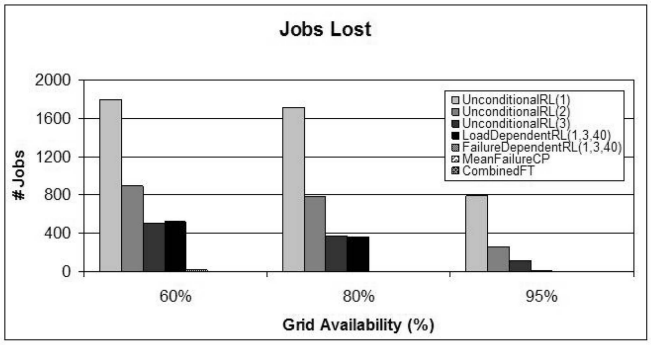
The algorithms are evaluated for high and low grid loads, which are accomplished by varying the job submission parameters of the Lublin model (see Section 3.5). In the case of high grid load, the same model parameters are utilized as the ones applied in the simulation scenario of the Section 3.5. In this scenario, about 7,000 jobs arrive into the system during the observation period of seven days (simulated time), which leads to long periods of system overload alternating with relatively short "idle" time intervals. In the case of low grid load, jobs are generated occasionally (about 700 during seven days of simulated time), while most of the time, a large part of the resources remains idle. To warrant low system utilization, also the average job length is reduced from 2.5 hours in the first scenario to 0.3 hour in the latter. The size of job input and output data in both simulation scenarios is set to 10 Gbytes to yield large data volumes often generated by real-world computationally intensive applications. Finally, varying system availability is achieved by modifying the parameters of the Weibull distribution within Schroeder and Gibson's model (for more details, see Section 3.5). It is again considered that each site possess different failure and restore patterns, with failure and restore intervals varying respectively from 2 hours to a week and from 30 minutes to a day.

Figs. 7a, 7b, 7c visualize the evaluated scheduling methods' performance on a highly loaded grid, while Figs. 8a, 8b, 8c summarize the results for low grid load. The following system parameters are observed: number of successfully executed/lost jobs, average job execution time, and average job length. It is Important to notice that a replicated job is assumed to be lost when all its replicas were started and afterwards failed.

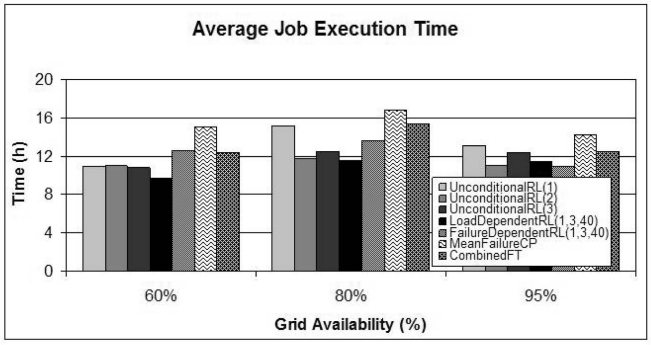
For heavily, as well as lightly loaded grids with relatively low availability, additional replication clearly provides better system performance and lower job loss rate. This is



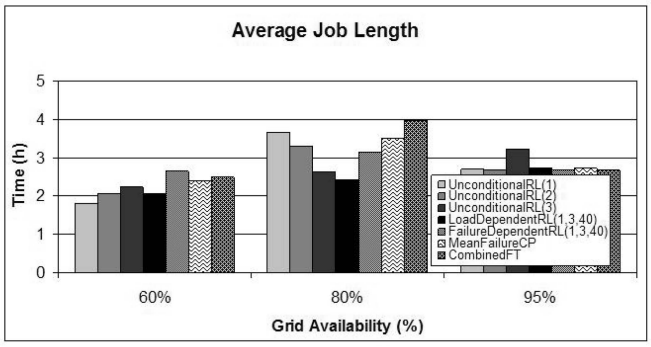
(a)



(b)

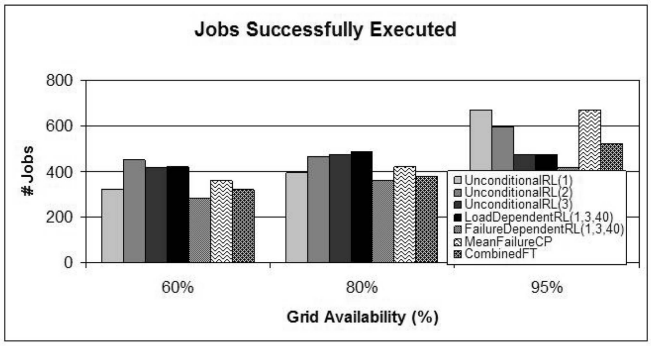


(c)

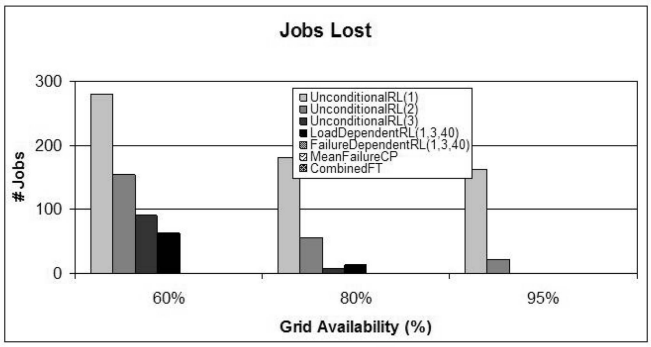


(d)

Fig. 7. Performance of replication-based, checkpointing-based, and hybrid algorithms on heavily loaded grids with varying availability: (a) number of successfully executed jobs, (b) number of jobs lost, (c) average job runtime, and (d) average job length.

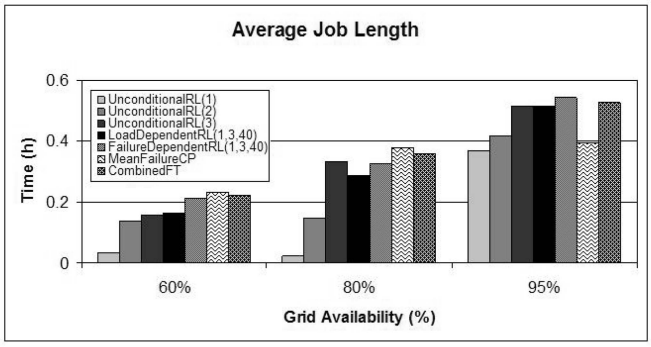


(a)



(b)

(c)



(d)

Fig. 8. Performance of replication-based, checkpointing-based, and hybrid algorithms on grids with low load: (a) number of successfully executed jobs, (b) number of jobs lost, (c) average job runtime, (d) average job length.

the consequence of the fact that replication-related overhead is compensated by increased grid reliability and consequently by a higher ratio of successfully executed jobs. However, as the grid availability improves (95 percent), additional replication provided by UnconditionalRep(2) and UnconditionalRep(3) leads to system throughput reduction. This reduction is getting more significant as the grid load increases and resources become more scarce. The

results for LoadDependentRep show that the performance of unconditional replication can be improved by postponing the execution of additional replicas during the peak hours. The higher the system load, the more gain can be achieved from the postponement (see Fig. 7a). On the other hand, for slightly loaded systems (see Fig. 8a), LoadDependentRep performs only slightly better than UnconditionalRep(2), since replication almost never has to be delayed. The main advantage of FailureDependentRep is certainly its high reliability in absence of sophisticated mechanisms for providing fault tolerance. Implementation of the algorithm requires only a replica counter and a simple job monitoring facility. Another benefit is that failure-sensitive long jobs have a higher chance to finally get processed due to the restart mechanism (see longer average job lengths in Figs. 7d and 8d). The disadvantage of FailureDependentRep is the slower grid performance (larger average execution times) as a result of the postponed replication in combination with the runtime overhead related to repetitive restart of failed jobs (see Figs. 7c and 8c). However, lightly loaded systems are less sensitive to this last disadvantage since, most of the time, enough computational resources are available, and thus, multiple job restarts do not penalize the execution of other jobs. In the condition of high load, the fully fault-tolerant MeanFailureCP results in the best system throughput compared to the other considered heuristics. This is the consequence of the considerable overhead introduced by the execution of additional replicas in an overloaded grid system. On the other hand, the average job execution time in case of the checkpointing approach is always relatively high, which leads to the algorithm performance reduction in the lightly loaded grid, where replication provides for almost costless fault tolerance. However, it is important to notice that the exact relation between the performance of the checkpointing and the replication-based solutions is largely determined not only by the system load but also by the runtime cost of checkpointing and the size of job input and output data. Finally, the throughput and average job execution times generated by CombinedFT for both types of system load are located, as can be observed in Figs. 7a and 7c, and Figs. 8a and 8c between, respectively, the throughputs and average job execution times of FailureDependentRep and MeanFailureCP. This is the logical consequence of the fact that job submissions are clustered in time and that the heuristic performs some calibrations, after each variation in the system load, before achieving its "optimal" state. Regarding the other observed performance parameters, CombinedFT is almost fully fault-tolerant and results in one of the best average job lengths among the considered algorithms.

5 Conclusion and Future Work

Fault tolerance forms an important problem in the scope of grid computing environments. To deal with this issue, several adaptive heuristics, based on job checkpointing, replication, and the combination of both techniques, were designed. The heuristics were evaluated in the DSiDE grid simulator under varying system load and availability. The results have shown that the runtime overhead characteristic to periodic checkpointing can significantly be reduced when the checkpointing frequency is dynamically adapted in function of resource stability and remaining job execution time. Furthermore, adaptive replication-based solutions can provide for even lower cost fault tolerance in systems with low and variable load by postponing replication in function of system parameters. Finally, the advantages of both techniques are combined in the hybrid approach that can best be applied when the distributed system properties are not known in advance.

In the following phase of our research, scheduling methods will be considered that adapt to dynamically changing estimations of job execution time.

References

[1] M. Chtepen, F. Claeys, B. Dhoedt, F. De Turck, P. Vanrolleghem, and P. Demeester, "Dynamic Scheduling of Computationally Intensive Applications on Unreliable Infrastructures," *Proc. Second European Modeling and Simulation Symp. (EMSS '06),* Oct. 2006.

[2] D. Feitelson, *Parallel Workloads Archive,* http://www.cs.huji.ac.il/labs/parallel/workload/, 2008.

[3] B. Schroeder and G. Gibson, "A Large-Scale Study of Failures in High-Performance-Computing Systems," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '06),* June 2006.

[4] S. Hwang and C. Kesselman, "A Flexible Framework for Fault Tolerance in the Grid," *J. Grid Computing,* vol. 1, no. 3, pp. 251-272, Sept. 2003.

[5] A. Subbiah and D. Blough, "Distributed Diagnosis in Dynamic Fault Environments," *Parallel and Distributed Systems,* vol. 15, no. 5, pp. 453-467, 2004.

[6] Y. Derbal, "A New Fault-Tolerance Framework for Grid Computing," *Multiagent and Grid Systems,* vol. 2, no. 2, pp. 115-133, 2006.

[7] Y. Zhang, M. Squillante, A. Sivasubramaniam, and R. Sahoo, "Performance Implications of Failures in Large-Scale Cluster Scheduling," *Proc. 10th Workshop Job Scheduling Strategies for Parallel Processing (JSSPP '04),* pp. 233-252, 2004.

[8] A. Oliner and J. Stearley, "What Supercomputers Say: A Study of Five System Logs," *Proc. 37th Ann. IEEE/IFIP Int'l Conf. Dependable Systems and Networks (DSN '07),* pp. 575-584, June 2007.

[9] A. Dogan and F. Osgunger, "Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing," *Parallel and Distributed Systems,* vol. 13, no. 3, pp. 308-323, 2002.

[10] D. Silva, W. Cirne, and F. Brasileiro, "Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids," *Proc. Int'l Conf. Parallel and Distributed Computing (Euro-Par '03),* pp. 169-180, Aug. 2003.

[11] R. De Camargo, A. Goldchleger, F. Kon, and A. Goldman, "Checkpointing-Based Rollback Recovery for Parallel Applications on the InteGrade Grid Middleware," *Proc. Second Workshop Middleware for Grid Computing (MGC '04),* pp. 35-40, 2004.

[12] A. Oliner, R. Sahoo, J. Moreira, and M. Gupta, "Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems," *Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '05),* Apr. 2005.

[13] Y. Li and M. Mascagni, "Improving Performance via Computational Replication on a Large-Scale Computational Grid," *Proc. Third Int'l Symp. Cluster Computing and the Grid (CCGrid '03),* May 2003.

[14] C. Bossie and P. Fiorini, "On Checkpointing and Heavy-Tails in Unreliable Computing Environments," *SIGMETRICS Performance Evaluation Rev.,* vol. 34, no. 2, pp. 13-15, 2006.

[15] S. Agarwal, R. Garg, M. Gupta, and J. Moreira, "Adaptive Incremental Checkpointing for Massively Parallel Systems," *Proc. 18th Ann. Int'l Conf. Supercomputing (SC '04),* Nov. 2004.

[16] S. Chakravorty and L. Kale, "A Fault Tolerance Protocol with Fast Fault Recovery," *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '07),* Mar. 2007.

[17] J. Young, "A First Order Approximation to the Optimum Checkpoint Interval," *Comm. ACM,* vol. 17, no. 9, pp. 530-531, Sept. 1974.

[18] E. Gelenbe, "On the Optimum Checkpoint Interval," *J. ACM,* vol. 26, no. 2, pp. 259-270, Apr. 1979.

[19] A. Tantawi and M. Ruschitzka, "Performance Analysis of Checkpointing Strategies," *ACM Trans. Computer Systems,* vol. 2, no. 2, pp. 123-144, May 1984.

[20] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio, "Min-Max Checkpoint Placement under Incomplete Failure Information," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '04),* June-July 2004.
[21] A. Oliner, L. Rudolph, and R. Sahoo, "Cooperative Checkpointing: A Robust Approach to Large-Scale Systems Reliability," *Proc. 20th Ann. Int'l Conf. Supercomputing (SC '06),* June-July 2006.
[22] A. Oliner and R. Sahoo, "Evaluating Cooperative Checkpointing for Supercomputing Systems," *Proc. 20th Int'l Parallel and Distributed Processing Symp. (IPDPS '06),* Apr. 2006.
[23] Y. Xiang, Z. Li, and H. Chen, "Optimizing Adaptive Checkpointing Schemes for Grid Workflow Systems," *Proc. Fifth Int'l Conf. Grid and Cooperative Computing (GCC '06),* Oct. 2006.
[24] P. Katsaros, L. Angelis, and C. Lazos, "Performance and Effectiveness Trade-Off for Checkpointing in Fault-Tolerant Distributed Systems," *Concurrency and Computation: Practice and Experience,* vol. 19, no. 1, pp. 37-63, 2007.
[25] Y. Li and Z. Lan, "Using Adaptive Fault Tolerance to Improve Application Robustness on the TeraGrid," *Proc. TeraGrid Conf.,* June 2007.
[26] C. Hou and K. Shin, "Replication and Allocation of Task Modules in Distributed Real-Timesystems," *Proc. 24th Int'l Symp. Fault-Tolerant Computing (FTCS '94),* June 1994.
[27] S. Choi, M. Baik, J. Gil, C. Park, S. Jung, and C. Hwang, "Group-Based Dynamic Computational Replication Mechanism in Peer-to-Peer Grid Computing," *Proc. Sixth IEEE Int'l Symp. Cluster Computing and the Grid (CCGRID '06),* May 2006.
[28] A. Ziv and J. Bruck, "Performance Optimization of Checkpointing Schemes with Task Duplication," *IEEE Trans. Computers,* vol. 46, no. 12, pp. 1381-1386, Dec. 1997.
[29] D. Pradhan and N. Vaidya, "Roll-Forward Checkpointing Scheme: A Novel Fault-Tolerant Architecture," *IEEE Trans. Computers,* vol. 43, no. 10, pp. 1163-1174, Oct. 1994.
[30] M. Hajdukovic, Z. Suvajdzin, Z. Zivanov, and E. Hodzic, "A Problem of Program Execution Time Measurement," *Novi Sad J. Math.,* vol. 33, no. 1, pp. 67-73, 2003.
[31] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *J. Concurrency and Computation: Practice and Experience,* vol. 14, nos. 13-15, Wiley, Nov.-Dec. 2002.
[32] A. Legrand, L. Marchal, and H. Casanova, "Scheduling Distributed Applications: The SimGrid Simulation Framework," *Proc. Third Int'l Symp. Cluster Computing and the Grid (CCGrid '03),* May 2003.
[33] P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt, and P. Demeester, "Evaluation of Grid Scheduling Strategies through NSGrid: A Network-Aware Grid Simulator," *J. Neural, Parallel and Scientific Computations,* special issue on grid computing, vol. 12, no. 3, pp. 353-378, 2004.
[34] F. Kelly, "Charging and Rate Control for Elastic Traffic," *European Trans. Telecomm.,* vol. 8, pp. 33-37, 1997.
[35] H. Casanova and L. Marchal, "A Network Model for Simulation of Grid Application," technical report, École Normale Supérieure de Lyon, Laboratoire de l'Informatique du Parallélisme, 2002.
[36] U. Lublin and D. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs," *Parallel and Distributed Computing,* vol. 63, no. 11, pp. 1105-1122, Nov. 1992, 2003.

Maria Chtepen received the MSc degree in computer science from Ghent University in 2004. She is a research assistant with the Fund of the Institute for Innovation and Technology (IWT), Department of Information Technology, Ghent University. Her main research interest include distributed systems and their performance optimization.

Filip H.A. Claeys received the MSc degree in computer science from Ghent University in 1992 and the MSc degree in artificial intelligence from K.U. Leuven in 1995. He currently works as a senior software engineer for MOSTforWATER N.V. and leads a research group on modeling software tools at Ghent University.

Bart Dhoedt received the PhD degree in engineering from Ghent University in 1995. In 1997, he became professor at the Faculty of Applied Sciences, Department of Information Technology. His research interests are software engineering and distributed computing. He is the author of approximately 150 peer-reviewed papers. He is a member of the IEEE and the IEEE Computer Society.

Filip De Turck received the PhD degree in electronic engineering from Ghent University. At the moment, he is a professor at the Department of Information Technology, Ghent University. He is author of 120 peer-reviewed papers. His main research interests include scalable and resilient software architectures and distributed computing. He is a member of the IEEE and the IEEE Computer Society.

Piet Demeester received the PhD degree from Ghent University in 1988. In 1992, he started a new research activity on broadband communication networks, resulting in the IBCN Group (INTEC Broadband communications network research group). In 1993, he became professor at Ghent University. His research activities cover various communication networks, including network and service management. He is the author of more than 500 publications and a member of the editorial board of several international journals. He is a senior member of the IEEE and the IEEE Computer Society.

Peter A. Vanrolleghem received the PhD degree. He is a bioengineer. He is a former head of the BIOMATH research team at Ghent University and has ample experience with modeling, monitoring, and control of wastewater treatment systems. He has more than 175 peer-reviewed papers and is very active within the International Water Association.